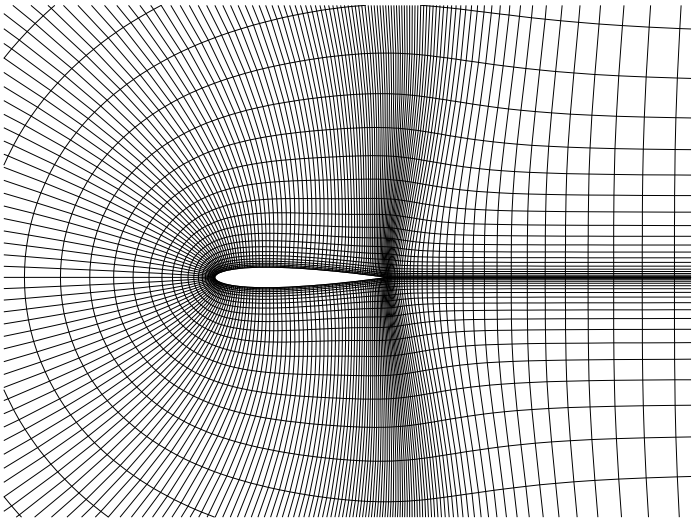
AE2220 II

Computational Modelling



S. J. Hulshoff
HSL 0.36, Aerodynamics Group
Faculty of Aerospace Engineering, TU Delft
S.J.Hulshoff@TUDelft.NL

2019 Version 1

Contents

1 Introduction **7**
- 1.1 Uncertainties and errors . . . 9
- 1.2 The computational modelling process . . . 10
  - 1.2.1 Modelling . . . 10
  - 1.2.2 Discretisation . . . 10
  - 1.2.3 Solution . . . 12
  - 1.2.4 Verification, *then* Validation . . . 14
- 1.3 What do we require from our model? . . . 14
- 1.4 What do we require from our discretisation? . . . 16
- 1.5 What do we require from our solution procedure? . . . 17
- 1.6 Final remarks . . . 18

2 Modelling with PDEs **19**
- 2.1 Why PDEs? . . . 19
- 2.2 Common PDEs in Science and Engineering . . . 20
  - 2.2.1 The linear diffusion equation . . . 21
  - 2.2.2 The linear advection equation . . . 22
  - 2.2.3 The linear advection-diffusion equation . . . 23
  - 2.2.4 The second-order wave equation . . . 24
  - 2.2.5 The non-linear advection (Burgers) equation . . . 25
  - 2.2.6 The Laplace equation . . . 26
  - 2.2.7 The Poisson equation . . . 27
  - 2.2.8 The equations of linear elasticity (optional) . . . 28
  - 2.2.9 The Navier-Stokes equations (optional) . . . 29
- 2.3 From PDEs to a computational model . . . 31
- 2.4 The classification of PDEs . . . 33
  - 2.4.1 Nomenclature for a general problem . . . 33
  - 2.4.2 PDE order and Homogeneity . . . 34
  - 2.4.3 Linear, quasi-linear, and non-linear PDEs . . . 34
  - 2.4.4 Hyperbolic, parabolic, and elliptic PDEs . . . 37
- 2.5 A first-order hyperbolic example . . . 38
- 2.6 A Second-Order hyperbolic example . . . 40
  - 2.6.1 Characteristic analysis . . . 40
  - 2.6.2 Boundary conditions . . . 41

2.7 A parabolic example . . . 43
2.7.1 Characteristic analysis . . . 43
2.7.2 Boundary conditions . . . 44
2.8 An elliptic example . . . 46
2.8.1 Characteristic analysis . . . 46
2.8.2 Boundary conditions . . . 46

3 Discretisation with the finite-difference method 47
3.1 From PDE to algebraic system . . . 47
3.1.1 Example: A transient 1D problem . . . 48
3.1.2 Example: A steady 2D problem . . . 51
3.1.3 Deriving arbitrary difference expressions: The Taylor table . . . 54
3.1.4 Applying conditions on boundaries . . . 56
3.1.5 Upwinding and artificial dissipation . . . 58
3.2 Dealing with irregular meshes . . . 62
3.2.1 Operators for unequal mesh spacing . . . 62
3.2.2 The generalised transformation . . . 63
3.3 Confirming consistency and stability . . . 65
3.3.1 Confirming consistency: The modified equation . . . 65
3.3.2 Confirming stability: Fourier analysis . . . 66

4 Verification 71
4.1 Code verification . . . 71
4.1.1 The method of manufactured solutions (MMS) . . . 72
4.1.2 The order-of-accuracy test . . . 73
4.2 Solution verification . . . 74
4.2.1 Richardson extrapolation . . . 74
4.2.2 Artificial boundary studies . . . 76

5 Discretisation with spectral and finite-element methods 77
5.1 Approximating the solution with functions . . . 77
5.2 The method of weighted residuals . . . 79
5.3 Equivalence of the strong and weak forms . . . 79
5.4 Solving for a_i; the Galerkin method . . . 80
5.5 Requirements for the weak form, w and ϕ . . . 82
5.6 An example spectral method . . . 85
5.6.1 The basic system . . . 85
5.6.2 Dirichlet boundary condition . . . 86
5.6.3 Neumann boundary condition . . . 87
5.6.4 The final system . . . 87
5.6.5 Observations and results . . . 88
5.7 An example finite-element method . . . 88
5.8 Convergence rates . . . 94
5.9 FEM in multiple dimensions . . . 94
5.9.1 Division of the domain into elements . . . 95

5.10 Unsteady problems . . . 97
5.10.1 Semi-discrete approach . . . 97
5.10.2 Fully-discrete approach . . . 98
5.11 Further Developments . . . 100

6 Time march methods 101
6.1 Accuracy of transient computations . . . 101
6.1.1 The exact amplification factor . . . 101
6.1.2 Definition of amplitude error . . . 102
6.1.3 Definition of phase error . . . 103
6.1.4 Errors of some example methods . . . 103
6.2 General analysis of linear methods . . . 106
6.2.1 From PDE to algebraic system . . . 106
6.2.2 The exact solution of the semi-discrete system . . . 107
6.2.3 The exact solution of the fully-discrete system . . . 111
6.2.4 The relation between λ_m and σ_m . . . 112
6.2.5 Choosing a time march . . . 113

7 Iterative solvers 121
7.1 A brief overview of direct solvers . . . 121
7.2 Iterative solvers: model problems . . . 123
7.3 Point iteration methods . . . 124
7.3.1 Jacobi iteration . . . 124
7.3.2 Gauss-Seidel iteration . . . 124
7.3.3 Over and under relaxation . . . 125
7.4 Analysis of iterative methods . . . 126
7.4.1 Fourier analysis . . . 126
7.4.2 Estimating the rate of convergence . . . 127
7.5 Line iteration methods . . . 129
7.5.1 Line Jacobi . . . 130
7.5.2 Line Gauss-Seidel . . . 130
7.6 Modern iterative methods . . . 130

A Notation 137
A.1 Analysis . . . 137
A.2 Symbols . . . 137
A.3 Abbreviations . . . 137
A.4 Function Spaces . . . 138

B Finite-volume methods 139
B.1 Finite-volume methods for fluid dynamics . . . 140
B.1.1 Choice of control volume . . . 142
B.1.2 Choice of flux evaluation . . . 142

C Common notation for SM/FEM **145**
C.1 $C^n(\Omega)$ 145
C.2 $L^2(\Omega)$ 145
C.3 $H^n(\Omega)$ 145
C.4 $\{\cdot|....\}$ 146
C.5 $(\cdot,\cdot)_\Omega$ 146
C.6 A precise statement of the sample problem 146

D Arbitrary finite-element geometries **147**
D.1 Isoparametric approach 147
D.2 Physical coordinate approach 149

Chapter 1

Introduction

Computational modelling is a broad term used to describe the representation of complex systems using digital computers. In this course we will focus on a specific area of computational modelling: the representation of complex physical systems by the numerical solution of partial differential equations. In the last few decades this area has revolutionised engineering by providing quantitative estimations for the effects of detailed design changes, which were previously only obtainable using physical experiments, or in the prototype stage. Consider, for example, the prediction of forces and moments on the swept wing shown in figure 1.1. Both the geometry and flow in this problem are complex, the latter

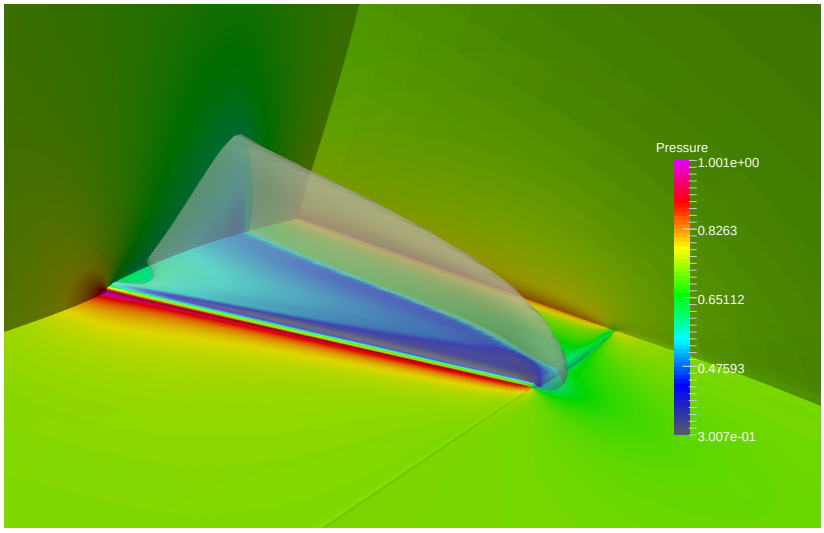


Figure 1.1: *Pressure distribution computed for the ONERA M6 wing operating in transonic conditions. On the wing surface an oblique and normal shock start near the root and intersect before the tip (near 80% span). A transparent Mach=1 isosurface is also shown.*

due to the appearance of multiple shockwaves and embedded supersonic regions. Testing large-scale models at transonic Mach numbers is time-consuming and expensive, and even then matching flight Reynolds numbers is usually infeasible. The emergence of computational models which can accurately predict forces for such conditions has thus had a large impact on both aerodynamic and aeroelastic design.

The first step in the process of computational modelling is to select an appropriate system of governing equations. For this example, a good choice is the Navier-Stokes equations for compressible flow, which describe the conservation of mass, momentum and energy:

$$\frac{\partial}{\partial t}\int_V U \; dV \; + \; \int_S \vec{F}\cdot\vec{n} \; dS = 0 \tag{1.1}$$

$$\text{or} \qquad \frac{\partial U}{\partial t} + \vec{\nabla}\cdot\vec{F} \; = \; 0 \tag{1.2}$$

Integro (1.1) and partial differential (1.2) forms of the Navier Stokes equations for compressible flows. Here $U = [\rho, \rho\vec{u}, \rho E]$ is the vector of conserved variables (mass, momentum and energy per unit volume) and $\vec{F}$ is the vector of fluxes of those quantities. (1.1) states the time rate of change of U in a volume V is determined by net flux through the surface, S. (1.2) can be obtained from (1.1) with the divergence theorem. To complete the system, an additional equation of state, such as the ideal gas law, is required (see appendix B for more details).

The Navier-Stokes equations can be solved analytically, but only for simple (typically 1D) problems. In order to take 3D geometric effects into account, we must resort to their approximate solution. The equations defined by (1.1) could be applied to any of an infinite number of control volumes which can be defined in the flow around the wing. Similarly, the equation defined by (1.2) apply for any of the infinite number of points in the flow around the wing. Current computing technology, however, is based on the arithmetic manipulation of a finite number of discrete values. Thus the next step of the computational modelling process is *discretisation*, for which the infinite-dimensional exact solution is exchanged for one based on a finite number of values, and then the governing equations are approximated using a system of algebraic equations relating those values. A common approach is to introduce a mesh covering the domain of interest (figure 1.2) and define the numerical solution using values at the mesh nodes or within the elements that surround them. Then algebraic equations approximating the governing equations can be derived using finite-difference, finite-volume or finite-element methods.

Computational modelling typically introduces several deficiencies in the representation of the physical system. These are introduced by the choice of PDEs used to model the system, the methods used to discretise the PDEs, and the methods used to solve the resulting algebraic system. In the remainder of this chapter, the computational modelling process will be described in more detail.

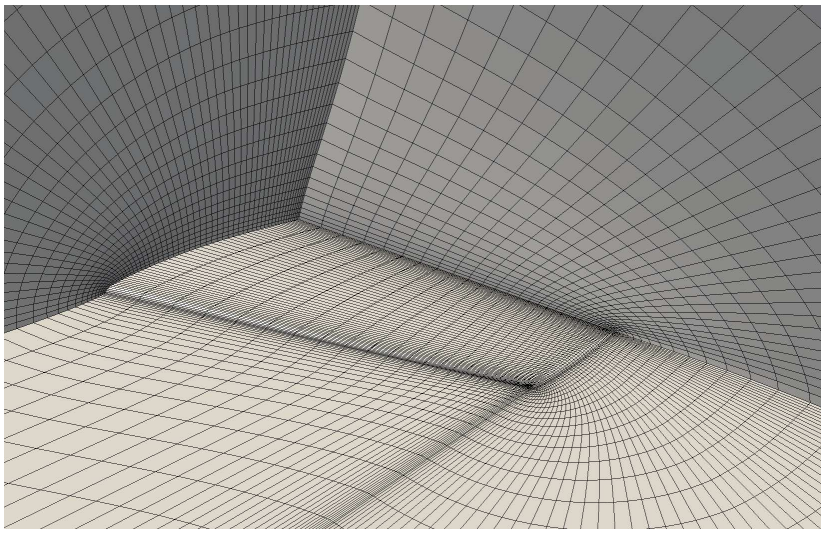


Figure 1.2: *Mesh for the ONERA M6 wing. Only mesh planes in the immediate vicinity of the wing are shown. The full mesh extends several spans away from the wing. The cell density is appropriate for an inviscid (Euler) computation. A viscous computation requires a much larger number of cells.*

Precise terminology will be introduced to describe the process, its deficiencies, and its desired behaviours.

1.1 Uncertainties and errors

In general, the deficiencies introduced by computational modelling may be categorised as either *uncertainties* or *errors* [1]:

Uncertainties

Uncertainties are deficiencies which can be attributed to lack of information about the system. There are two main types:

1. **Epistemic (systematic) uncertainties:** Those associated with deviations of the model from the real system due to unrecognised factors, such as the presence of unanticipated physical phenomena or non-linear behaviours.

2. **Aleatory (statistical) uncertainties:** Those which are impractical to measure and are therefore essentially irreducible. Examples include material imperfections or manufacturing defects, or the effects of atmospheric turbulence.

Errors

Errors are deficiencies in either the modelling or numerical solution procedures which are not due to lack of knowledge. These can be divided into:

1. **Model errors:** Those arising from the use of approximate governing equations to describe the system
2. **Discretisation errors:** Those associated with representing the infinite-dimensional solution of the governing equations with a finite number of algebraic relations
3. **Iteration errors:** Those due to the termination of iterative procedures used to determine the numerical solution
4. **Round-off errors:** Those arising from the finite precision with which computers carry out arithmetic

In the remainder of these notes we will primarily focus on methods for quantifying and controlling errors in computational modelling, rather than uncertainties [1]. In the next section we will discuss how these errors arise.

1.2 The computational modelling process

The complete computational modelling process is shown in figure 1.3. The terms shown in the figure will be described in detail below.

1.2.1 Modelling

Starting from the left, one begins by representing the physical system using one or more PDEs and boundary conditions. Since this inevitably will involve approximations of the processes defining and influencing the system's behaviour, both model uncertainties and model errors are introduced. Uncertainties arise if there are no PDEs known which sufficiently describe the process under consideration, or if it is not possible to consider all deviations in problem input parameters. Model errors, on the other hand, are normally associated with the use of approximate governing equations, such as simplified or linearised PDEs. Model errors also arise when using approximate boundary conditions, either to limit complexity, or to limit the extent of the physical domain which is considered.

1.2.2 Discretisation

Apart from their deficiencies, the chosen PDEs and their associated boundary conditions normally describe a continuous solution for all possible points in the

[1] It is interesting to note, however, that there has recently been much progress in Aleatory uncertainty quantification. The interested reader is referred to reference [19].

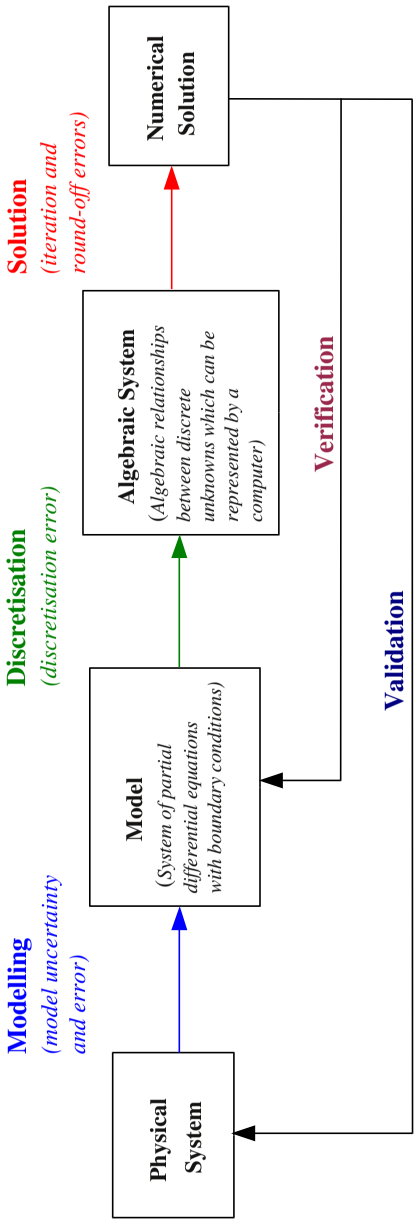


Figure 1.3: The computational modelling process

space/time domain of interest. Computers, on the hand, are most efficient when dealing with large finite-dimensional systems, e.g. those requring matrix-vector operations. Thus, the next step in computational modelling is *discretisation*, in which the continuous solution is replaced by a finite-dimensional vector of values called *unknowns* or *degrees of freedom*, and an *algebraic system* is derived which approximates the action of the PDEs and boundary conditions [2]. The later parts of this text will describe the two most popular approaches to discretisation, the finite-difference method (chapter 3) and the finite-element method (chapter 5) [3].

Discretisation clearly introduces additional deficiencies in the representation of the solution, called *discretisation errors*. These can arise, for example, from the inability of a mesh to represent the exact geometry, or from insufficient mesh refinement in regions where the physical solution has large gradients.

When the solution depends on time, discretisation is often performed in two steps. First a finite-difference, finite-volume or finite-element method is defined for only the spatial terms of the governing equations leading to a system of equations where both the discrete unknowns and their time derivatives appear. Then a separate finite-difference or finite-element method is chosen for the discretisation time, depending on the specific conditions under consideration. These time discretisations, known collectively as *time march methods*, can be developed independently, and a wide variety of them exist from the theory of the numerical solution of ODEs. Being able to choose them independently allows the computation to be tailored to the circumstances at hand, allowing reductions in computational cost when it is not necessary to resolve the highest frequencies of transient behaviour. The choice of time-march method will be discussed in chapter 6.

1.2.3 Solution

Many problems of interest have non-linear governing equations or boundary conditions, which lead to non-linear systems of algebraic equations. Such systems are typically solved using iterative methods (e.g. the Newton method). These start from an initial guess for the solution, then try improve the guess so that the residual[4] is minimised. The iterations terminate when the norm of the residual reaches an acceptably low tolerance. The fact that the (nonlinear) algebraic system is not solved exactly in such circumstances leads to another form of error, called *nonlinear iteration error* [5].

[2] A discrete representation of the solution is not required when using symbolic manipulation programs, but these have a limited range of application.

[3] Finite-Volume methods, which are applied to integral forms of conservation laws such as (1.1) are briefly described in appendix B.

[4] The residual is the vector left over when the guess is substituted into the system

[5] For unsteady nonlinear problems, some time march methods use several non-linear iterations each time step to ensure that the nonlinear iteration error remains sufficiently small. Other time march methods use time steps small enough so that the incurred nonlinear iteration error per step is small, which require nonlinear behaviours to be represented over several time steps

Iterative methods for nonlinear problems typically make use of a local linearisation of the nonlinear algebraic system, leading to a linear system of algebraic equations to be solved each iteration. On the other hand, the discretisation of linear problems normally leads to a linear algebraic system directly. In both cases, the use of discretisation methods such as the finite-difference or finite-element method produce systems with sparse matrices[6] (this will be demonstrated in chapters 3 and 5). Although such systems could in principle be solved using standard techniques from linear algebra, in practice their dimensions are so large that such techniques are very costly. However, sparse systems can be solved efficiently using iterative solvers. These start with a guess for the solution, and use the linear algebraic system to estimate a correction. The approximate solution is then updated, and the process is repeated until the corrections to the solution vector are sufficiently small. The error that remains due to the halting of the iteration process is called *linear iteration error*.

Thus in the solution of nonlinear computational models, it is common to have both an outer non-linear iteration method and an inner iterative solver. In a typical situation, one might perform several inner iterations to converge to the solution of the linear algebraic system before performing a non-linear update and correcting the solution of the nonlinear algebraic system (Algorithm 1).

Algorithm 1 Nested linear solver iterations in a non-linear solution method

```
while ( Solution change since last non-linear update > tolerance 1 ) do
  while ( Solution change since last linear solver iteration > tolerance 2 )
  do
    [ Update the solution of the linear algebraic system ]
  end while
  [ Complete the non-linear update (e.g. a Newton update) ]
end while
```

Finally, due to the fact that computers represent numbers with finite precision, performing algebraic operations leads to *round-off errors*. Their magnitude can be controlled by choosing how many bytes are used to represent floating-point numbers, although the chosen hardware and software also play a role. For a given hardware-software combination a parameter known as machine epsilon, ϵ_m, gives an upper bound on the relative errors incurred due to rounding in floating point arithmetic. Typical values are given in table 1.2.3. For most applications in solid and fluid mechanics, double precision provides enough accuracy so that the effects of round-off error are small compared to those of other types of error. If the problem at hand involves a very wide range of physical scales, however, higher levels of precision may be required.

[6] Sparse matrices are those for which most of the elements are zero

Precision	Bytes	Bits	$\epsilon_m \approx$
Single	4	32	6×10^{-08}
Double	8	64	1×10^{-16}
Quad	16	128	1×10^{-34}

Table 1.1: Typical values of machine epsilon for standard precisions

1.2.4 Verification, *then* Validation

At the end of the process a numerical solution is produced. At this stage two crucial parts of computational modelling take place, *verification* and *validation*.

Verification is an ordered procedure in which the consistency of the numerical method with the model equations is confirmed, and the magnitude of discretisation, iteration and round-off errors are estimated for the problem under consideration. Confirming consistency of the numerical method requires comparisons with an exact solution. As will be shown in chapter 4, complicated exact solutions can be "manufactured" for verification purposes by modifying the governing equations.

Estimating discretisation errors for a given problem can be done by comparing solutions with different levels of refinement, as will be described in chapter 4 [7]. Estimating iteration and round-off errors is relatively straightforward, as it simply requires comparing solutions obtained with different tolerances or levels of precision.

In validation, the numerical solution is directly compared with the output of the physical system. It is thus intended to provide estimates for model uncertainty and model error. Validation can only be performed after the process of verification, however, otherwise it becomes impossible to separate uncertainty and model errors from the other errors in the compuational modelling process. A description of validation methods falls outside the scope of this text, but an introduction to the main concepts can be found in [23].

The importance of verification and validation cannot be over-emphasised, since it is quite easy to generate poor solutions through the casual application of computational modelling techniques. A poor numerical solution is a very dangerous thing, as its veracity and utility will almost always be over-estimated.

1.3 What do we require from our model?

Naturally we would like to choose governing equations and boundary conditions which closely resemble the processes in the physical system. However, since the mathematical problem that results (the model) is distinct from physical system,

[7] There are also specialised error estimation procedures, such as those employing as adjoint-based techniques, which can be quite effective. Those interested in advanced error estimation can find an overview in [11].

it may also have additional solutions which do not occur physically. Furthermore, the exact solution of the model might exhibit extreme sensitivity to the input data, making it difficulty to ensure that its approximate discrete solution will resemble any physically realisable state. For a robust computational modelling procedure, we thus require the model to be "well behaved" in some sense. A strict mathematical interpretation of this requirement was provided by Hadamard (1902), who introduced the concept of a well-posed problem:

A problem is well-posed if its solution exists, is unique, and depends continuously on the input data.

By "unique" it is meant that there exists only one solution which satisfy the PDEs and their boundary conditions. The utility of this requirement is fairly obvious. By "depending continuously on the input data" it is meant that the solution responds proportionately to modifications in the input parameters of the problem. The latter is best illustrated by examples. Consider a problem where the interior solution changes dramatically depending on the wavelength of oscillating boundary data. Then for the discrete problem in which a large range of wavelengths are considered, the minimum wavelength introduced to the problem change as the mesh is refined. The result is solutions that depend completely on the chosen mesh. More common examples of ill-posedness occur in problems going backward in time. Because of the second law of thermodynamics, physical systems tend to destroy information about their original state. It is difficult to tell from a mixture, for example, which one of its two components was poured on top before mixing. Attempting to do so requires increasing precision as the starting time considered for the problem is increased.

In practice physical problems might have small number of non-unique solutions, or be "ill-posed" in the sense of the second example above. Normally the former can be dealt with by starting iterative solution procedures with an initial guess close to the desired physical solution. Dealing with the latter remains an interesting area of research. Common approaches in this area include those which increase the smoothness or "regularity" of the problem by slightly modifying the governing equations.

Within the scope of this course, however, we will use the above definition of a well-posed problem for identifying inappropriate combinations of PDEs and boundary conditions. More specifically, specifying an insufficient number of boundary conditions may lead to problems with an infinite number of solutions, while specifying too many may lead to problems where no solutions exist. We will require our models to be well-posed in the sense that these two situations are avoided. It is important to understand enough about the behaviour of PDEs so that ill-posed problems are avoided. The solutions to such problems are often unusable, even though their deficiencies are not always obvious.

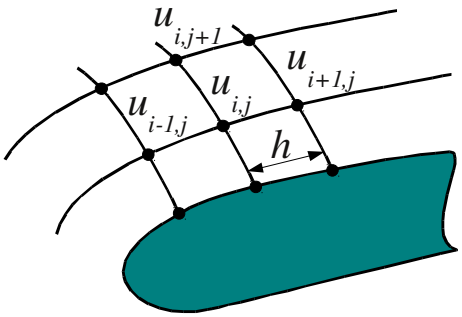


Figure 1.4: In finite-difference methods, one solves for the solution $u_{i,j}$ at selected points $x_{i,j}$ in the domain

1.4 What do we require from our discretisation?

The algebraic equations produced by the discretisation process are approximations of the equations defining the model, and exhibit distinct behaviours. Numerical solutions may thus differ substantially from exact model solutions. In order to reliably exploit a computational model, the discretisation must behave predictably. More precisely, we need our discretisations to be convergent, in the following sense:

A discretisation is convergent if its numerical solution approaches the exact model solution as its number of degrees of freedom is increased.

In a finite-difference method for example, the degrees of freedom are the solution values at the mesh nodes (figure 1.4). In this case convergence implies that as we add nodes, and thus decrease decrease the spacing, h, between them, we will obtain numerical solutions which are increasingly close to the exact model solution.

In the development of discretisations, it is common to break up the problem of verifying convergence into two sub-problems, the verification of *consistency* and *stability*. This is related to the following definitions:

A discretisation is consistent if when substituting an exact solution into the discrete equations the only terms which remain are those which tend to zero as the number of degrees of freedom is increased.

A discretisation is stable if the numerical solution for a given number of degrees of freedom is unique, and small changes to the input data produce only small changes in the numerical solution.

Using the above definitions, the condition for convergence can then be stated

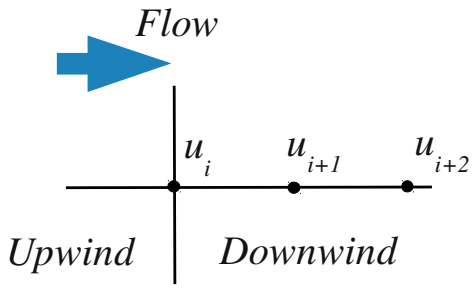


Figure 1.5: *Approximations for derivatives can be made using nodal solution values which are biased in a certain direction. For a flow problem, if this direction is purely downwind, the numerical solution becomes independent of the incoming data, leading to unphysical solutions. This would be the case if one evaluated the derivative at point i using only data from $i, i+1$ and $i+2$.*

using the **Lax equivalence theorem**:

For consistent discretisations of well-posed linear problems, stability implies convergence.

At first glance, consistency and convergence seem quite similar to each other. We will see, however, that one can make discretisations which are consistent but not stable, and therefore cannot be relied upon to produce better numerical solutions as the number of degrees of freedom is increased. An example would be where one uses approximations for the terms in a PDE which only take information from the downwind direction of a flow, when physically information is coming from the upwind direction. Such is the case in figure 1.5, where the derivatives in the PDE are only approximated using one-sided finite-difference formulas employing u_{i+1} and u_{i+2}. Although substitution of a known exact solution into such a formulation might produce left-over terms which get smaller as the mesh is refined, it could not be used to find a unique numerical solution as it does not account for information flowing from the upwind direction which physically should define the downstream flow. The formulation would then be consistent but not stable. Instability can also occur in the solution of unsteady problems, where it tends to have a more dynamic character. In this case an unstable discretisation may initially produce a bounded solution, but after some time its discretisation errors will amplify exponentially until the solutions becomes useless.

1.5 What do we require from our solution procedure?

Of course we would like to choose solution procedures which provide the numerical solution with minimum computational effort. When iterative procedures are

used, this can be interpreted has requiring convergence as quickly as possible. The use of the word convergence here is not coincidental. In fact, there is a strong analogy between iterative methods which update guesses of the numerical solution, and time-marching methods which advance an unsteady solution in time. We thus also require iterative methods to be both consistent and stable, and will analyse them using techniques similar to those used for analysing discretisation methods. This is discussed in more detail in chapter 7.

1.6 Final remarks

You will find out in this course that computational modelling is interesting (and fun!) because the choices involved in modelling, discretisation and solution are not trivial. Unwise choices can result in a lack of convergence, poor accuracy, or high computational cost. Wise choices, on the other hand, can lead to extremely useful solutions at moderate or even low cost.

Of course steady improvements in computer hardware have been a factor in making computational modelling common in both research and industry. But this is only half of the story. The last 40 years have also seen a revolution in the development of numerical discretisation and solution techniques, which has been just as important for the achievement of practical results. There are still many important problems which cannot be solved using today's hardware and numerical methods. Hopefully this course will inspire you to contribute to the advancement of this fascinating combination of mathematics, science and engineering.

Chapter 2

Modelling with PDEs

Partial differential equations (PDEs) are essential for the description of many physical systems. In this chapter we will briefly review several common types of PDEs, and then discuss how PDEs are combined with boundary conditions to develop practical computational models. In general the behaviour of a PDE strongly affects the choice of numerical solution procedure. To discuss such behaviours in precise terms, several systems of PDE classification will be introduced. These will separate PDEs based on their order, linearity and how information propagates in their solutions. Each of these factors can influence the choice of numerical scheme for both the interior and boundaries of a problem domain.

2.1 Why PDEs?

In engineering we often deal with problems that are inherently multi-dimensional. Determining the shear stress on the surface of an airfoil, for example, requires a description of the flow in the volume of air surrounding it. Similarly, predicting the local surface temperatures on the wall of a combustion chamber requires knowledge of the reactions occurring within the chamber volume. Examining the stresses within at the junctions of stress-skin constructions or predicting crack propagation in composites requires at least a 2D analysis. Even when the spatial domain is effectively 1D (e.g. for a long bar) we are often interested in processes which evolve in the time direction (e.g. the evolution of the bar's temperature if a heat source is applied to one end).

These problems define space-time regions of interest (domains) in which the physical state u varies in more than one direction $u = u(x, y, z, t)$. Furthermore, the physical processes which occur within such domains are usually dependent on the gradients of the physical state (e.g. $\frac{\partial u}{\partial t}$, $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial y}$, $\frac{\partial^2 u}{\partial x^2}$, $\frac{\partial^2 u}{\partial x \partial y}$) . For example, in problems where disturbances advect, the local the time rate of change of u, $\frac{\partial u}{\partial t}$, is normally a function of its first gradient in space, $\frac{\partial u}{\partial x}$. As a result, we require equations where derivatives appear simultaneously in more than one

direction (including time), or partial differential equations.

Normally PDEs can have many solutions. They become tied to a particular physical situation by the selection of a problem domain and the introduction of boundary conditions. Furthermore, for cases where more than one variable is required to describe the physical state, (e.g. both temperature and velocity), a system of PDEs might be required to determine the complete solution. Although it is tempting to simply rely on numerical techniques to produce solutions for more complex (systems of) PDE, the following sections will illustrate that in fact some basic knowledge of their properties is essential for obtaining reliable numerical results.

2.2 Common PDEs in Science and Engineering

In this section a number of basic PDEs will be introduced, and the behaviour of their solutions discussed. Then two systems of PDEs, from the fields of solid and fluid mechanics, will be presented. It will be shown how their more complex solutions can be interpreted using those of the basic PDEs.

2.2.1 The linear diffusion equation

Consider the problem of the evolution of temperature $u = u(x, t)$ within a long bar. In this case, the time rate change of u at any point x is determined by the difference between the temperature gradients to the left and right of the point. This results in the linear diffusion equation (also known as the heat equation):

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} \tag{2.1}$$

A solution of the heat equation for a space-time domain is shown for a sharp-pulse initial condition in figure 2.1. In this case the values at $x = 0$ and $x = 1$ are held constant. As the solution moves forward in time from the $t = 0$ boundary, it spreads out, removing the large initial gradients.

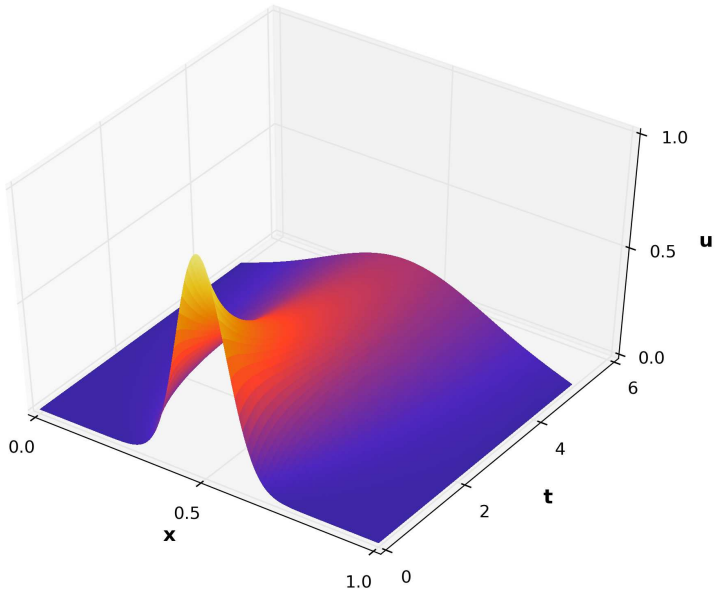


Figure 2.1: A solution for the linear diffusion (heat) equation

This can be anticipated by considering the negative time rate of change, $\frac{\partial u}{\partial t}$, predicted by the heat equation for the yellow area of the initial peak, where $\frac{\partial^2 u}{\partial x^2}$ is strongly negative. This type of diffusive behaviour is typical of processes where a substance moves from regions of high concentration to regions with lower concentrations. As such processes occur in many physical systems, linear diffusion terms are often present as a part of more complex PDEs.

2.2.2 The linear advection equation

The linear advection equation describes the unattenuated propagation of information. For one spatial dimension, it can be written:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{2.2}$$

A typical solution is shown in figure 2.2. This shows the evolution of u on a space-time domain, starting from a sharp pulse along the $t = 0$ boundary which maintains its shape as it advects in time to the right with speed c.

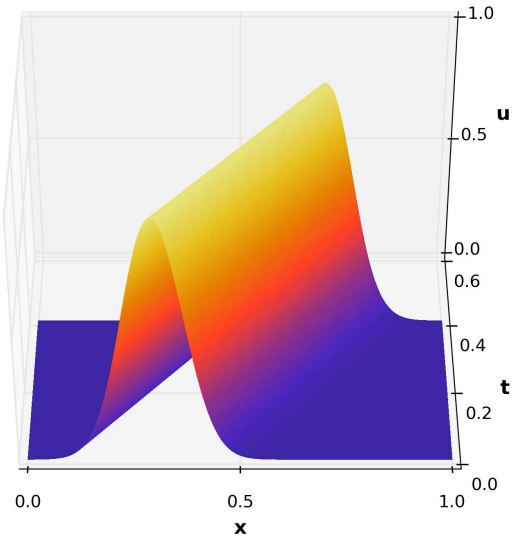


Figure 2.2: A solution for the linear advection equation

Terms similar to those in the advective equations are common in PDEs for fluids, where they represent the transport of quantities with the fluid velocity. They can also be used to represent the transmission of waves through gases and solids.

2.2.3 The linear advection-diffusion equation

The linear advection-diffusion equation is simple combination of the previous two PDEs:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{2.3}$$

which produces both behaviours. In other words, its solution describes the simultaneous transport and spreading out of the initial condition, as shown in figure 2.3.

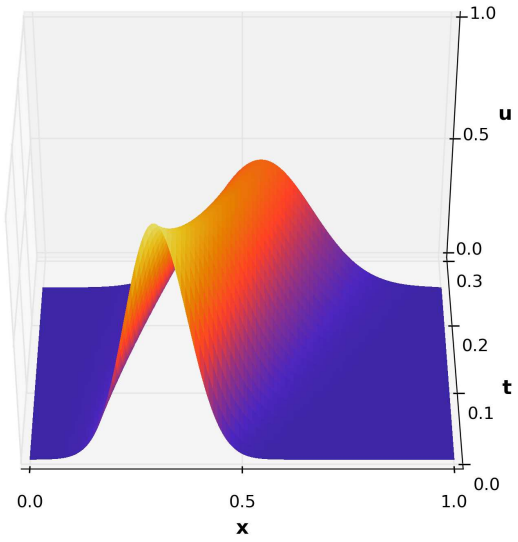


Figure 2.3: A solution for the linear advection-diffusion equation

The advection-diffusion equation is often used as a testbed for the development of numerical methods for more complex PDEs, such as the Navier-Stokes equations, which they resemble.

2.2.4 The second-order wave equation

The second-order wave equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \tag{2.4}$$

describes advection in two directions. In fact, we will show in section 2.6 that it can be rewritten as a system of two linear advection equations. A solution for the wave equation for a pulse initial condition is shown in figure 2.4. In this case, the pulse decomposes itself into two separate waves, which then propagate unattenuated to the left and to the right.

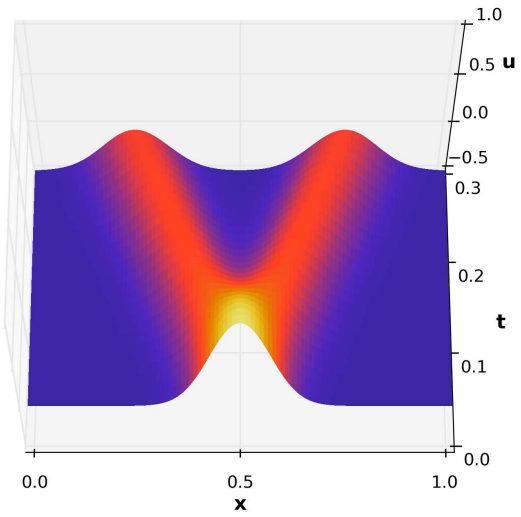


Figure 2.4: A solution for the second-order wave equation

The wave equation is used in the description of electromagnetism, supersonic flow and elastic wave propagation, among other applications.

2.2.5 The non-linear advection (Burgers) equation

The non-linear advection equation resembles its linear counterpart, except for the local advection velocity is equal to u.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 \tag{2.5}$$

A solution for the non-linear advection equation for a pulse initial condition is shown in figure 2.5. In this case the part of the solution to the right of the peak is travelling at a lower speed than the peak, so the peak catches up with it. This is known as wave steepening, and is part of the process of shock formation.

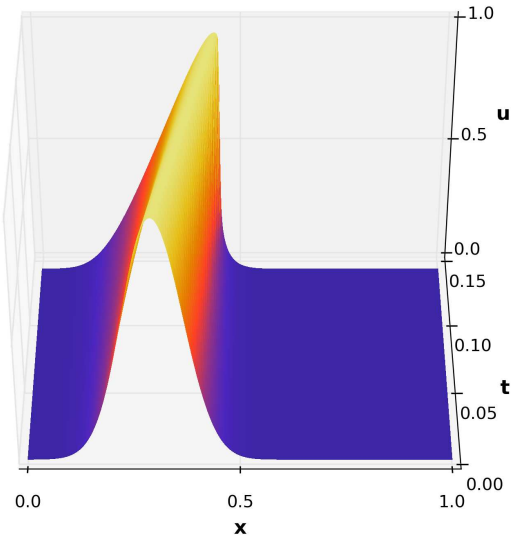


Figure 2.5: A solution for the non-linear advection (Burgers) equation

The Burgers equation is used to study traffic flow, and as a prototypical conservation law in theoretical and computational gasdynamics.

2.2.6 The Laplace equation

Laplace's equation, which is often used to describe incompressible potential flows, can be written in two dimensions as:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{2.6}$$

Solutions for the Laplace equation resemble steady-state solutions of the linear diffusion equation. In general the Laplace equation acts as a smoothing operator, averaging the solution between boundary values, as shown in figures 2.6 and 2.2.6.

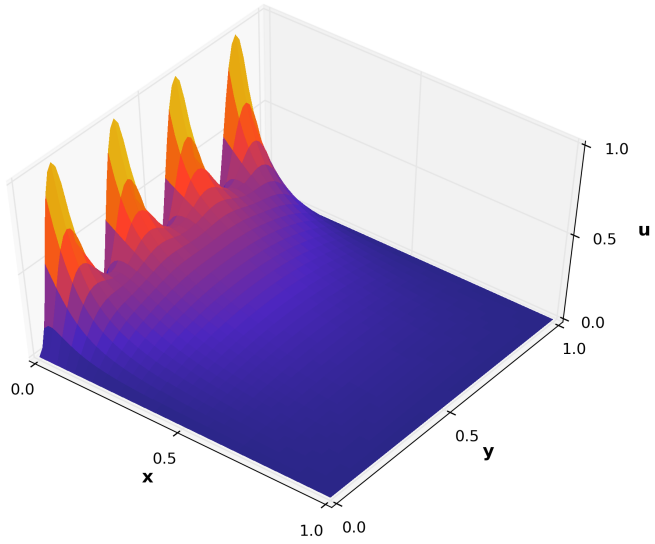


Figure 2.6: A solution for Laplace's equation with a $\sin^2(y)$ boundary condition

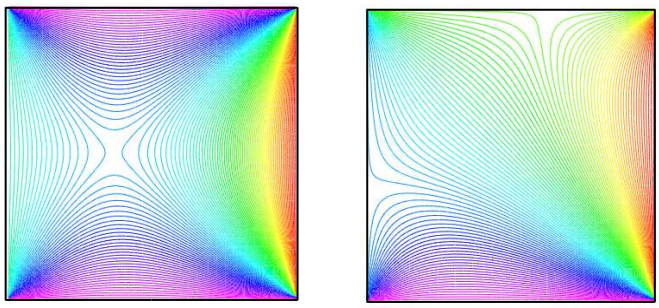


Figure 2.7: Laplace solutions for boundaries with different constant values

2.2.7 The Poisson equation

The Poisson equation is simply Laplace's equation with the addition of a source term, $S(x, y)$:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = S(x, y) \tag{2.7}$$

A solution shown in figure 2.8 is for a constant $S(x, y) = -1$. Here the values of u have been set to 0 on the boundaries of the domain. Since $S(x, y) = -1$ implies that the curvature of the solution is negative when proceeding radially, the values of u will reduce when moving away from the center of the domain. In contrast, specifying $S(x, y) = +1$ would give a solution that resembled a smooth valley.

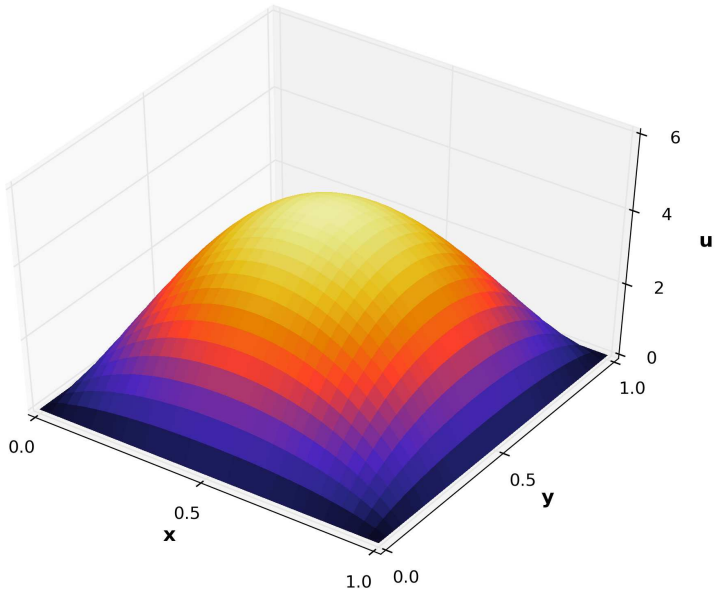


Figure 2.8: A solution for the Poisson equation

The Poisson equation $\nabla^2 u = -\rho$ appears in electrostatics, where u is the electric potential and ρ is the charge distribution. It is also used to model gravitational potential in astrodynamics.

2.2.8 The equations of linear elasticity (optional)

The small displacements of an elastic body subject to surface tractions and body forces can be predicted by the equations of linear elasticity. If we denote the local displacement vector by $u_i = u_i(x, y, z), i = 1, 2, 3$ the equations can be written:

$$\frac{\partial \sigma_{ij}}{\partial x_j} + f_i = \rho \frac{\partial^2 u_i}{\partial t^2} \tag{2.8}$$

$$\sigma_{ij} = \lambda \epsilon_{kk} \delta_{ij} + 2\mu \epsilon_{ij} \tag{2.9}$$

$$\epsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{2.10}$$

Where $\sigma_{\mathbf{ij}}$ is the stress tensor, f_i is the body force vector, ρ is the mass density and ϵ_{ij} is the strain tensor. The Lamé constants λ and μ are properties of the material, and can be written in terms of the Young's modulus E and Poisson's ratio ν as

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)} \tag{2.11}$$

A solution for a square bar with a discontinuous change in material properties halfway across its length is shown in figure 2.9. A force has been applied at the bar end, resulting in the steady-state response shown ($\frac{\partial^2 u_i}{\partial t^2} = 0$). The y component of displacement vector is plotted as contours on the surface of the bar, illustrating some of the intricacy that can be produced by even a relatively simple problem.

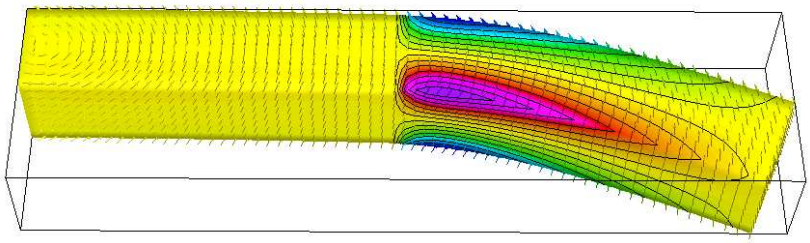


Figure 2.9: A solution for the equations of linear elasticity

For an isotropic and homogeneous material, the equations can be simplified to:

$$\mu \frac{\partial^2 u_i}{\partial x_j^2} + (\mu + \lambda) \frac{\partial^2 u_j}{\partial x_i \partial x_j} + f_i = \rho \frac{\partial^2 u_i}{\partial t^2} \tag{2.12}$$

The first and last terms produce behaviours similar to that of the wave equation. Correspondingly, a sudden displacement of the bar halfway along its length

results in waves travelling to the left and right. On the other hand, when $\frac{\partial^2 u_i}{\partial t^2} = 0$, the first term tends to smooth the solution, so that the interior displacement field resembles an interpolation of its boundary values.

2.2.9 The Navier-Stokes equations (optional)

The equations of mass and momentum conservation for an incompressible Newtonian viscous flow are:

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{2.13}$$

$$\rho \frac{\partial u_i}{\partial t} + \rho \frac{\partial}{\partial x_j}(u_i u_j) = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} \tag{2.14}$$

Here u_i is the velocity vector, p is the pressure, ρ is the density and τ_{ij} is the viscous stress tensor defined by:

$$\tau_{ij} = 2\mu S_{ij}, \qquad S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{2.15}$$

where μ is the molecular viscosity and S_{ij} is the strain-rate tensor. Using (2.13), (2.14) can also be written:

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} + \frac{1}{\rho}\frac{\partial p}{\partial x_i} - \nu \frac{\partial^2 u_i}{\partial x_j^2} = 0 \tag{2.16}$$

where $\nu = \mu/\rho$. This can be interpreted in terms of the simpler PDEs already introduced. Where the first two terms dominate, the process will resemble solutions from a multi-dimensional form of the Burgers equation. Where the third and fourth terms dominate, the process will resemble the solution of a Poisson equation for the velocity driven by local pressure gradients. Where the first and last terms dominate, the process will resemble a solution of the diffusion equation.

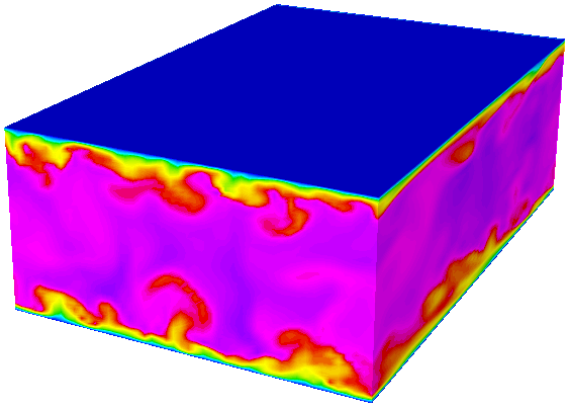


Figure 2.10: A solution for the Navier-Stokes equations

The nonlinearity introduced by the Burgers-like terms can introduce more than just wave steepening. In three dimensions, vortical interactions arising from this nonlinearity can quickly redistribute the flow's kinetic energy over a wide range of length scales. This results in solutions like that shown in figure 2.10 for the turbulent flow between two plates. The large number of scales in such flows can make them intractable, even for modern supercomputers ($\approx 10^6$ cores). The nonlinearity also introduces non-uniqueness to the solution. Interestingly, a low-drag laminar flow solution can also be obtained for the identical problem geometry, provided the initial condition is smooth enough. This provides us considerable motivation to understand and control the dynamic processes which lead to turbulent flows.

2.3 From PDEs to a computational model

Once an appropriate system of PDEs has been selected to model the physical process, it needs to be tied to a particular physical situation by the definition of a computational domain and the selection of appropriate boundary conditions. Since computational resources are limited, one naturally wishes to truncate the physical problem in order to select the smallest computational domain possible. Correspondingly, the boundaries which define computational domains may be divided into two basic types. The first type, which can be referred to as true boundaries, are those which have direct counterparts in the physical situation, such as the walls of a channel, or the surface of an airfoil (figure 2.11). The second type, known as artificial boundaries, are those which are introduced to limit the size of the numerical domain relative to the physical one. These include the inflow and outflow boundaries of a channel or the outer boundary of an airfoil problem.

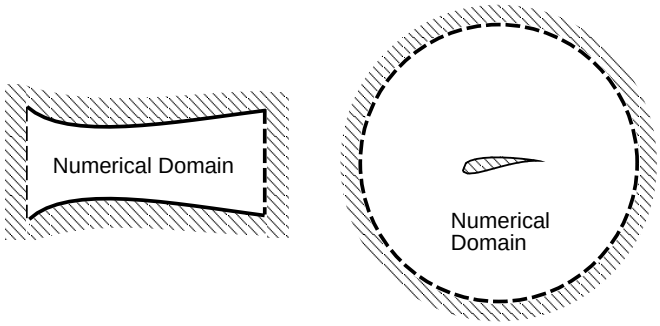


Figure 2.11: Numerical domains for channel flow (left) and an airfoil problem (right) with true boundaries (solid lines) and artificial boundaries (dashed lines)

The values which are specified on true boundaries are normally given directly in the definition of the problem. For the channel flow walls, one might specify a flow tangency or no-slip condition, for example. In order to determine solution values on artificial boundaries, a model of the processes occurring outside the computational domain is required. The quality of this model can have a considerable impact the computed results. For example, consider the case of the airfoil shown in figure 2.11. If the airfoil is generating lift, then there will be a finite velocity perturbation at the outer boundary which is initially unknown and therefore cannot be specified. One option is to place the outer boundary far enough from the airfoil so that the perturbations are small, and are modelled as being zero (figure 2.12, left). This may require the boundary to be placed more than 100 chord lengths away, which is expensive in terms of the number of unknowns used to represent the solution. An alternative is to use a more accurate model. For example, one could estimate the perturbations on the outer boundary using the velocities induced by a vortex with a circulation

equivalent to that determined for the airfoil by the computation (figure 2.12, right). This can allow for a substantial reduction in domain size relative to that required for a zero perturbation model.

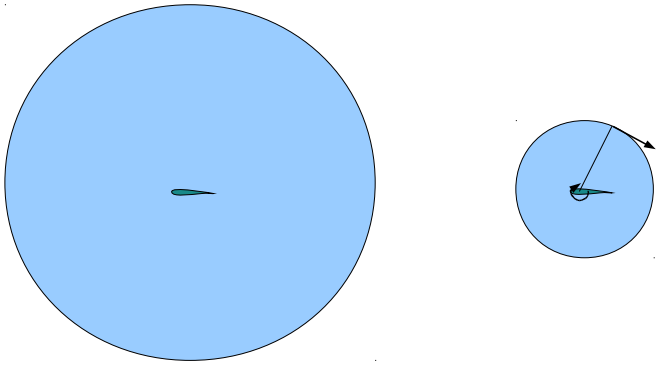


Figure 2.12: Reduction in domain size for airfoil problem using an outer boundary model (Not to scale; in 2D an outer boundary distance of order 100 chord lengths can be reduced to as little as 25 chord lengths)

Since the model for the processes occurring outside the computational domain will be an approximation, introducing an artificial boundary introduces an additional source of model error. The impact of this error must be must be evaluated by studying the sensitivity of the desired output of the computation to the size of the truncated domain.

On both true and artificial boundaries, the boundary values must be specified in a way that leads to a well-posed problem. For problems which involve the advection of information in the computational domain (for example in figures 2.2, 2.4 and 2.5) care must be taken that the advection continues undisturbed through an artificial boundary. This can be challenging when the PDE allows information to travel in more than one direction at a time. For example, consider the flow in a channel with a sound source as shown in figure 2.13. This could be described using the Euler or Navier-Stokes equations for compressible flows. In this case, some information is travelling into the domain via the inflow boundary, but some information is also travelling out of the domain, in this instance, acoustic waves from the sound source. Specifying the complete solution on the inflow boundary is clearly not allowed, since this would not be consistent with the as yet unknown variations which occur when the outgoing acoustic waves cross the boundary. In order for this problem to be well-posed, only part of the information at the boundary may be specified, and the rest must be determined by the solution of the PDEs used to model the processes within the domain.

A computational model of the process must respect the directions of information propagation as described above. For practical implementation purposes, however, explicit relations are required for all variables on the boundary. When these relations are based on the information which may be specified at

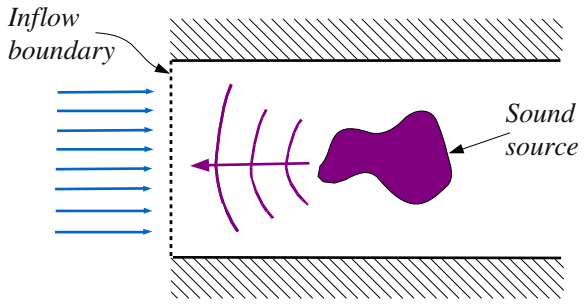


Figure 2.13: Channel flow with acoustic waves propagating from an interior sound source

a boundary , they are called *physical conditions*. When they are instead based on relations which approximate the PDE near the boundary, they are said to be *numerical conditions*. Both true and artificial boundaries may have either physical or numerical conditions. In the following sections, we will consider the classification of PDEs. This will provide more information on exactly where physical and numerical conditions are required.

2.4 The classification of PDEs

2.4.1 Nomenclature for a general problem

Before starting our discussion on the classification of PDEs, we will introduce some notation which will be used throughout these notes. As mentioned previously, it is normally desired to simulate some phenomena over a limited region of space. We will refer to this region as the problem domain, Ω, with boundary $\partial\Omega$ (figure 2.14).

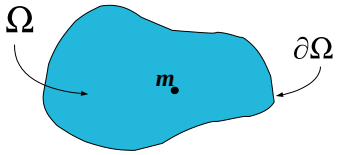


Figure 2.14: Notation for the computational domain

The solution of a PDE at all points m in Ω might be determined by a problem of the form:

$$\mathcal{L}(u) = f \quad \forall m \in \Omega \tag{2.17}$$

$$u = g \quad \forall m \in \partial\Omega \tag{2.18}$$

where in this case:

- $\mathcal{L}(u)$ is the differential operator of the PDE
- f is a (possibly variable in space and time) source term
- g is a (possibly variable in space and time) boundary value

For example, if one was solving the Poisson equation, $\mathcal{L}$ would correspond to the ∇^2 operator and $f = S(x, y)$ (See also the table of mathematical symbols given in appendix A). Equation (2.18) then corresponds to the direct specification of u on the boundary. This is referred to as the application of a *Dirichlet* condition. Alternatively, one might choose to specify the gradient on the boundary, q, which is referred to as the application of a *Neumann* condition. We can also specify a combination of these, r, resulting in a mixed or *Robin* condition. More formally we can write these conditions as:

- Dirichlet: $u = g$
- Neumann: $\frac{\partial u}{\partial n} = q$
- Mixed (Robin): $\frac{\partial u}{\partial n} + u = r$

The relations above are normally used for the specification of physical boundary conditions. In the context of discrete approximations, one may also require discretisations of the PDE near the boundary to provide solution values where physical conditions cannot be applied. These are referred to as numerical conditions:

- Numerical : $\mathcal{L}(u) = f$

and are typically formulated using a one-sided version of the discretisation procedure used in the interior of the domain.

2.4.2 PDE order and Homogeneity

The order of a PDE is simply equal to the order of the term within $\mathcal{L}(u)$ with the highest order derivative. Homogeneous PDE are those for which $\mathcal{L}(u) = 0$.

2.4.3 Linear, quasi-linear, and non-linear PDEs

Linear

A linear PDE is one for which the following is true:

$$\mathcal{L}(u_1 + u_2) \quad = \quad \mathcal{L}(u_1) + \mathcal{L}(u_2) \tag{2.19}$$

The reader can quickly verify that Laplace's equation is a linear PDE. Although it is often possible to find analytic solutions for linear PDEs, in many cases we wish to consider complex problem geometries for which no analytic solution

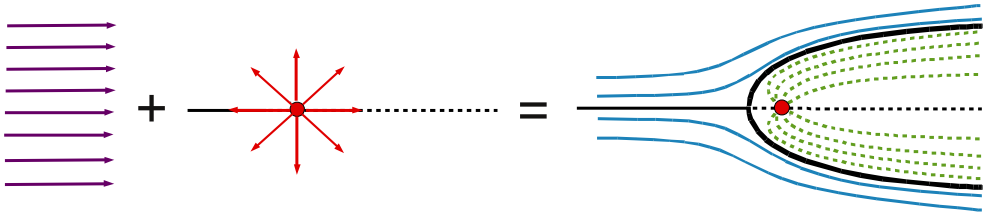


Figure 2.15: Potential flow solution for a Rankine half body obtained by the superposition of a freestream flow with a source

is available. If the PDE under consideration is linear and homogeneous, the principle of superposition can be used to construct such solutions in a very efficient manner.

When solving PDEs using the principle of superposition, we typically seek to find a combination of simple solutions which together represent the complex solution. If each of the simple solutions satisfies the PDE, then we no longer need to consider the PDE directly. Instead we need only find the combination of simple solutions which satisfies the problem's boundary conditions. An example of this is the solution to Laplace's equation for the flow around a Rankine half-body. This can be constructed simply by adding the expression for the potential of a freestream to that for a point source (figure 2.15). In this case we obtain a solution which satisfies the flow tangency boundary condition on the surface of the half-body (in black).

If we are interested in more general shapes, we can break the boundary into a finite number of panels (figure 2.16), and associate each panel with a simple local source or doublet solution of unknown strength. A system of equations can then be constructed by requiring flow tangency over each panel, and then used to solve for the unknown strengths. In the particular context of linear aerodynamics, this is known as a panel method, but more generally this is referred to as a boundary-element method. There also exists a number of alternative approaches to constructing boundary-element methods, some of which have been extended to non-linear problems (for a review, see [8]).

It is worth noting that when using boundary-element methods, the number of unknowns is relatively small, as degrees of freedom (unknown strengths) are only required for the boundaries of the domain. In contrast, when solving Laplace's equation using e.g. a finite-difference approach, unknowns are required both on the boundaries and in the interior of the problem domain (e.g. in a 3D cloud of points surrounding the body in figure 2.16). A disadvantage of boundary element-methods is that they usually produce full matrices, which can be expensive to solve relative to the sparse matrices produced by finite-difference and finite-element techniques. For many problems, however, the low number of degrees of freedom in boundary-element methods still provides considerable increases in efficiency. In general, when considering linear PDEs, one should

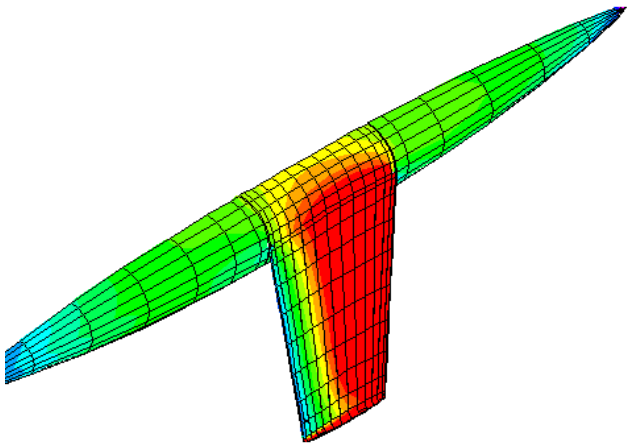


Figure 2.16: Pressure distribution for a wing-body geometry obtained from a panel method (CMARC example, Aerologic)

consider that it might be possible to develop effective numerical techniques by making use of superposition.

Quasi-linear

A quasi-linear PDE is one which does not have non-linearities occurring due to the highest-order derivatives and cross derivatives of its unknowns. A simple approach to determining if a PDE is quasi-linear is:

1. Underline the highest-order derivatives (w.r.t. x, y, etc.) and cross derivatives (w.r.t. x, y, x, x, y, etc.)

2. Replace all other terms and coefficients by constants (including the remaining derivative terms)

3. If the remaining operator is linear, the original operator is quasi-linear

For example, the following differential operator is quasi-linear:

$$\mathcal{L}(u) = x^2 \frac{\partial^2 u}{\partial x \partial y} + \ln(u) \frac{\partial^2 u}{\partial x^2} + \left(\frac{\partial u}{\partial x} \right)^2 \frac{\partial^2 u}{\partial y^2} \tag{2.20}$$

while the following operator is not:

$$\mathcal{L}(u) = \left(\frac{\partial u}{\partial x} \right)^2 + \frac{\partial^2 u}{\partial y^2} \tag{2.21}$$

since $\frac{\partial u}{\partial x}$ is the highest-order derivative in the x direction, and it appears squared.

Quasi-linear PDEs are capable of considerably more complex behaviours than their linear counterparts, including the development of discontinuities, and an extreme sensitivity to initial and boundary conditions leading to chaotic solutions. They differ from non-linear PDEs, however, in that it is generally easier to find analytic solutions for quasi-linear PDEs.

Non-linear

If a PDE is neither linear nor quasi-linear, then it is non-linear. Analytic solutions for non-linear equations are rare, and tend to be specialised to specific problems. Therefore, numerical techniques are particularly useful for finding general non-linear solutions.

Since a wide range of PDEs is encountered in practice, we will consider discretisation methods such as the finite-difference and finite-element methods which are not limited to linear or quasi-linear operators. In the remaining chapters, however, we will often consider examples based on linear PDEs, since these allow the straightforward analysis and interpretation of results.

2.4.4 Hyperbolic, parabolic, and elliptic PDEs

Some PDEs have *characteristics*, which are lines or surfaces along which their solution can be descibed by ODEs. Characteristics not only aid in determining the solution, but also make the behaviour of a PDE easier to interpret. PDEs are classified as hyperbolic, parabolic and elliptic based on the number of characteristics they have. Hyperbolic PDEs have sufficient characteristics to allow their solution to be completely described in terms of ODEs. Thus, the solution of a hyperbolic PDE at any given point is determined by information coming from a finite number of directions. Parabolic PDEs also have characteristics, but these are insufficient in number to fully describe the solution. They can still be used, however, to help establish the zones of dependence of the solution at a given point. An elliptic PDE has no characteristics, which implies that its solution at each point is dependent on information from all directions. This also implies that the solution at each point is dependent on the solution at all other points in the domain.

The nomenclature comes from the study of a general second-order linear PDE:

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} = f \tag{2.22}$$

for which it has been observed that when:

$$b^2 - 4ac > 0 \tag{2.23}$$

the equation is hyperbolic, while for

$$b^2 - 4ac = 0 \tag{2.24}$$

the equation is parabolic, and for

$$b^2 - 4ac < 0 \tag{2.25}$$

the equation is elliptic. The names hyperbolic, parabolic and elliptic have been chosen in analogy with the general second-order equation of analytic geometry.

Since the definitions (2.23), (2.24) and (2.25) are restricted to PDEs of the form (2.22), however, we will generally use the number of characteristics found to classify PDEs as hyperbolic, parabolic or elliptic:

- A PDE is hyperbolic if it has a complete set of characteristics
- A PDE is parabolic if it has an incomplete set of characteristics
- A PDE is elliptic if it has no (real) characteristics

where by complete it is meant that sufficient characteristics exist to allow the definition of a system of ODEs which are equivalent to the original PDE. Example characteristic analyses will be given later in this chapter. Generally these will follow the procedure below:

1. Rewrite the equation as a system of first-order PDEs
2. Attempt to find characteristics along which the system can be written as ODEs
3. Classify the PDE based on the number of characteristics found

In the case of hyperbolic PDEs, the system of ODEs obtained using the procedure above can be used in an analytic solution method known as the *method of characteristics*.

2.5 A first-order hyperbolic example

As illustrated in section 2.2.2, the linear advection equation:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{2.26}$$

describes the propagation of an initial condition with speed c. This indicates that the equation may be hyperbolic. To show that it is, we follow the procedure given at the end of section 2.4.4. In this case, however, we have a single first-order PDE, so we can skip the first step.

To define the characteristics, we look for lines defined by $x(s), t(s)$ along which the PDE can be written as an ODE. Here s is a parameter which varies along the lines. Using the chain rule, we can write:

$$\frac{du}{ds} = \frac{\partial u}{\partial t}\frac{dt}{ds} + \frac{\partial u}{\partial x}\frac{dx}{ds} \tag{2.27}$$

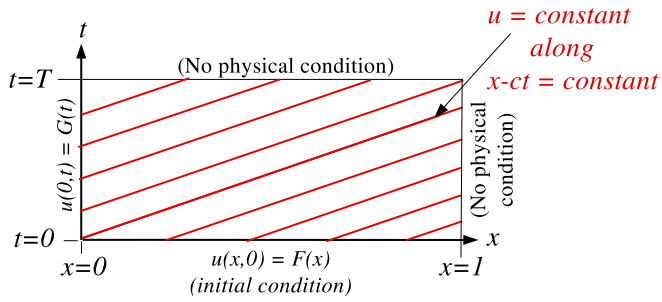


Figure 2.17: Characteristic lines for the linear advection equation

If we choose:

$$\frac{dt}{ds} = 1 \tag{2.28}$$

$$\frac{dx}{ds} = c \tag{2.29}$$

then we have

$$\frac{du}{ds} = \frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{2.30}$$

which results in a particularly nice ODE, which states u will be constant along s. Equations (2.28) and (2.29) form a parametric definition for s. Integrating we have

$$t = s + C_1 \tag{2.31}$$

$$x = cs + C_2 \tag{2.32}$$

Substitution of the first of the above relations into the second shows that the s direction defines a family of lines given by by $x = ct + \text{const}$. These are shown in red in figure 2.17. For this linear first-order PDE, we could arrive at a similar conclusion by inspecting the operator form:

$$\left(\frac{\partial}{\partial t} + c\frac{\partial}{\partial x}\right) u = 0 \tag{2.33}$$

and then considering a change in the $x-t$ plane consisting first of a step of $c\delta$ in x followed by a step of δ in t (figure 2.18). If δ is very small, (2.33) implies a net zero change in u, which defines the same direction for s as found by examining the parametric relations.

With the s direction defined, the solution in the entire field can be found by combining the equation

$$\frac{du}{ds} = 0 \tag{2.34}$$

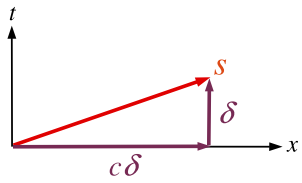


Figure 2.18: Characteristic lines of the linear advection operator

with appropriate boundary conditions. Thus we have replaced our PDE with an equivalent ODE, and since this is sufficient to solve for our single unknown u, the original PDE was hyperbolic.

For most time-dependent problems, we begin by specifying the initial state of the system, and then find the evolution of the system in time. The fact that u is constant along lines in the s direction therefore has implications for the type of boundary conditions that can be applied. Examining figure 2.17 we see that if our normal interpretation of causality holds (the future is a function of the past but not vice versa), we can only specify u on the $t = 0$ boundary and the left $x = 0$ boundary. The values on the right $x = 1$ and top $t = T$ boundaries are determined by the fact that u must remain constant along the characteristic lines. Specifying other values on the $x = 1$ or $t = T$ boundaries as physical conditions would lead to an ill-posed problem. If one is lucky this will cause the software used for the computational model to fail, giving a clear indication of the inconsistency. Unfortunately this not always the case.

2.6 A Second-Order hyperbolic example

Like for the linear advection equation, we could suspect that the second-order wave equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \tag{2.35}$$

which describes the propagation of two waves in a one-dimensional domain, is hyperbolic. This is also implied by comparing (2.35) to (2.22).

2.6.1 Characteristic analysis

To prove that second-order wave equation is hyperbolic, we again follow the procedure given at the end of section 2.4.4. For step 1, we introduce the variables $r = \frac{\partial u}{\partial t}$, $p = \frac{\partial u}{\partial x}$, allowing us to express (2.35) equivalently as:

$$\frac{\partial r}{\partial t} - c^2 \frac{\partial p}{\partial x} = 0, \tag{2.36}$$

$$\frac{\partial p}{\partial t} - \frac{\partial r}{\partial x} = 0. \tag{2.37}$$

The first of these equations is simply a re-statement of the PDE. The second equation states $u_{xt} = u_{tx}$, which is true for any continuous function. Consider a linear combination of the two equations:

$$\left(\frac{\partial r}{\partial t} - c^2\frac{\partial p}{\partial x}\right) + \alpha\left(\frac{\partial p}{\partial t} - \frac{\partial r}{\partial x}\right) = 0 \tag{2.38}$$

which can also be written:

$$\left(\frac{\partial}{\partial t} - \alpha\frac{\partial}{\partial x}\right) r + \left(\frac{\partial}{\partial t} - \frac{c^2}{\alpha}\frac{\partial}{\partial x}\right)(\alpha p) = 0 \tag{2.39}$$

Our objective is to find characteristics along which this partial differential equation becomes an ordinary differential equation. This will be the case when r and αp in (2.39) are differentiated along the same direction in space, or in other words, when the two differential operators in the large brackets are equivalent. Requiring their equivalence leads to two possible values for α:

$$\frac{\partial}{\partial t} - \alpha\frac{\partial}{\partial x} = \frac{\partial}{\partial t} - \frac{c^2}{\alpha}\frac{\partial}{\partial x} \quad \Rightarrow \quad \alpha = \frac{c^2}{\alpha} \quad \Rightarrow \quad \alpha = \pm c. \tag{2.40}$$

The wave equation can thus be re-written as the system of ODEs defined by $\alpha = \pm c$:

$$\alpha = c: \qquad \frac{\mathrm{d}}{\mathrm{d}s_1}(r + cp) = 0, \qquad \frac{\mathrm{d}}{\mathrm{d}s_1} \equiv \frac{\partial}{\partial t} - c\frac{\partial}{\partial x}, \tag{2.41}$$

$$\alpha = -c: \qquad \frac{\mathrm{d}}{\mathrm{d}s_2}(r - cp) = 0, \qquad \frac{\mathrm{d}}{\mathrm{d}s_2} \equiv \frac{\partial}{\partial t} + c\frac{\partial}{\partial x}. \tag{2.42}$$

Since we have found two ODEs, and we have only two unknowns (r and p), we have sufficient equations to re-write our PDE as a system of ODEs. The wave equation is therefore indeed hyperbolic. The system of ODEs says that the quantity $(r + cp)$ does not change in a direction defined by a positive unit step in t followed by a negative step of magnitude c in x (equation (2.41)), while the quantity $(r - cp)$ does not change in a direction defined by a positive unit step in t and a positive step of c in x (equation (2.42)). $(r + cp)$ and $(r - cp)$ are called *Riemann invariants*, and are thus constant along lines defined by $x + ct =$ constant and and $x - ct =$ constant, respectively (figure 2.19). The lines along which the Riemann invariants are constant are the characteristics.

The general behaviour of solutions to the wave equation can be easily anticipated by examining the characteristic diagram. Consider the initial solution shown in figure 2.20. A local solution which has $r - cp \neq 0$ and $r + cp = 0$ will propagate to the right with a speed c. Similarly, one with $r + cp \neq 0$ and $r - cp = 0$ will propagate to the left at a speed c.

2.6.2 Boundary conditions

The fact that there are distinct characteristics means that boundary conditions for the wave equation cannot be arbitrarily specified. This can be understood by

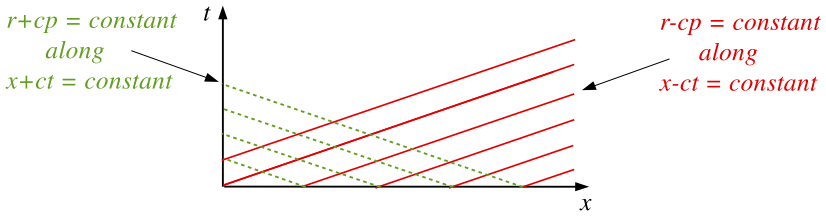


Figure 2.19: Characteristic lines for the wave equation

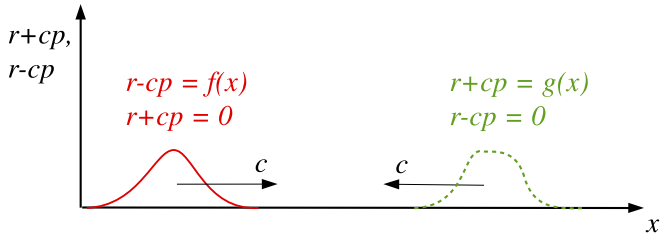


Figure 2.20: Propagation of initial solutions described by the wave equation

considering the characteristic lines shown in figure 2.21. For unsteady problems we normally know the complete initial solution $u(x, 0)$, which can be expressed in terms of two local values at each point, one for $r+cp$ and one for $r-cp$. Since the right-running characteristic must have a constant value for $r - cp$, however, we are no longer free to specify its value on the right boundary. Therefore, on the right boundary, we can only specify one physical boundary condition, for $r + cp$ [1].

Similarly, since the left boundary shares left-running characteristics with the initial condition, only $r-cp$ values can be specified. On the final time boundary, the solution will be completely determined by characteristics coming from the left and right boundaries, so no physical boundary condition may be specified.

If in a numerical approximation one attempts to specify the value of, for example, $r - cp$ both on the right boundary and as an initial condition, an ill-posed problem will be obtained, in the sense that there is no solution to the PDE which could satisfy the problem [2].

[1] In a computational model where values must be produced for both variables, the second relation required will come from a discretisation of the PDE in the parts of the interior closest to the boundary (a numerical condition).

[2] If a numerical approximation produces a solution under these circumstances, it will be quite useless.

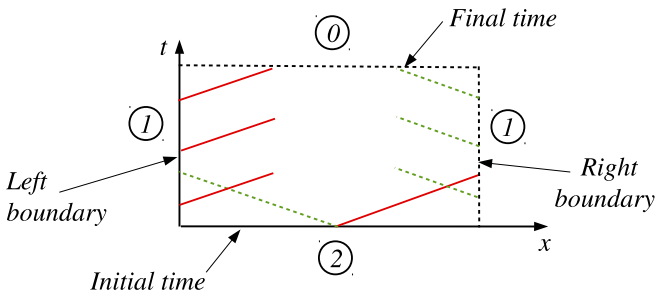


Figure 2.21: Boundary conditions for the wave equation. The number of physical conditions which can be specified on each boundary is indicated by the numbers within the circles.

2.7 A parabolic example

In this case we consider the linear diffusion (or heat) equation:

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} \tag{2.43}$$

which, among other things, can be used to describe the evolution of temperature, u, due to diffusion in a one-dimensional domain.

2.7.1 Characteristic analysis

As before we re-write the equation as a first-order system:

$$\frac{\partial u}{\partial t} - \nu \frac{\partial p}{\partial x} = 0, \tag{2.44}$$

$$p - \frac{\partial u}{\partial x} = 0. \tag{2.45}$$

Consider a linear combination of the above:

$$\alpha \left(\frac{\partial u}{\partial t} - \nu \frac{\partial p}{\partial x} \right) + \left(p - \frac{\partial u}{\partial x} \right) = 0 \tag{2.46}$$

which can also be written:

$$\left(\alpha \frac{\partial u}{\partial t} - \frac{\partial u}{\partial x} \right) + \left(-\alpha\nu \frac{\partial p}{\partial x} \right) = -p \tag{2.47}$$

Requiring the differentiation to act in one direction implies $\alpha = 0$, which results in a re-expression of (2.45). There is one characteristic direction ($ds = dx$), but that is insufficient to re-express the PDE as a system of ODEs, since we need two ODEs to solve for the two unknowns, p and u. The linear diffusion equation is therefore parabolic.

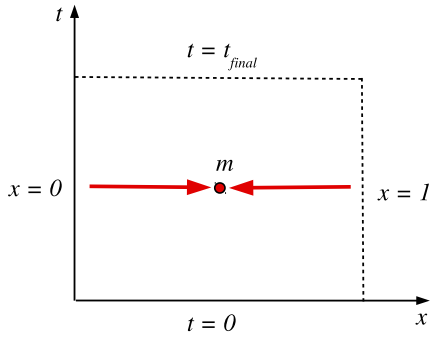


Figure 2.22: Space-time domain for the heat equation showing the domain of dependence of point m implied by the characteristic found by the analysis

2.7.2 Boundary conditions

The single characteristic direction $ds = dx$ implies that the solution at a point m (figure 2.22) is dependent on other points at the same t. Since the right-hand side of the ODE is a function of the solution, however, there is insufficient information for us to determine completely what point m is dependent on.

Parabolic equations normally have limited zones of dependence, however, and exploiting these can considerably simplify their (numerical) solution. In this case, we can gain additional insight by examining the exact solution of the linear diffusion equation for $\nu = 1$ on space-time domain shown in figure 2.23 with:

$$u(x,0) = \sin(\frac{\pi x}{2}); \quad u(0,t) = u(1,t) = 0 \tag{2.48}$$

With this initial condition, the linear diffusion equation has the exact solution (verify by substitution):

$$u(x,t) = \sin(\frac{\pi x}{2})e^{-\frac{\pi^2}{4}t} \tag{2.49}$$

This implies that the solution at any time t is dependent on the solution at a previous time as well as all the other points at time t (figure 2.23). In other words, to solve the problem in the space-time domain in the figure, we require an initial condition along the $t = 0$ boundary, and boundary conditions along the $x = 0$ and $x = 1$ boundaries from $t = 0$ to $t = t$ (see figure 2.1). This result is once again natural given our normal interpretation of time, where we expect that the present depends on the past. One could also argue, however, that given a boundary condition at t_{final} we could solve for the solution at time t by considering the space-time domain above t (i.e. by solving the problem backwards in time). For this equation, however, this can lead to an ill-posed problem. Examining the exact solution, it is clear that this equation describes a diffusion of the initial condition in time, until its values are hardly present

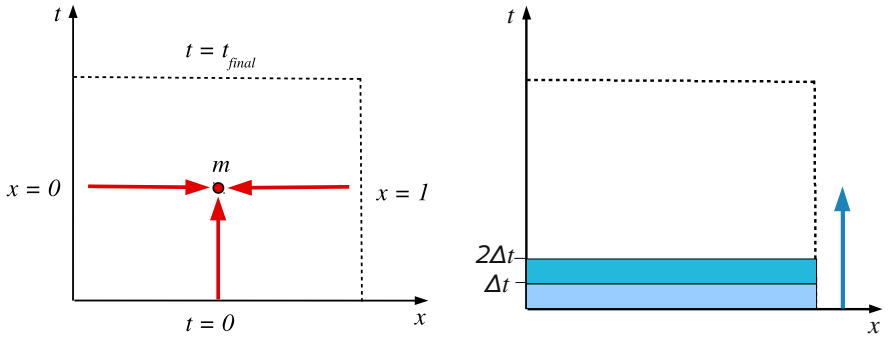


Figure 2.23: Incoming data to point m implied by the exact solution of the heat equation (left), and solution of the problem via time marching (right)

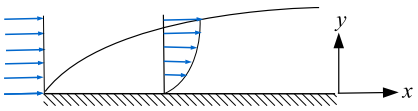


Figure 2.24: The boundary layer near a flat plate

in the solution. For the case where the boundary conditions at $x = 0$ and $x = 1$ are zero, the solution at t_{final} approaches zero if the time interval is large. A solution at time t computed using a condition at t_{final} will thus be extremely sensitive to small changes in the values at t_{final}. Given finite precision arithmetic, very poor results can be obtained.

Noting that the domain of dependence of the solution at t is restricted to the values at that time and those at a previous time is very advantageous for numerical solution procedures, since then we need not solve for the whole space-time solution at once. Instead, we can find the solution at a small finite time away from the initial condition (Δt), and then use this solution to advance another small step in time (figure 2.23 right). Repeating in this way, step after step until the desired time is reached, is referred to as *time marching*.

The opportunity to exploit the limited domain of dependence which exists in parabolic equations also occurs in less obvious situations. A notable example is when solving the boundary-layer equations of fluid mechanics (figure 2.24). These are parabolic in space, such that given the solution for all y at a given x position, one can march in space towards values of increasing x. This results in substantially less work than required to simultaneously determine the solution for all unknowns in the $x - y$ domain.

2.8 An elliptic example

As an example of an elliptic PDE we will consider Laplace's equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{2.50}$$

2.8.1 Characteristic analysis

(2.25) implies Laplace's equation is elliptic. To verify this, we introduce $p = \frac{\partial u}{\partial x}$, $q = \frac{\partial u}{\partial y}$ to express (2.50) as:

$$\frac{\partial p}{\partial x} + \frac{\partial q}{\partial y} = 0 \tag{2.51}$$

$$\frac{\partial q}{\partial x} - \frac{\partial p}{\partial y} = 0 \tag{2.52}$$

We now consider a linear combination of these equations:

$$\left(\frac{\partial p}{\partial x} + \frac{\partial q}{\partial y}\right) + \alpha\left(\frac{\partial q}{\partial x} - \frac{\partial p}{\partial y}\right) = 0 \tag{2.53}$$

which can be rewritten as:

$$\left(\frac{\partial}{\partial x} - \alpha\frac{\partial}{\partial y}\right) p + \left(\frac{\partial}{\partial x} + \frac{1}{\alpha}\frac{\partial}{\partial y}\right)(\alpha q) = 0 \tag{2.54}$$

Requiring that p and αq be differentiated in the same direction leads to:

$$\frac{\partial}{\partial x} - \alpha\frac{\partial}{\partial y} = \frac{\partial}{\partial x} + \frac{1}{\alpha}\frac{\partial}{\partial y} \quad \Rightarrow \quad \alpha = -\frac{1}{\alpha} \quad \Rightarrow \quad \alpha = \pm\sqrt{-1} \tag{2.55}$$

α is complex, so no real characteristics can be found, meaning Laplace's equation is indeed elliptic.

2.8.2 Boundary conditions

As there are no characteristics, the solution at all points in the domain depends continuously on that of all other points. A well-posed problem for Laplace's equation thus requires information to be specified at every point on the boundary. This may be in the form of either Dirichlet, Neumann or Robin conditions.

Unlike for the Poisson equation, there are no additional input terms, so the boundary values will completely determine the solution. Problems based on the Laplace equation are sometimes referred to as "jury problems", where by analogy each point on the boundary can be considered to be the member of a jury which determines the final outcome.

Chapter 3

Discretisation with the finite-difference method

In this chapter we will consider the solution of PDEs using the finite difference method. It is assumed that the reader is already familiar with the definition of the Taylor series and the concepts of truncation error and order of accuracy from previous courses. If not, a quick review of these concepts might be useful, using references such as [12, 10].

3.1 From PDE to algebraic system

The application of finite-difference methods to the solution of PDEs is conceptually straightforward. The first step is to consider the discrete representation of the continuous solution to be the collection of solution values on pre-specified points in the domain. The lines connecting such points are collectively referred to as a mesh (figure 3.1). PDEs have terms containing the derivatives of the

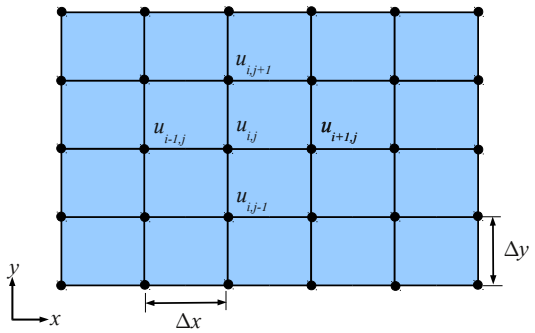


Figure 3.1: A uniform structured mesh for a rectangular domain

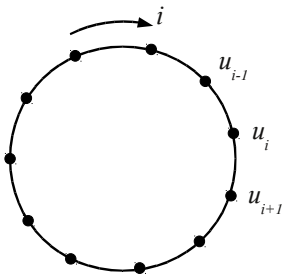


Figure 3.2: A one-dimensional periodic domain

solution. In a finite-difference method, one simply replaces these terms using finite-difference approximations, which results in an algebraic system of equations for the solution values at the mesh points. This will be demonstrated using examples given in the following sections. To simplify the discussion, we will sometimes consider domains which are periodic in one or more directions. These are domains which re-connect to themselves so that boundaries are avoided and therefore boundary conditions are not required. A 1D periodic domain is shown in figure 3.2.

3.1.1 Example: A transient 1D problem

Consider the following problem based on the linear advection equation on a periodic domain of length L:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{3.1}$$

$$u(x,0) = u_0(x) \tag{3.2}$$

$$u(0,t) = u(L,t) \tag{3.3}$$

which describes the undamped propagation of an initial solution $u_0(x)$ in the x direction with speed c (figure 3.3). Since there is only one set of characteristics, and all of the initial data is given, we may find the solution at some time Δt away from the initial condition by considering the portion of the space-time domain in between $t = 0$ and $t = \Delta t$ alone. Then, as with the heat-equation example of the previous chapter, we can proceed to subsequent time levels by repeating the procedure and marching in time.

To determine the unknown values of the solution at the next time level, we will use an approximation of (3.1) evaluated at each point in the mesh. Mesh point solutions can be uniquely identified by their indices, (i, n), where i indicates the node position in space, and n indicates the position in time (figure 3.4). To ensure a consistent method, all of the derivatives in (3.1) will be approximated using Taylor series expansions centred on (i, n), which is referred to as the expansion point (indicated by the large circle with a bold outline).

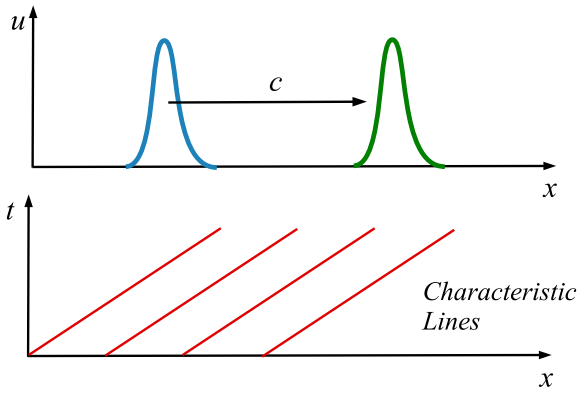


Figure 3.3: Characteristic lines of the linear advection equation (bottom) and propagation of an initial solution with speed c (top)

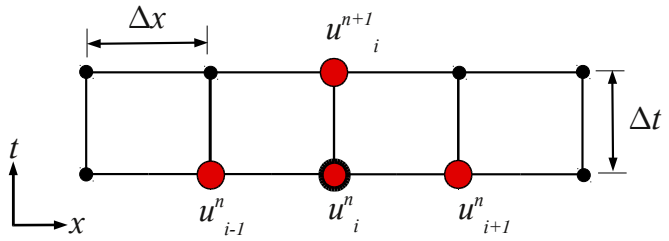


Figure 3.4: Finite-difference stencil for an explicit method. The circle with a bold outline indicates the expansion point for the Taylor series.

The approximations will use a few of the neighbouring points (indicated by the remaining large circles). These points, along with (i, n), are collectively referred to as the finite-difference *stencil*.

We begin by approximating the $\frac{\partial u}{\partial t}$ term in (3.1), using the following the Taylor series expansion in time:

$$u_i^{n+1} = u_i^n + \Delta t \left.\frac{\partial u}{\partial t}\right|_{(i,n)} + \frac{\Delta t^2}{2!} \left.\frac{\partial^2 u}{\partial t^2}\right|_{(i,n)} + O(\Delta t^3) \tag{3.4}$$

where the Landau symbol O indicates the order of the remaining terms in the expansion. This can be re-arranged to get the following first-order accurate expression for the time derivative:

$$\frac{\partial u}{\partial t} = \frac{(u_i^{n+1} - u_i^n)}{\Delta t} + O(\Delta t) \tag{3.5}$$

Similarly, the Taylor series expansion for u^n_{i+i} and u^n_{i-i} are:

$$u^n_{i+1} = u^n_i + \Delta x \left.\frac{\partial u}{\partial x}\right|_{(i,n)} + \frac{\Delta x^2}{2!} \left.\frac{\partial^2 u}{\partial x^2}\right|_{(i,n)} + O(\Delta x^3) \tag{3.6}$$

$$u^n_{i-1} = u^n_i - \Delta x \left.\frac{\partial u}{\partial x}\right|_{(i,n)} + \frac{\Delta x^2}{2!} \left.\frac{\partial^2 u}{\partial x^2}\right|_{(i,n)} + O(\Delta x^3) \tag{3.7}$$

The second expansion can be subtracted from the first to obtain:

$$\frac{\partial u}{\partial x} = \frac{(u^n_{i+1} - u^n_{i-1})}{2\Delta x} + O(\Delta x^2) \tag{3.8}$$

Therefore, a discretised version of (3.1) is:

$$\frac{(u^{n+1}_i - u^n_i)}{\Delta t} + c\frac{(u^n_{i+1} - u^n_{i-1})}{2\Delta x} = O(\Delta t) + O(\Delta x^2) \tag{3.9}$$

which, neglecting higher order terms, can be rearranged to solve for u^{n+1}_i:

$$u^{n+1}_i = u^n_i - \frac{c\Delta t}{\Delta x}\frac{(u^n_{i+1} - u^n_{i-1})}{2} \tag{3.10}$$

We then apply (3.10) at each node in the mesh where the solution is to be determined by the PDE. On a mesh with M such points, this will result in an algebraic system of M equations for the M unknown values of u^{n+1}_i of the form:

$$\vec{u}^{\,n+1} = ([I] - [D])\,\vec{u}^{\,n} \tag{3.11}$$

where $\vec{u}^{\,n}$ is the vector of solution values $[u^n_1, u^n_2, u^n_3 ...]^T$ for all i, and $[D]$ is the sparse matrix given by:

$$[D] = \frac{c\Delta t}{2\Delta x}\begin{bmatrix} 0 & 1 & & & & & -1 \\ -1 & 0 & 1 & & & & \\ & -1 & 0 & 1 & & & \\ & & -1 & 0 & 1 & & \\ & & & -1 & 0 & 1 & \\ & & & & -1 & 0 & 1 \\ 1 & & & & & -1 & 0 \end{bmatrix} \tag{3.12}$$

The non-zero values in the left lower and right upper corners of $[D]$ result from considering a periodic domain. Note that such an algebraic system is the expected result of the discretisation step described in figure 1.3.

The system (3.11) for u^{n+1}_i is not coupled, and can be solved directly. In fact, since expression (3.10) gives values for u^{n+1}_i purely in terms of known data from the previous time level, it can be used to immediately obtain a value of the solution at i at the next time level without considering the u^{n+1} values of neighbouring points. Methods of this form, which do not require the solution of

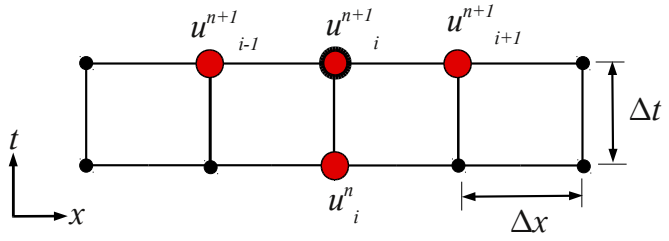


Figure 3.5: Finite-difference stencil for an implicit method

a coupled system of equations, are referred to as *explicit*. The particular first-order accurate time-integration method obtained by using (3.5) and evaluating all spatial derivatives at time level n is referred to as the *explicit Euler method*.

In contrast, we could also have considered a stencil centred on the point $(i, n+1)$ (figure 3.5). A similar procedure leads to:

$$u_i^{n+1} + \frac{c\Delta t}{\Delta x}\frac{(u_{i+1}^{n+1} - u_{i-1}^{n+1})}{2} = u_i^n \tag{3.13}$$

In this case the values of the solution used to compute the spatial derivatives are unknowns. This leads to an *implicit* method for advancing the solution in time, which requires the solution of the following coupled algebraic system:

$$([I] + [D])\,\vec{u}^{\,n+1} = \vec{u}^{\,n} \tag{3.14}$$

where $[D]$ is the same sparse matrix used for the explicit scheme. This implicit method, based on a simple first-order accurate approximation of the time derivative, with all spatial derivatives evaluated at the $n+1$ time level, is called the *implicit Euler method*.

3.1.2 Example: A steady 2D problem

The procedure for multi-dimensional operators is similar to that shown in the previous section, with the exception of some details related to the structure of the solution vector, and the use of multi-dimensional Taylor expansions. For example, consider the discretisation of the equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial x \partial y} = 0 \tag{3.15}$$

on the uniform structured mesh shown in figure 3.1. Assume that the spacing between the points, Δx and Δy is constant, and the mesh lines are aligned with the x and y coordinate directions. Referring to the stencil in figure 3.6, second-order accurate finite-difference approximations for the first and second

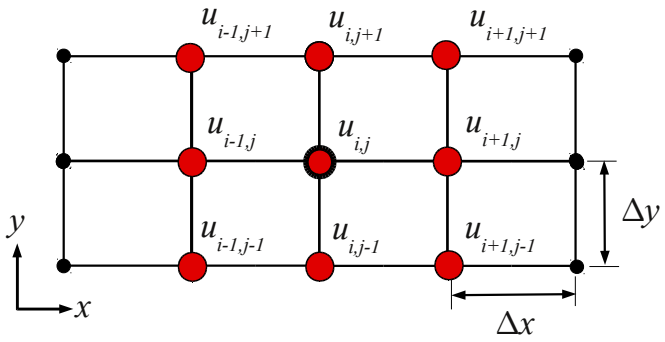


Figure 3.6: Finite-difference stencil for an 2D domain

terms are:

$$\frac{\partial^2 u}{\partial x^2} = \frac{(u_{i+1,j} - 2u_{i,j} + u_{i-1,j})}{\Delta x^2} + O(\Delta x^2) \tag{3.16}$$

$$\frac{\partial^2 u}{\partial y^2} = \frac{(u_{i,j+1} - 2u_{i,j} + u_{i,j-1})}{\Delta y^2} + O(\Delta y^2) \tag{3.17}$$

For the third term, one needs to consider the multi-dimensional version of the Taylor series approximation:

$$u(x_o + \Delta x, y_o + \Delta y) = u(x_o, y_o) + \left[\frac{\partial u}{\partial x}\Delta x + \frac{\partial u}{\partial y}\Delta y\right] + \frac{1}{2!}\left[\frac{\partial^2 u}{\partial x^2}\Delta x^2 + 2\frac{\partial^2 u}{\partial x \partial y}\Delta x \Delta y + \frac{\partial^2 u}{\partial y^2}\Delta y^2\right] + ... \tag{3.18}$$

From which the following expression can be derived:

$$\frac{\partial^2 u}{\partial x \partial y} = \frac{(u_{i+1,j+1} - u_{i+1,j-1} + u_{i-1,j-1} - u_{i-1,j+1})}{4\Delta x \Delta y} + O(\Delta x^2, \Delta y^2) \tag{3.19}$$

To visualise the resulting algebraic system, consider the solution at the mesh points collected into a solution vector, $\vec{u}$, ordered first by moving through the mesh in the i and then in the j direction:

$$\vec{u} = \{u_{1,1} \;\; u_{2,1} \;\; u_{3,1} \;\cdots\; u_{1,2} \;\; u_{2,2} \;\; u_{3,2} \;\cdots\; u_{I,J}\}^T \tag{3.20}$$

where I is the number of points in the i direction and J is the number of points in the j direction (figure 3.7). We can write the resulting algebraic system as:

$$[A]\vec{u} = ([A_1] + [A_2] + [A_3])\,\vec{u} = 0 \tag{3.21}$$

where the three matrices $[A_1]$, $[A_2]$ and $[A_3]$ correspond to finite-difference approximations of the terms $\partial^2 u/\partial x^2$, $\partial^2 u/\partial y^2$, and $\partial^2 u/\partial x \partial y$. Each row in this

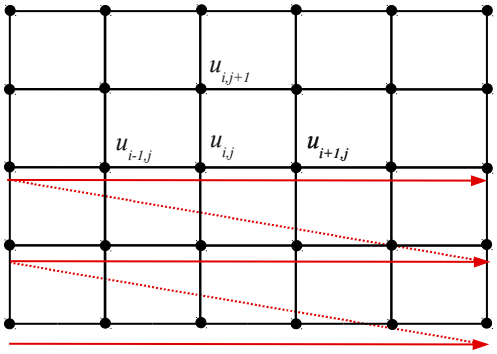


Figure 3.7: Solution vector ordering for a 2D domain

system thus corresponds to the finite-difference approximation at one of the mesh points, with the row order determined by figure 3.7. Away from boundaries, the matrix due to the $\frac{\partial^2 u}{\partial x^2}$ term will have a structure similar to:

$$[A_1] = \frac{1}{\Delta x^2}\begin{bmatrix} \ddots & \ddots & \ddots & & & & \\ & 1 & -2 & 1 & & & \\ & & 1 & -2 & 1 & & \\ & & & 1 & -2 & 1 & \\ & & & & \ddots & \ddots & \ddots \end{bmatrix} \tag{3.22}$$

while that from the $\frac{\partial^2 u}{\partial y^2}$ term will contain similar coefficients, but due to the ordering of the data in the vector, they will no longer be in adjacent columns:

$$[A_2] = \frac{1}{\Delta y^2}\begin{bmatrix} \ddots & & \ddots & & \ddots & & & \\ & 1 & \cdots & -2 & \cdots & 1 & & \\ & & 1 & \cdots & -2 & \cdots & 1 & \\ & & & 1 & \cdots & -2 & \cdots & 1 \\ & & & & \ddots & & \ddots & & \ddots \end{bmatrix} \tag{3.23}$$

In this case, the maximum number of columns between coefficients on a given row (known as the *bandwidth*), will be twice the number of mesh nodes in the i direction $+1$. A slightly higher bandwidth can be expected for the $[A_3]$ matrix. The final $[A]$ matrix will thus be structured but sparse, with a bandwidth which increases with refinement of the mesh. There exist special methods for solving such coupled systems of equations efficiently, which will be described in the last chapter.

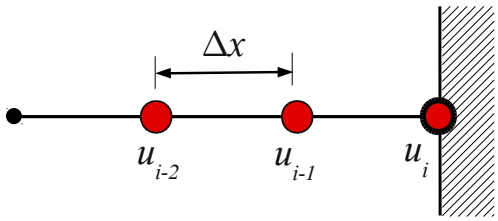


Figure 3.8: A finite-difference stencil for a boundary solution value

3.1.3 Deriving arbitrary difference expressions: The Taylor table

The previous examples used finite-difference expressions based on mesh points immediately adjacent to the point on which the expansions are centred. In order to obtain higher-order accuracy, however, the finite-difference stencil must also involve mesh points which are further away. In addition, as the boundaries of a domain are approached, it becomes necessary to use one-sided stencils. A procedure is therefore required to derive the most accurate finite-difference approximation possible for a desired derivative given an arbitrary stencil.

Consider the one-sided stencil shown in figure 3.8. Assume we need an approximation for the first derivative at point i, obtained using a linear combination of the solution values:

$$\frac{\partial u}{\partial x} = \frac{(au_{i-2} + bu_{i-1} + cu_i)}{\Delta x} - TE \tag{3.24}$$

where TE is the truncation error. Since there are three unknown coefficients a, b, c, we need a system of three equations. This can be generated using a Taylor table [20]. We begin by writing (3.24) as:

$$au_{i-2} + bu_{i-1} + cu_i = \Delta x \left(\left.\frac{\partial u}{\partial x}\right|_i + TE \right) \tag{3.25}$$

Note that the values of the solution u_{i-2} and u_{i-1} can be expressed as Taylor series expansions about point i. This can be summarised in tabular form, as shown in table 3.1. The table contains three Taylor expansions for the solution about point i, each multiplied by a coefficient a, b or c. These appear vertically in the three interior columns labelled $a \cdot u_{i-2}$, $b \cdot u_{i-1}$ and $c \cdot u_i$, respectively. The columns contain the numerical coefficients of the series which multiply the terms in the leftmost column of the table. The (-1) and (-2) values indicate the multiples of Δx the solution point is displaced from the expansion point i. Together, the three interior columns represent the terms on the left-hand side of (3.25). The rightmost column contains the terms on the right-hand side of (3.25).

The first three rows are in fact a system of equations we can use to determine a, b and c. The first row states that a, b and c should have values such that there

	$a \cdot u_{i-2}$	$b \cdot u_{i-1}$	$c \cdot u_i$	
u_i	a	b	c	0
$\Delta x \frac{\partial u}{\partial x}\big\vert_i$	$a(-2)$	$b(-1)$	0	1
$\Delta x^2 \frac{\partial^2 u}{\partial x^2}\big\vert_i$	$a(-2)^2/2!$	$b(-1)^2/2!$	0	0
$\Delta x^3 \frac{\partial^3 u}{\partial x^3}\big\vert_i$	$a(-2)^3/3!$	$b(-1)^3/3!$	0	

Table 3.1: Taylor table for computing a one-sided 3-point expression for the first derivative. The interior columns (between the double vertical bars) contain the numerical coefficients for three Taylor expansions about the point i.

are no u_i terms on the right-hand side. The second row states that the first derivatives in the Taylor series should sum to produce a first derivative with unit magnitude on the right-hand side. The third row states that there should be no $\frac{\partial^2 u}{\partial x^2}$ terms on the right hand side. This will ensure that they do not become part of the truncation error. The fourth row is not required for determining a, b and c, but is retained to determine the leading TE term. The system of equations for a, b and c given by the first three rows of the table is thus:

$$\begin{bmatrix} 1 & 1 & 1 \\ -2 & -1 & 0 \\ 2 & \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{3.26}$$

Solving this system gives $a = \frac{1}{2}$, $b = -2$ and $c = \frac{3}{2}$. We can substitute these values into the fourth row to get the first additional contribution to the right-hand side, $-\frac{\Delta x^3}{3}\frac{\partial^3 u}{\partial x^3}\big\vert_i$, which is Δx times the leading term of the truncation error, TE. Therefore we can express (3.24) as:

$$\frac{\partial u}{\partial x}\bigg\vert_i = \frac{(u_{i-2} - 4u_{i-1} + 3u_i)}{2\Delta x} + O(\Delta x^2) \tag{3.27}$$

Naturally we could have introduced more solution values into (3.24) (e.g. u_{i-3}, u_{i-4} ...) and generated equations for their unknown coefficients by adding more rows to the table. We could then choose to set the right-hand side contribution of increasingly high order terms to zero, eliminating more terms from the TE, and generating increasingly accurate expressions for $\frac{\partial u}{\partial x}\big\vert_i$.

The Taylor table: Summary

The finite-difference approximation for a derivative using a stencil with N points can be derived with a Taylor table via the following procedure:

1. In the first column, write $N+1$ terms of the Taylor series expansion about the point of interest vertically without the numerical coefficients

2. In the following column, write the numerical coefficients which multiply the terms in the Taylor series to produce the solution at a point in the stencil multiplied by an unknown magnitude (e.g. a,b,c). Repeat this in additional columns for the other points in the stencil.

3. In the first N rows of the last column of the table, write a 1 in the row corresponding to the Taylor term with the desired derivative, and a zero in the other rows.

4. Solve the system of equations given by the first N rows of the table for the unknown magnitudes (a,b,c).

5. Substitute the values found for the magnitudes into the last $(N+1)$ rows of the table, and divide the remainder by the factor multiplying the desired derivative to obtain the TE.

3.1.4 Applying conditions on boundaries

At boundaries we may need to impose solution values (Dirichlet conditions) or solution derivatives (Neumann conditions). Alternatively, we may need to allow the boundary values to be determined by the solution in the interior (numerical conditions). We discuss how to apply such conditions when using finite-difference methods below.

Dirichlet boundary conditions

Specifying a boundary solution value directly is straightforward. First, the finite-difference equation corresponding to the PDE expanded about the boundary point is removed from the algebraic system. Then the specified boundary value is substituted into the remaining equations wherever it appears. For explicit methods, the algebraic equations are uncoupled and one may simply load the Dirichlet boundary value directly. For implicit time-marching methods, or steady problems resulting a coupled system of equations, the matrix and right-hand side of the algebraic system can be modified to incorporate the Dirichlet B.C.

Consider the implicit discretisation for the linear advection equation on a periodic domain described in section 3.1.1. The algebraic system (3.14) can be expressed in terms of the combined matrix $[C] = [I] + [D]$ for a 7-node mesh as $[C]\vec{u}^{\,n+1} = \vec{u}^{\,n}$, or:

$$\begin{bmatrix} 1 & \mu & & & & & -\mu \\ -\mu & 1 & \mu & & & & \\ & -\mu & 1 & \mu & & & \\ & & -\mu & 1 & \mu & & \\ & & & -\mu & 1 & \mu & \\ & & & & -\mu & 1 & \mu \\ \mu & & & & & -\mu & 1 \end{bmatrix} \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ u_4^{n+1} \\ u_5^{n+1} \\ u_6^{n+1} \\ u_7^{n+1} \end{bmatrix} = \begin{bmatrix} u_1^{n} \\ u_2^{n} \\ u_3^{n} \\ u_4^{n} \\ u_5^{n} \\ u_6^{n} \\ u_7^{n} \end{bmatrix} \tag{3.28}$$

where $\mu = \frac{c\Delta t}{2\Delta x}$. The first of these equations expresses an approximation of the PDE at node $i = 1$. If instead a Dirichlet condition $u = g$ was desired, this row would be replaced in both the matrix and right hand side as shown below:

$$\begin{bmatrix} 1 & -- & -- & -- & -- & -- & -- \\ -\mu & 1 & \mu & & & & \\ & -\mu & 1 & \mu & & & \\ & & -\mu & 1 & \mu & & \\ & & & -\mu & 1 & \mu & \\ & & & & -\mu & 1 & \mu \\ & & & & & ? & ? \end{bmatrix} \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ u_4^{n+1} \\ u_5^{n+1} \\ u_6^{n+1} \\ u_7^{n+1} \end{bmatrix} = \begin{bmatrix} g^{n+1} \\ u_2^n \\ u_3^n \\ u_4^n \\ u_5^n \\ u_6^n \\ u_7^n \end{bmatrix} \tag{3.29}$$

Note that by applying a Dirichlet boundary condition at $i = 1$ we have broken the periodic domain assumed in the example, and now need to do something with the $i = I = 7$ (right) boundary, as indicated by the question marks. Since the linear advection equation has only one set of characteristic lines, which bring data forward in time and to the right, (for positive c), we need to apply a numerical condition at $i = I$. An approach to doing so will be given after the application of Neumann conditions is explained.

Neumann boundary conditions

To specify a Neumann (derivative) boundary condition in a coupled system, the finite-difference equation corresponding to the original PDE at the boundary point is replaced with a one-sided finite-difference expression which equates the derivative at the boundary point to the required value. In section 3.1.3, an expression for a one-sided approximation to the first derivative on a boundary point was derived. This can be expressed for the case of the boundary on the left and the numerical domain on the right as:

$$\left.\frac{\partial u}{\partial x}\right|_i = \frac{(-3u_i + 4u_{i+1} - u_{i+2})}{2\Delta x} + O(\Delta x^2) \tag{3.30}$$

For our linear advection example, the Neumann condition $\frac{\partial u}{\partial x} = q$ at $i = 1$ could then be implement by replacing the first equation:

$$\begin{bmatrix} \frac{-3}{2\Delta x} & \frac{2}{\Delta x} & \frac{-1}{2\Delta x} & & & & \\ -\mu & 1 & \mu & & & & \\ & -\mu & 1 & \mu & & & \\ & & -\mu & 1 & \mu & & \\ & & & -\mu & 1 & \mu & \\ & & & & -\mu & 1 & \mu \\ & & & & & ? & ? \end{bmatrix} \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ u_4^{n+1} \\ u_5^{n+1} \\ u_6^{n+1} \\ u_7^{n+1} \end{bmatrix} = \begin{bmatrix} q^{n+1} \\ u_2^n \\ u_3^n \\ u_4^n \\ u_5^n \\ u_6^n \\ u_7^n \end{bmatrix} \tag{3.31}$$

where the first element of the right-hand side is replaced by the desired value for $\frac{\partial u}{\partial x}$. Here we have used the one-sided difference expression derived in section 3.1.3, with a sign change reflecting the different orientation of the boundary.

Numerical conditions

In our discretisation for the linear advection equation, we have so far assumed that c is positive. A characteristic analysis will show that information therefore travels from left to right, which allows the specification of physical boundary conditions (e.g. Dirichlet or Neumann) on the left boundary. Correspondingly, we cannot specify a physical boundary condition on the right boundary, where its solution must be determined using the PDE. This can be done by using a finite-difference approximation of the PDE expanded about the boundary point, resulting in a numerical boundary condition. Naturally, the finite-difference equation must be constructed using one-sided differences to allow the terms of the PDE to be determined using values from the interior. Continuing from the previous example with a Neumann condition at $i = 1$, adding a numerical condition using a one-sided operator at the $i = I$ boundary would result in:

$$\begin{bmatrix} \frac{-3}{2\Delta x} & \frac{2}{\Delta x} & \frac{-1}{2\Delta x} & & & & & \\ -\mu & 1 & \mu & & & & & \\ & -\mu & 1 & \mu & & & & \\ & & -\mu & 1 & \mu & & & \\ & & & -\mu & 1 & \mu & & \\ & & & & -\mu & 1 & \mu & \\ & & & & \frac{c\Delta t}{2\Delta x} & \frac{-2c\Delta t}{\Delta x} & \left(1+\frac{3c\Delta t}{2\Delta x}\right) \end{bmatrix} \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ u_4^{n+1} \\ u_5^{n+1} \\ u_6^{n+1} \\ u_7^{n+1} \end{bmatrix} = \begin{bmatrix} q \\ u_2^n \\ u_3^n \\ u_4^n \\ u_5^n \\ u_6^n \\ u_7^n \end{bmatrix}$$

Note the difference between the first and last equations of the system. The first expresses $\frac{\partial u}{\partial x} = q$ on the left boundary. The last expresses $\Delta t \left(\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x}\right) = 0$ on the right boundary, where $\frac{\partial u}{\partial x}$ is evaluated using the one-sided expression derived in section 3.1.3. Note also that the right boundary condition is discretised in time using the same first-order approximation to the time derivative used in equation (3.10).

For the linear advection equation, u is the characteristic variable, so the expression for the right boundary condition shown above is relatively simple. If the wave equation had been considered, additional equations defining the p and q variables of chapter 2 would need to be included. Then a numerical condition would be specified for the outgoing Riemann invariant, and a Dirichlet or Neumann condition for the incoming Riemann invariant.

3.1.5 Upwinding and artificial dissipation

By now it should be clear that there is a wide range of different finite-difference expressions which could be used to approximate a PDE. These can have different orders of accuracy, or be biased in a particular direction. It is natural to ask if one is better than the other. We will use the linear advection equation to demonstrate that the choice of finite-difference expressions can indeed have profound impact on the solution.

As shown in figure 3.3, the linear advection equation simply propagates initial conditions without changing their shape. When computed with a finite-difference scheme, however, the presence of truncation error means that defects

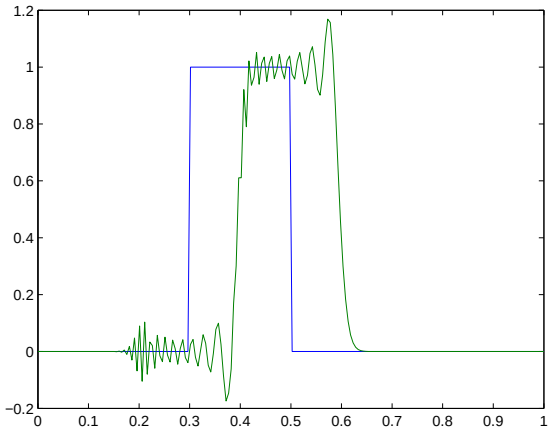


Figure 3.9: Numerical solution for the linear advection equation with a square wave initial condition produced by an implicit Euler in time, central in space discretisation (CFL number = 0.1). Errors such as those shown, due to some frequency components travelling at the incorrect speed, are referred to as "dispersive errors".

will occur during the propagation. The implicit in time, central in space discretisation described in section 3.1.1, for example, tends to produce *dispersive errors* as shown in figure 3.9. In practice such oscillations are undesirable, so an early fix was to introduce an *artificial dissipation* term to smooth them. This was justified with the idea that it is unreasonable to expect a numerical scheme to resolve high-gradient regions with thicknesses less than the mesh size. Therefore, we should instead solve a modified PDE which includes some additional dissipation which increases the thickness of high-gradient regions so that they are on the order of the mesh size. Adding an artificial dissipation term to (3.13), for example, results in:

$$\frac{(u_i^{n+1} - u_i^n)}{\Delta t} + c\frac{(u_{i+1}^{n+1} - u_{i-1}^{n+1})}{2\Delta x} - k\Delta x^2 \frac{(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1})}{\Delta x^2} = 0 \quad (3.32)$$

where k is a user-specified constant, and the Δx^2 term is added to ensure that the dissipation term vanishes as the mesh is refined. The artificial dissipation term in this case is inspired by the diffusive behaviour inherent to the advection-diffusion equation:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0$$

as illustrated in figure 3.10. A solution computed using (3.32) is shown in

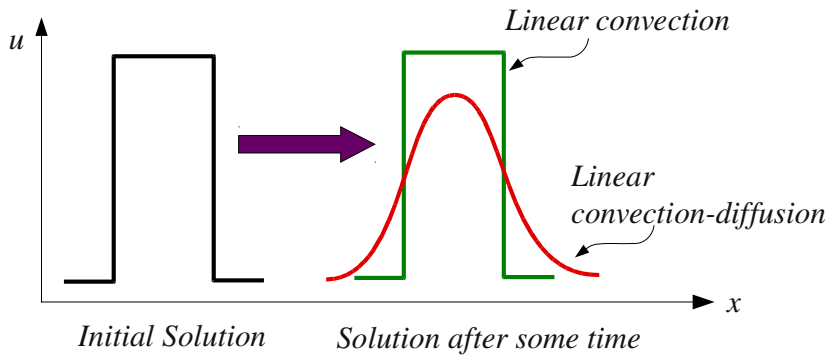


Figure 3.10: Solutions of the linear advection and advection-diffusion equations

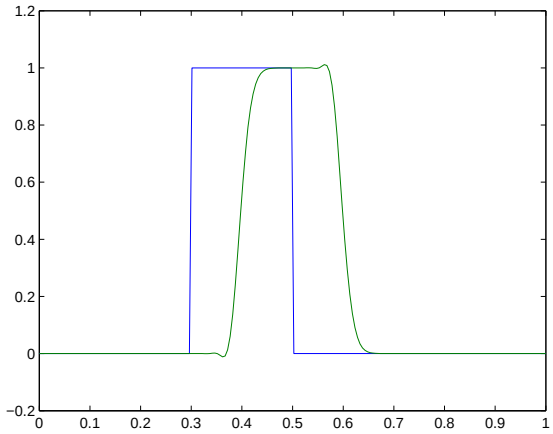


Figure 3.11: Numerical solution for the linear advection equation with a square wave initial condition produced by an implicit Euler in time, central in space discretisation with artificial dissipation (CFL number $= 0.1$, $k\Delta t = 0.02$)

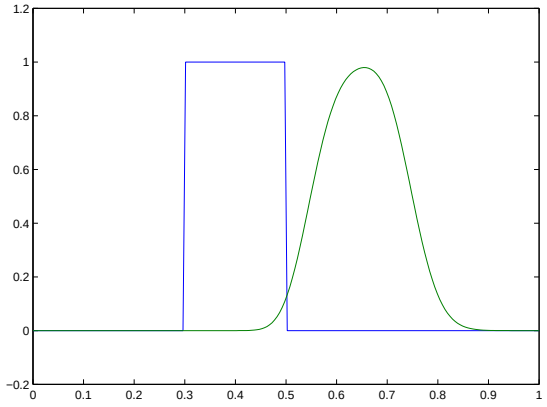


Figure 3.12: Numerical solution for the linear advection equation with a square wave initial condition produced by an implicit Euler in time, upwind in space discretisation (CFL number = 0.1)

figure 3.11.

As an alternative to using a central scheme with added artificial dissipation, we could also consider a scheme where the spatial derivative is evaluated in a biased manner:

$$u_i^{n+1} + \frac{c\Delta t}{\Delta x}(u_i^{n+1} - u_{i-1}^{n+1}) = u_i^n \tag{3.33}$$

In this case, the derivative is evaluated using an upstream point (one in the direction that information is coming from). This is convenient since the same expression can be used for numerical boundary conditions, for which its bias will ensure that the problem remains well posed. The expression is only first-order accurate however, and its truncation error tends to be highly diffusive (figure 3.12) The truncation error of this scheme produces a solution behaviour which is similar to that due to adding an additional dissipative term. In fact, one can re-write the spatial derivative term in (3.33) in a way which illustrates this more clearly:

$$\underbrace{c\frac{(u_i^{n+1} - u_{i-1}^{n+1})}{\Delta x}}_{\text{upwind}} = \underbrace{c\frac{(u_{i+1}^{n+1} - u_{i-1}^{n+1})}{2\Delta x}}_{\text{central}} \underbrace{-c\frac{\Delta x}{2}\frac{(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1})}{\Delta x^2}}_{\text{artificial dissipation}}$$

In this case using upwinding is similar to using a particular type of artificial dissipation. This analogy can not always be made, however, and in fact higher-order upwind schemes need additional terms to prevent oscillations. Upwinding is still favoured by many practitioners, however, at it provides a clear basis for the design of numerical methods for hyperbolic PDEs.

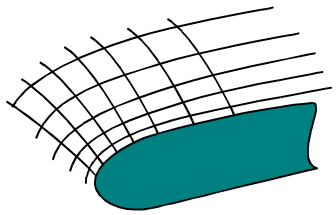


Figure 3.13: Mesh refinement near a curved boundary

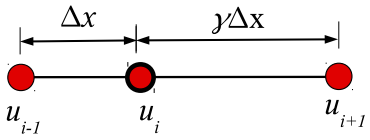


Figure 3.14: Stencil for $\frac{\partial u}{\partial x}$ with irregular mesh spacing

Both the upwind and central + artificial dissipation approaches remain popular in schemes for solving advective problems. In essence, both are attempting to modify the effects of the truncation error so they provide a good model for what is *not* computed. Modern multiscale theory addresses this topic directly.

3.2 Dealing with irregular meshes

Up to now we have considered the application of finite-difference methods to rectangular meshes. Many problems, however, are defined in non-rectangular domains. In addition, since truncation errors are typically proportional to solution gradients, it may also be desirable to cluster mesh points in regions of strong solution variation to obtain maximum accuracy (figure 3.13). One approach to discretisation on irregular stencils is to use multi-dimensional Taylor series expansions to derive unique finite-difference expressions for each node under consideration. Another option is to use a generalised transformation to map the problem to a Cartesian domain. Both of these are discussed below.

3.2.1 Operators for unequal mesh spacing

As an example, we consider the derivation of a three-point approximation for $\frac{\partial u}{\partial x}\Big|_i$ where the solution points u_{i-1}, u_i and u_{i+1} are spaced unequally (figure 3.14). We denote the spacing between the points specifying u_{i-1} and u_i as Δx, and that between u_i and u_{i+1} as $\gamma \Delta x$, where γ is the mesh expansion ratio. We are looking for a difference expression of the form:

$$\frac{(au_{i-1} + bu_i + cu_{i+1})}{\Delta x} = \frac{\partial u}{\partial x}\bigg|_i + TE \tag{3.34}$$

	$a \cdot u_{i-1}$	$b \cdot u_i$	$c \cdot u_{i+1}$	
u_i	a	b	c	0
$\Delta x \frac{\partial u}{\partial x}\big\vert_i$	$a(-1)$	0	$c(\gamma)$	1
$\Delta x^2 \frac{\partial^2 u}{\partial x^2}\big\vert_i$	$a(-1)^2/2!$	0	$c(\gamma)^2/2!$	0
$\Delta x^3 \frac{\partial^3 u}{\partial x^3}\big\vert_i$	$a(-1)^3/3!$	0	$c(\gamma)^3/3!$	

Table 3.2: Taylor table for computing a 3-point expression for the first derivative on a stretched mesh

As before, the expression can be derived using the Taylor table shown in table 3.2.1, which leads to the system:

$$\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & \gamma \\ \frac{1}{2} & 0 & \frac{\gamma^2}{2} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{3.35}$$

Solving this system gives:

$$[a, b, c] = \frac{1}{\gamma(\gamma+1)}[-\gamma^2, (\gamma^2-1), 1], \tag{3.36}$$

which results in the finite-difference expression:

$$\frac{\partial u}{\partial x}\bigg|_i = \frac{-\gamma^2 u_{i-1} + (\gamma^2-1)u_i + u_{i+1}}{\gamma(\gamma+1)\Delta x} \tag{3.37}$$

Equation (3.37) can be seen to be consistent with the standard second-order accurate expression for the first derivative when $\gamma = 1$.

3.2.2 The generalised transformation

An alternative to working with finite-difference expressions in physical coordinates is to transform the problem to a Cartesian domain where simple finite-difference expressions can be used. As an example, we will consider the discretisation of:

$$\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} = 0 \tag{3.38}$$

on a curvilinear mesh, as shown in figure 3.15 (left). We will employ a transformation of the form:

$$x = x(\xi, \eta) \tag{3.39}$$

$$y = y(\xi, \eta) \tag{3.40}$$

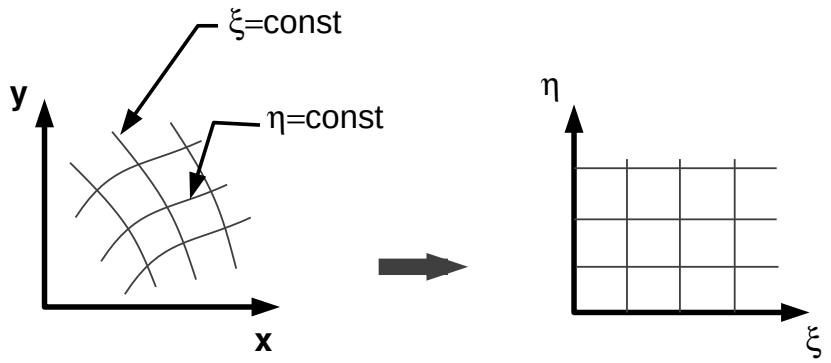


Figure 3.15: Physical and transformed coordinates

which transforms the curvilinear mesh lines to Cartesian grid lines in the (ξ, η) plane, as shown in the right of the figure. For the transformation to be well-defined, the curvilinear mesh must be structured (i.e. in 2D it must be made up of quadrilaterals and have two clearly identifiable coordinate directions).

The PDE is to be solved on the Cartesian domain, so it must be expressed in (ξ, η) coordinates. This can be done using the chain rule, with which (3.38) can be expressed as:

$$\frac{\partial u}{\partial \xi}\xi_x + \frac{\partial u}{\partial \eta}\eta_x + \frac{\partial u}{\partial \xi}\xi_y + \frac{\partial u}{\partial \eta}\eta_y = 0 \tag{3.41}$$

where quantities such as $\xi_x = \frac{\partial \xi}{\partial x}, \eta_y = \frac{\partial \eta}{\partial y}$... are called metrics of the transformation. Evaluating metrics such as ξ_x and η_y directly would require using finite differences in (x, y) coordinates, which would defeat the purpose of using the transformation. However, we can express the values of these metrics in terms of another set of metrics which can easily be evaluated on the Cartesian mesh. To do so, we note that if we take differential steps in the transformed coordinate system, this will imply differential steps in the physical coordinate system given by:

$$\begin{aligned} dx &= x_\xi \, d\xi + x_\eta \, d\eta \\ dy &= y_\xi \, d\xi + y_\eta \, d\eta \end{aligned} \tag{3.42}$$

This can be written in matrix form as:

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{bmatrix} \begin{bmatrix} d\xi \\ d\eta \end{bmatrix} \tag{3.43}$$

We could have also started by considering differential steps in the physical coordinate system:

$$\begin{bmatrix} d\xi \\ d\eta \end{bmatrix} = \begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} \tag{3.44}$$

By comparison, the second matrix of metrics is equivalent to the inverse of the first. This leads to:

$$\begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix} = J^{-1} \begin{bmatrix} y_\eta & -x_\eta \\ -y_\xi & x_\xi \end{bmatrix} \tag{3.45}$$

where $J = (x_\xi y_\eta - y_\xi x_\eta)$ is called the Jacobian of the transformation.

The metrics x_ξ, x_η, y_ξ and y_η can easily be evaluated on the Cartesian mesh, using expressions such as:

$$x_\xi = \frac{(x_{i+1,j} - x_{i-1,j})}{2\Delta\xi} \tag{3.46}$$

Normally it is chosen to define the transformation such that $\Delta\xi = \Delta\eta = 1$, further simplifying the expression. Once the x_ξ, x_η.. are computed, the ξ_x, η_y... metrics needed for solving the transformed version of the PDE can be quickly evaluated using (3.45).

Transforming PDEs with higher than first-order derivatives follows the same procedure, but the final expressions can be complex. For more details, the reader is referred to reference [29].

3.3 Confirming consistency and stability

3.3.1 Confirming consistency: The modified equation

Recall our definition of consistency from chapter 1:

A discretisation is consistent if when substituting an exact solution into the discrete equations the only terms which remain are those which tend to zero as the number of degrees of freedom is increased.

We can establish consistency for any finite-difference approximation of a PDE simply by substituting the Taylor series expansion of the exact solution for each of the solution terms which appear in the finite-difference equations (these expansions must be about the same point in space). This will produce the modified equation, which is the version of the PDE (including truncation error terms) which is actually solved. As an example, consider the explicit Euler in time, central in space finite-difference approximation for the linear advection equation given in section 3.1.1:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \quad \rightarrow \quad \frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} = 0 \tag{3.47}$$

We begin by writing out the Taylor-series expansions of the solution terms with

respect to u_i^n:

$$\begin{aligned} u_i^{n+1} &= u_i^n + \Delta t \left.\frac{\partial u}{\partial t}\right|_{n,i} + \frac{1}{2}\Delta t^2 \left.\frac{\partial^2 u}{\partial t^2}\right|_{n,i} + O(\Delta t^3), \\ u_{i+1}^n &= u_i^n + \Delta x \left.\frac{\partial u}{\partial x}\right|_{n,i} + \frac{1}{2}\Delta x^2 \left.\frac{\partial^2 u}{\partial x^2}\right|_{n,i} + \frac{1}{6}\Delta x^3 \left.\frac{\partial^3 u}{\partial x^3}\right|_{n,i} + O(\Delta x^4), \\ u_{i-1}^n &= u_i^n - \Delta x \left.\frac{\partial u}{\partial x}\right|_{n,i} + \frac{1}{2}\Delta x^2 \left.\frac{\partial^2 u}{\partial x^2}\right|_{n,i} - \frac{1}{6}\Delta x^3 \left.\frac{\partial^3 u}{\partial x^3}\right|_{n,i} + O(\Delta x^4) \end{aligned}$$

These are then substituted into the finite-difference expression to obtain the modified equation:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} + \frac{\Delta t}{2}\frac{\partial^2 u}{\partial t^2} + c\frac{\Delta x^2}{6}\frac{\partial^3 u}{\partial x^3} = O(\Delta t^2, \Delta x^4). \tag{3.48}$$

The discrete formula in (3.47) is thus consistent, since if the number of degrees of freedom is increased, Δt and Δx will be decreased, and we will regain the original PDE. Explicitly writing the leading truncation error terms in this way can give some indication of how the solution will behave. If the leading truncation error term had been $\frac{\Delta x}{2}\frac{\partial^2 u}{\partial x^2}$, we would anticipate a scheme which has an additional artificial-dissipation like behaviour, and is thus likely to be dissipative. On the other hand, leading odd-order spatial derivative terms, like the one found here, tend to produce dispersive behaviour.

3.3.2 Confirming stability: Fourier analysis

In the previous section we established the consistency of the explicit Euler in time, central in space discretisation for the linear advection equation (3.47). We now seek to establish the stability of the method, so that indirectly we prove that it is convergent. To prove that the method is stable, we will assume that the problem under consideration is on a periodic domain, and employ a technique called Fourier Analysis.

Since we are on a periodic domain, we may re-express the numerical solution in terms of a spatial Fourier series (figure 3.16) which evolves in time:

$$u_i^n = \sum_m b_m(t) e^{\mathrm{I} k_m x} \tag{3.49}$$

Here the wavenumber of Fourier component m is $k_m = \frac{2\pi}{l_m}$, where l_m is its wavelength, and I is $\sqrt{-1}$. As shown in figure 3.17, the shortest wavelength which can be resolved on a mesh with spacing Δx is $2\Delta x$ (this is sometimes referred to as the "two-delta" wave). Consequently, the maximum wave number which should be considered in a Fourier analysis is $k_m = \frac{\pi}{\Delta x}$. Normally the results of a Fourier analysis are expressed in terms of the non-dimensional wave number $\beta = k_m \Delta x$, for which the relevant magnitudes are from 0 to π.

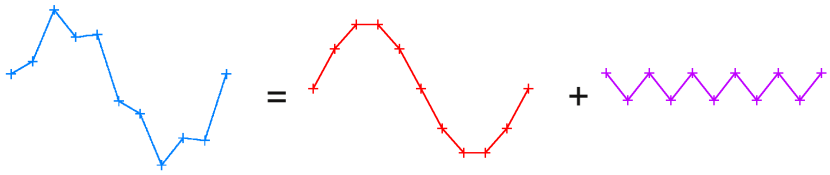


Figure 3.16: Decomposition of a numerical solution into Fourier components

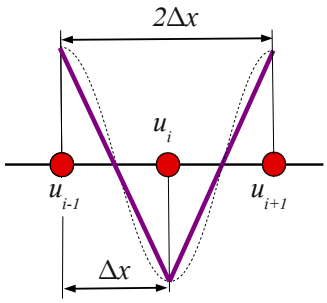


Figure 3.17: The shortest wavelength which can be represented on a mesh with spacing Δx is $2\Delta x$.

Since the finite-difference equation is linear, we may consider the evolution of one Fourier component at a time, and sum them up later to obtain the complete evolution of the solution. We therefore consider a solution made up of single arbitrary Fourier component, with an assumed exponential time variation:

$$u_i^n = U e^{at} e^{\mathrm{I} k_m x} \tag{3.50}$$

where $t = n\Delta t$, $x = i\Delta x$ and a may be complex. Substituting this solution into the finite-difference equation:

$$u_i^{n+1} = u_i^n - \frac{c\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) \tag{3.51}$$

leads to:

$$\begin{aligned} U e^{a(t+\Delta t)} e^{\mathrm{I} k_m x} &= U e^{at} e^{\mathrm{I} k_m x} \\ &\quad - \frac{c\Delta t}{2\Delta x}\left(U e^{at} e^{\mathrm{I} k_m (x+\Delta x)} - U e^{at} e^{\mathrm{I} k_m (x-\Delta x)} \right) \end{aligned} \tag{3.52}$$

or:

$$\rho \equiv e^{a\Delta t} = 1 - \frac{c\Delta t}{2\Delta x}(e^{\mathrm{I} k_m \Delta x} - e^{-\mathrm{I} k_m \Delta x}) \tag{3.53}$$

$$= 1 - \mathrm{I}\frac{c\Delta t}{\Delta x}\sin(\beta) \tag{3.54}$$

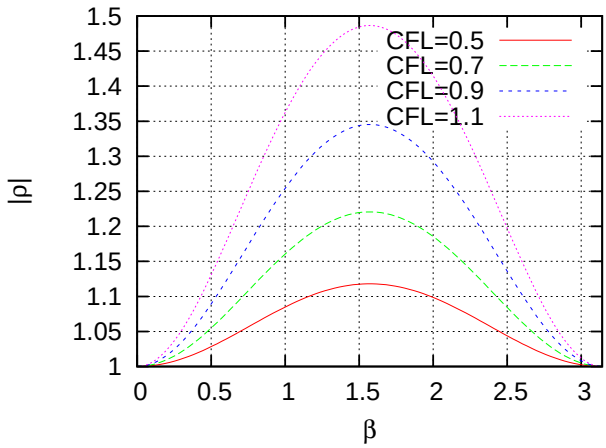


Figure 3.18: Amplification factor versus $\beta = k_m \Delta x$ for an explicit Euler in time, second-order central in space discretisation of the linear advection equation. Results for different values of $CFL = \frac{c\Delta t}{\Delta x}$ are shown.

where $\beta = k_m \Delta x$ and $\rho = e^{a\Delta t}$ is called the amplification factor. Consider the ratio of u_i^{n+1} over u_i^n (i.e (3.50) evaluated at time t and $t + \Delta t$). If the wave is not to increase its magnitude in time, then:

$$|\rho| = |e^{a\Delta t}| \leq 1 \tag{3.55}$$

From (3.54) we see that an explicit Euler time march combined with second-order central differencing results in $|\rho| > 1$ for almost all of the frequency range, for any combination of discretisation parameters (this is also shown in figure 3.18). We may think of any given solution as a combination of sinusoidal waves. Stepping repeatedly in time with this method will thus result in the exponential amplification of almost all of the solution components, meaning that the method is unstable. An example solution is shown in figure 3.19. In summary, the explicit Euler in time, second-order central in space discretisation of the linear advection equation is consistent, but it is not stable. Therefore it is not convergent.

Now consider the explicit Euler in time, first-order upwind method:

$$u_i^{n+1} = u_i^n - \frac{c\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) \tag{3.56}$$

in this case we have:

$$\rho = 1 - \frac{c\Delta t}{\Delta x}\left(1 - e^{-\mathrm{I}\beta}\right) \tag{3.57}$$

$$\tag{3.58}$$

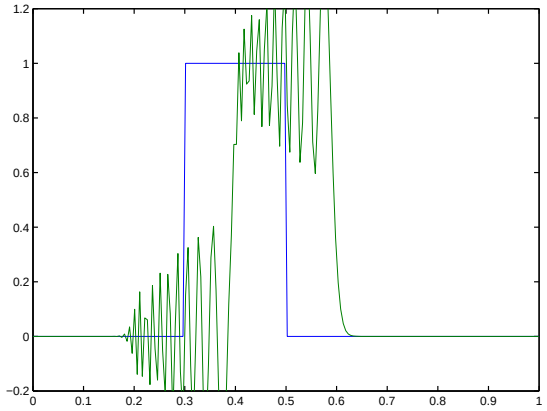


Figure 3.19: Solution from an explicit Euler in time, second-order central in space discretisation of the linear advection equation (CFL number = 0.1). The solution grows exponentially in time.

Making use of the double angle formulas $\cos(2a) = 1 - 2\sin^2(a)$ and $\sin(2a) = 2\sin(a)\cos(a)$,

$$\rho = 1 - 2\frac{c\Delta t}{\Delta x}\sin^2\frac{\beta}{2} - 2\mathrm{I}\frac{c\Delta t}{\Delta x}\sin\frac{\beta}{2}\cos\frac{\beta}{2} \tag{3.59}$$

so that:

$$|\rho| = \sqrt{1 - 4\frac{c\Delta t}{\Delta x}\left(1 - \frac{c\Delta t}{\Delta x}\right)\sin^2\frac{\beta}{2}}$$

this method is thus conditionally stable provided $\frac{c\Delta t}{\Delta x} \leq 1$ The ratio $\frac{c\Delta t}{\Delta x}$ comes up frequently in the discretisation of hyperbolic PDEs, and is referred to as the Courant-Friedrichs-Lewy (CFL) number. In figure 3.20 the amplification factor for this method is plotted versus β. At stable CFL numbers, the highest frequencies of the solution are significantly decreased in amplitude each time step, which accounts for the rounded appearance of the example solution shown in figure 3.21. This explicit Euler in time, first-order upwind in space discretisation is thus both consistent and stable, and therefore convergent, provided that $\frac{c\Delta t}{\Delta x} \leq 1$.

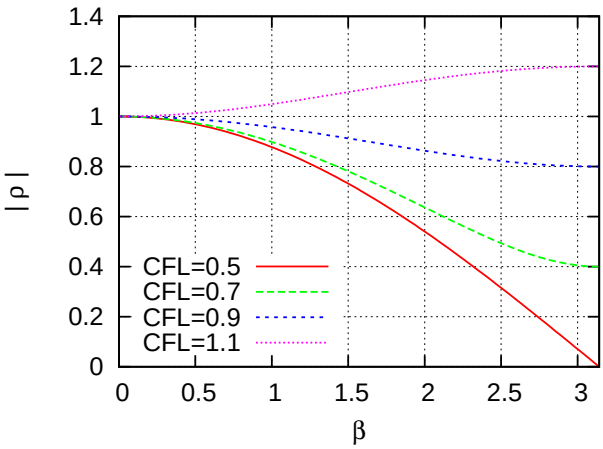


Figure 3.20: Amplification factor versus $\beta = k\Delta x$ for the an explicit Euler in time, first-order upwind in space discretisation of the linear advection equation

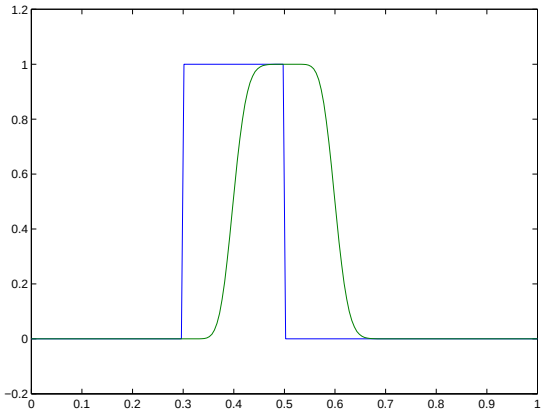


Figure 3.21: Solution from an explicit Euler in time, first-order upwind in space discretisation of the linear advection equation (CFL number = 0.1)

Chapter 4

Verification

When performing simulations one is ultimately interested in how well the computed results compare with reality. Such comparisons are part of the validation process (chapter 1), and simultaneously measure all possible sources of error in the simulations, including both model and discretisation errors. Yet the evaluation of model errors alone is often costly and complex. It is therefore illogical to consider them before quantifying the errors which can be easily estimated, those occurring between the model and the numerical representation of the model. This quantification occurs during the process of verification.

Verification can be split into two activities, *code verification* in which the consistency of the program used for the simulations with the model PDEs is established, and *solution verification* in which the errors associated with a particular case of interest are estimated. Both topics will be discussed in the following sections. The responsibility for verification ultimately lies with the user of computational modelling tools. The procedures required are straightforward, and should be considered a standard part of any professional work.

4.1 Code verification

Although we may have carefully chosen and implemented a suitable algorithm, there is always the possibility that conceptual errors or programming bugs have introduced inconsistencies into a code we intend to use for computational modelling. On the other hand, when using a code programmed by others, the assumptions made in its formulation and the quality of its implementation are by definition uncertain. Therefore a crucial part of any computational modelling exercise is to verify if the program being used is consistent with the chosen model. This should be performed before using the code for general problems of interest.

Code verification can involve a number procedures meant for testing different aspects of an algorithm. We will focus on the most important procedure, the *order-of-accuracy test*, which verifies that the solutions generated by the

code converge to exact solutions of the model equations with the correct order of accuracy. Note that the order-of-accuracy test is not able to catch all conceivable coding errors. Errors which affect the behaviour of an iterative solution procedure, for example, might not be detected. Experience has shown, however, that the order-of-accuracy test can pick up quite subtle coding problems which might otherwise go unnoticed. At the very least, a successful order-of-accuracy test will confirm that there are no remaining errors that prevent the results from the code from being useful.

The order-of-accuracy test requires an exact solution to the model equations. This should not be a trivial solution, since it is important that all features of the code are adequately exercised. Complex exact solutions can be generated using the method of manufactured solutions, which is explained next.

4.1.1 The method of manufactured solutions (MMS)

The method of manufactured solutions [24, 25] can be used to generate exact reference solutions for any type of PDE, on any type of problem domain. It is based on a reversal of the normal order of operations. Rather than finding a solution to a problem which results from the choice of certain boundary conditions and source terms, one assumes a solution and finds the corresponding problem. Consider for example, the non-linear Burgers equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0; \qquad \text{on} \quad 0 \leq x \leq \pi \tag{4.1}$$

As an exact solution we assume $u_e(x,t) = \sin(x)\cos(t)$. Substitution into (4.1) gives us the non-zero right-hand side:

$$S(x,t) = -\sin(x)\sin(t) + \sin(x)\cos(t)\cos(x)\cos(t) + \nu\sin(x)\cos(t) \tag{4.2}$$

We now define the problem as

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = S(x,t); \qquad \text{on} \quad 0 \leq x \leq \pi \tag{4.3}$$

$$u(0,t) = u(\pi,t) = 0 \tag{4.4}$$

$$u(x,0) = \sin(x) \tag{4.5}$$

where the specified boundary values and initial condition have been defined to be compatible with the assumed exact solution. The source term and boundary conditions can now be implemented within a code in order to test its ability to reproduce the exact manufactured solution.

If the differential operators within a model are complex, it may be necessary to resort to symbolic manipulation programs to find a manufactured solution. This is generally effective, as such programs can usually convert complex source expressions directly into computer code. Alternatively, one can consider sub-solutions for certain terms of the PDE in order to test certain parts of a program. This can be helpful for focusing debugging efforts.

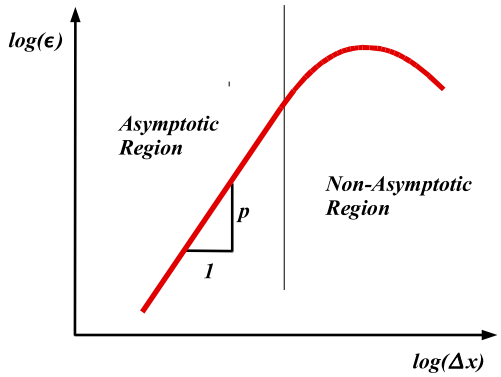


Figure 4.1: Log of error plotted versus the log of the mesh spacing for a scheme of order p

When generating manufactured solutions one should be aware that not all assumed forms are sufficient for verification. For example, if we had chosen a solution of the form $u_e(x,t) = x\cos(t)$, above, the second derivative term in (4.1) would not have been active, and it would be impossible to verify the implementation of the viscous term. Aside from this, it is generally advantageous to use smooth assumed solutions so that the correct order of accuracy can be obtained without resorting to excessively fine meshes. Furthermore, the use of discontinuous solutions might violate the assumptions on which order-of-accuracy estimates are based.

4.1.2 The order-of-accuracy test

With an exact solution at hand, a code can be verified by comparing results at different levels of refinement and ensuring that the computed solutions converge to the exact solution with the expected rate. For example, a one-dimensional finite-difference discretisation might be designed to produce errors of the form:

$$\epsilon = u_e(x,t) - u(x,t) = O(\Delta x^p) + O(\Delta x^{p+1}) + ... \tag{4.6}$$

Such a discretisation is said to have an order of accuracy p. If Δx is sufficiently small, then the $O(\Delta x^{p+1})$ and higher terms should be much smaller than the $O(\Delta x^p)$ term. When this is true the discretisation is said to be in the *asymptotic region*. We would then expect an order p convergence rate of the error, ϵ. In general the convergence rate can be verified by plotting the log of the error versus the log of the step size Δx (figure 4.1). When the leading error terms dominate, a line with constant slope should be obtained. If the slope of this line agrees with the expected slope of p, then the order-of-accuracy test has been passed.

While performing order-of-accuracy tests it is possible to encounter levelling-out behaviours similar to those shown in figure 4.2. These can be an indication

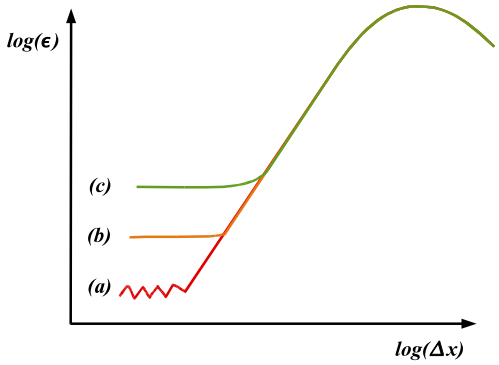


Figure 4.2: Levelling off of convergence due to (a) round-off error, (b) insufficient tolerances for iterative solution procedures, (c) incompatible refinement

of insufficiently low tolerances in an iterative solution procedure, or the lower threshold imposed by round-off error. Levelling off can also occur due to inconsistent refinement. Consider a finite-difference discretisation for a 1D space-time problem which produces errors of the form:

$$\epsilon = u_e(x,t) - u(x,t) = O(\Delta t) + O(\Delta x^2) + ... \tag{4.7}$$

Typically time and space convergence studies are performed separately. If the spatial convergence, for example, is examined with an insufficiently refined time step, then it will level off at a fixed level of (time) discretisation error. Note that for errors of the form given in (4.7), a factor of 4 refinement in time is required to be compatible with a factor 2 refinement in space.

4.2 Solution verification

After a code has been verified, computations are normally performed for a problem of interest for which the solution is unknown. To be useful, results of such computations should include estimates of their numerical error.

As mentioned in Chapter 1, there are a number of method-specific approaches to estimating discretisation errors. We will leave the description of these to other courses, and instead consider an approach to error estimation which is independent of the discretisation method, called Richardson extrapolation. After that, we will briefly discuss the estimation of model errors which occur due to domain truncation.

4.2.1 Richardson extrapolation

Error estimation via Richardson extrapolation is closely related to the order-of-accuracy test. Assume that we have a method which is p-th order accurate.

Then we can can anticipate that a value, f_d, which is a simple function of the discrete solution, can be expressed in terms of its value for the exact solution, f_e as:

$$f_d = f_e + gh^p + O(h^{p+1}) \tag{4.8}$$

where g is a function of the gradients of the continuous solution, and h is the step size (e.g. $\Delta x, \Delta y$ or Δt). Now consider the solution from a fine mesh h_1 and a coarser mesh h_2, with mesh spacings related by the ratio:

$$r = \frac{h_2}{h_1} \tag{4.9}$$

replacing h_1 by h, (4.8) can be written for the values computed on both meshes as:

$$f_1 = f_e + gh^p + O(h^{p+1}) \tag{4.10}$$
$$f_2 = f_e + g(rh)^p + O\left((rh)^{p+1}\right) \tag{4.11}$$

If we neglect the higher-order terms, these equations can be solved for an estimate of the exact solution:

$$f_e = f_1 + \frac{f_1 - f_2}{r^p - 1} \tag{4.12}$$

so that the estimation of discretisation error is:

$$\epsilon = f_e - f_1 = \frac{f_1 - f_2}{r^p - 1} \tag{4.13}$$

This of course relies on the assumption that the meshes h_1 and h_2 lie in the asymptotic region of the discretisation.

In order to confirm that we are in the asymptotic region, the *observed order of accuracy*, p_o, must be equal to the expected order of accuracy, p. The observed order of accuracy can be computed using a third mesh with spacing h_3 where $h_3 = rh_2$. We then can write

$$f_1 = f_e + gh^{p_o} + O(h^{p_o+1}) \tag{4.14}$$
$$f_2 = f_e + g(rh)^{p_o} + O\left((rh)^{p_o+1}\right) \tag{4.15}$$
$$f_3 = f_e + g(r^2h)^{p_o} + O\left((r^2h)^{p_o+1}\right) \tag{4.16}$$

which, neglecting higher-order terms, can be solved for the observed order of accuracy:

$$p_o = \frac{\ln\left(\frac{f_3 - f_2}{f_2 - f_1}\right)}{\ln(r)} \tag{4.17}$$

If p_o differs substantially from the expected asymptotic value, then the use of Richardson extrapolation is invalid.

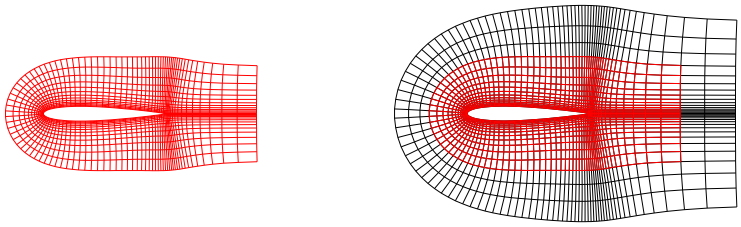


Figure 4.3: Meshes for an artificial boundary study

Occasionally it is necessary to compute the observed order of accuracy from solutions on meshes which are not related by the same grid spacing ratio. If the meshes are related by the arbitrary ratios $r_{12} = \frac{h_2}{h_1}$ and $r_{23} = \frac{h_3}{h_2}$ the equation for p_o is no longer explicit:

$$\frac{f_3 - f_2}{r_{23}^{p_o} - 1} = r_{12}^{p_o} \left(\frac{f_2 - f_1}{r_{12}^{p_o} - 1} \right) \tag{4.18}$$

and must be solved using an iterative procedure.

4.2.2 Artificial boundary studies

As mentioned in chapter 2, the truncation of a physical domain to produce a limited computational domain results in model errors. If the solution outside the domain is known to be smoothly varying, these can be estimated in a straightforward way by comparing the solutions computed on domains of different sizes. This is called an artificial boundary study, and is strictly speaking, a validation rather than a verification process. It is mentioned here since, like the solution verification procedures described above, it can be performed with minimal effort.

Artificial boundary studies are easiest to interpret when the mesh in the interior of the domain is fixed, and layers are added to increase the outer boundary distance (figure 4.3). Unlike in procedures for quantifying discretisation errors, it can be difficult to estimate at what rate the model errors due to artificial boundaries should decrease. It is therefore common to use the difference in the solution obtained from the standard domain size and from a significantly increased domain size as an error estimate.

Chapter 5

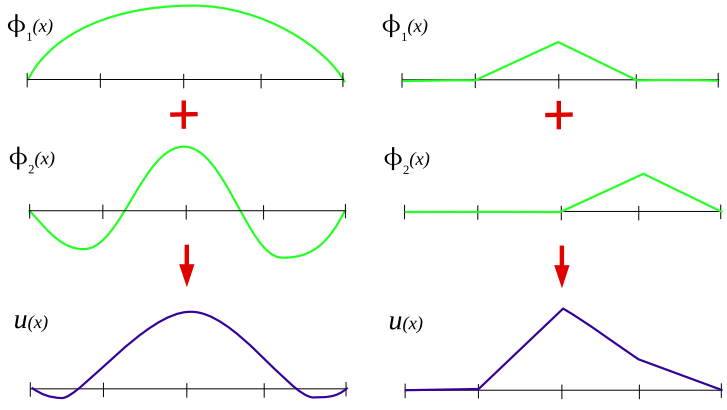
Discretisation with spectral and finite-element methods

Finite-difference methods are straight-forward to apply using structured meshes, for which gently-curved domains can be effectively treated using generalised transformations. However, it can be difficult to generate high-quality structured meshes in very complex domains. In such cases the use of unstructured meshes (including e.g. triangles/tetrahedrons and other shapes) is often favoured. Finite-difference methods can be derived for such meshes, but they are complex to implement, as the stencil may vary thought the mesh. This is not a problem for the finite-element method, however, which instead considers an integral form of the problem which allows for elements with arbitrary shapes and orientations. Finite-element methods approximate the solution with combination of known functions, and in this sense are closely related to spectral methods. Both of these will be discussed in the following sections.

5.1 Approximating the solution with functions

In contrast to approximating the solution of a PDE using point values and finite-difference expressions for the required derivatives, we now seek to approximate the solution using a combination of pre-specified *basis functions*. Examples of such functions are shown for a one-dimensional problem in figure 5.1. On the left, two functions which span the entire domain are used to construct an approximate solution. On the right, the approximate solution is constructed using two functions which are only non-zero in specific parts of the domain. For both of these cases, we can write the approximate solution as:

$$\hat{u}(x) = \sum_{i=1}^{N} a_i \phi_i(x) \tag{5.1}$$



(a) Global functions (b) Local functions

Figure 5.1: Approximation using global and local functions

where $\hat{u}$ indicates the approximation for the exact solution u, N is the number of basis functions used, and the a_i are the amplitudes of the basis functions, $\phi_i(x)$. With this approach, the a_i (often called *degrees of freedom*), are the discrete unknowns, and are determined by the numerical solution method. Methods which use global basis functions are usually referred to as *spectral methods*. The use of local basis functions, on the other hand, is the key concept behind *finite-element methods*.

Approaches to finding a_i

Once the functions are chosen, there are a number of approaches to determining their amplitudes. One approach is to define an equivalent problem which considers the minimisation of some scalar measure of the solution, $I[u]$. In structural problems, for example, this could be the total potential energy. $I[u]$ is called a *functional*, since it is itself a function of a function (in this case the function which defines $u(x)$). If $I[u]$ is to be minimised (or maximised), then a system of equations for a_i can be generated by substituting (5.1) into the definition of $I[u]$, and then requiring the derivatives of $I[u]$ with respect to the unknown a_i to be zero:

$$\frac{\partial I[u]}{\partial a_i} = 0, \qquad i = 1, ..., N \tag{5.2}$$

This is known as the Rayleigh-Ritz approach, and has its roots in the *calculus of variations*, the branch of mathematics concerned with finding the extrema of functionals. For many problems, however, there is no natural choice of a functional to be minimised or maximised. In these cases it is still possible to determine values for a_i using an approach known as the *method of weighted residuals* (MWR), which will be explained in the next section. Since it can in principal be applied to the solution of any PDE, we will concentrate on the

MWR for the remainder of the notes. For a broader view on this subject, the interested reader is encouraged to read more extensive texts, such as [15, 3, 5].

5.2 The method of weighted residuals

The method of weighted residuals provides a new form of the govening equations, known as the *weak form* which can be used to generate a system of algebraic equations for the a_i.

In order to illustrate the approach, we will make use of the following sample problem on the 1D domain Ω spanning the region from $x = 0$ to $x = 1$:

$$u_{xx} = f \quad \text{on} \quad \Omega \tag{5.3}$$
$$u(0) = g \tag{5.4}$$
$$u_x(1) = q \tag{5.5}$$

Equation (5.3) is referred to as the *strong form* of the governing equation. Equations (5.4) and (5.5) define Dirichlet and Neumann boundary conditions, respectively.

It is possible to define another form of the governing equation, known as the *weak form*, by requiring weighted integrals of the governing equation's *residual*, $R(x) = u_{xx} - f$ to be zero for all suitable weighting functions, w:

$$\int_0^1 w(u_{xx} - f)dx = 0 \tag{5.6}$$

By suitable, we mean all weighting functions which allow the integral in (5.6) to be meaningful (we shall elaborate on this shortly). [1]

We will describe how the to use the weak form to generate a system of equations for the unknown a_i in section 5.4. Before doing so, however, we will demonstrate that we do not loose anything by shifting our focus to the weak form.

5.3 Equivalence of the strong and weak forms

Spectral and finite-element methods are derived from the weak form of the governing equation rather than the strong form. It is thus important to confirm that the weak form has the same exact solution as the strong form. Clearly, the residual of the exact solution to the strong form, u, is zero, meaning that it will also satisfy the weak form. However, is it possible for the weak form to be satisfied by other solutions?

[1] It should be mentioned that many weak forms of the governing equations can be derived by approaches which consider the minimisation of a functional. As a result, weak forms are often called the *variational forms*, even when they have been obtained using the MWR.

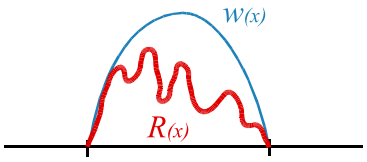


Figure 5.2: Testing the residual in a small region

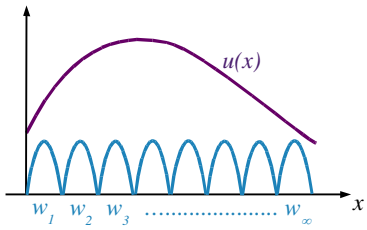


Figure 5.3: An exact solution can be obtained using an infinite number of tests of the weak form

To visualise why this cannot be true [2], consider a trial solution which produces a positive residual over some small region of the domain (figure 5.2). Since we are to express (5.6) for all suitable weighting functions, we must include a weighting function which is purely positive over this region, and zero everywhere else. Using such a weighting function would result in a non-zero result for the integral in (5.6), meaning that such a solution could not be a solution of the weak form. Note that the application of the weighted integral, as in the case of figure 5.2, provides essentially a "test" of the local residual. For this reason, weighting functions are also referred to as *test functions*.

5.4 Solving for a_i; the Galerkin method

The weak form produces the exact continuous solution when all suitable weighting functions are considered, in other words an infinite-dimensional set of functions (figure 5.3). We can also express any exact solution, u, using (5.1) with $N = \infty$ and a group of functions ϕ_i, from a similar infinite-dimensional set.

In our case, however, we wish to use finite-dimensional approximations for u, i.e. (5.1) with finite N. We therefore need only to test the weak form using a finite-dimensional set of weighting functions w_i. Specifically, for a solution interpolation defined by (5.1), no more than N weighting functions need be used, resulting in N equations similar to (5.6) which can be used to solve for the unknown a_i. Solving for the mode amplitudes in this way is known as the

[2] More formal proofs of the equivalence of the strong and weak forms can be found in [15] and [17].

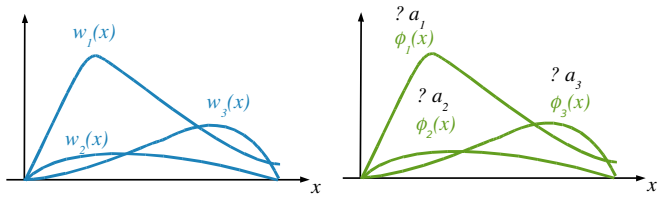


Figure 5.4: In the Bubnov-Galerkin method, the weighting functions w_i are chosen to be the same as those used to expand the solution, ϕ_i

Galerkin method [3].

The most popular Galerkin method is to choose the weighting functions to be equivalent to the solution basis functions, i.e. $w_i = \phi_i(x)$ (figure 5.4). This is formally known as the *Bubnov-Galerkin method*, although the Bubnov is often omitted. Bubnov-Galerkin are popular as they maximise the reuse of information, and produce symmetric matrices for certain classes of problems. All the examples given in these notes will be based on the Bubnov-Galerkin method, but the extension to Petrov-Galerkin methods (those which use weighting functions which are different from the solution expansion functions) is straightforward.

To see how an algebraic equation is obtained with each test of the weak form, consider the first test of (5.6) in a Bubnov-Galerkin method with $w_1 = \phi_1(x)$ and $N = 3$. Substuting the known functions $\phi_i(x)$ into (5.6) produces:

$$\int_0^1 w(u_{xx} - f)dx = 0$$

$$\rightarrow \int_0^1 \phi_1 \quad \left(a_1 \frac{\partial^2 \phi_1}{\partial x^2} + a_2 \frac{\partial^2 \phi_2}{\partial x^2} + a_3 \frac{\partial^2 \phi_3}{\partial x^2} - f\right) \; dx = 0$$

where we have used (5.1) differentiated twice in x, and noted that only the ϕ_i are functions of x. Since the a_i are not functions of x, this can be written

$$\left[\int_0^1 \phi_1 \phi_{1xx} \; dx\right] a_1 + \left[\int_0^1 \phi_1 \phi_{2xx} \; dx\right] a_2 ... - \left[\int_0^1 \phi_1 f \; dx\right] = 0$$

As f and the ϕ_i are known functions, the [ ] terms can be evaluated, resulting in an algebraic equation of the form:

$$C_1 a_1 + C_2 a_2 + C_3 a_3 = C_f$$

[3] The Galerkin method can be also related to an older approach based on the MWR. If we test (5.6) using delta functions as weighting functions the integration is bypassed, which results in a system which directly enforces a zero residual at a set of points. This approach is known as the collocation method [12].

In order to generate a system of three equations for the three unknown a_i, we could test the weak form again with $w_2 = \phi_2(x)$ and $w_3 = \phi_3(x)$. For our sample problem problem, however, we will also need to apply boundary conditions, which can be done as described in the following sections.

5.5 Requirements for the weak form, w and ϕ

As we shall see later on, the local piecewise linear functions shown in figure 5.1(b) can provide both accuracy and flexibility. If we would like to use them for our sample problem, however, we are faced with the issue that the evaluation of $\hat{u}_{xx}$ would result in zero [4]. This means that if (5.6) is used, we would be forced to reject piecewise linear functions in favour of those with a non-zero u_{xx} in order for the weak form to make sense. However, we are free to use other, mathematically equivalent versions of the weak form. In this case, the problem can be circumvented by re-expressing (5.6) using integration by parts:

$$wu_x\Big|_0^1 - \int_0^1 w_x u_x dx - \int_0^1 wf dx = 0 \tag{5.7}$$

This formulation only requires us to evaluate first derivatives, which are well defined for the functions in 5.1(b). This form is also better suited for a Bubnov-Galerkin discretisation, as the requirements for the weighting functions and solution basis functions are similar [5].

Although (5.7) is mathematically equivalent to (5.6) it includes a term which is only evaluated on the boundaries: $wu_x\Big|_0^1$. If u_x is to be determined as part of the solution (e.g. when numerical conditions are to be applied) this term can be re-expressed in terms of the unknown a_i by taking the derivative of the appoximate solution

$$\frac{\partial \hat{u}}{\partial x} = \sum_{i=1}^{N} a_i \frac{\partial \phi_i}{\partial x} \tag{5.8}$$

If instead a physical boundary condition is to be applied, (e.q. a boundary value of $u_x = q$ is to be specified), we have two options. The first option is to drop one of the algebraic equations generated by the weak form and in its place write an equation which sets the local discrete derivative to q:

$$\sum_{i=1}^{N} a_i \frac{\partial \phi_i}{\partial x} = q \tag{5.9}$$

This is known as a strong Neumann boundary condition. The second option is to retain the algebraic equations obtained from the weak form, and simply

[4] Note that kinks of the curves where u_{xx} is undefined do not affect the value of the integral

[5] Integrating by parts in order to move derivatives to the weighting functions can also be useful in certain non-linear problems, to avoid taking gradients of complicated functions of the solution.

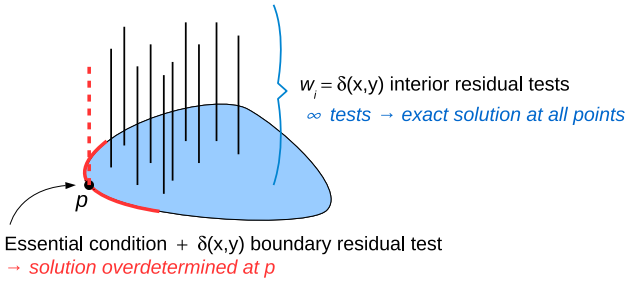


Figure 5.5: Exact solution obtained by testing the residual with an infinite number of Dirac delta functions.

replace the value of u_x with its known value. For example, applying (5.5) leads to:

$$w(1)q - w(0)u_x(0) - \int_0^1 w_x u_x dx - \int_0^1 wf dx = 0 \tag{5.10}$$

This is known as a weak Neumann boundary condition. This approach is quite different from what we used up to now, in the PDE and a physical boundary condition appear in the same equation. It is only possible because the Neumman boundary condition appears explicity in the chosen weak form. When boundary conditions appear explicitly in the weak form, they are said to be natural boundary conditions. For our example PDE, Neumman conditions are natural for the weak form (5.7), but not for the weak form (5.6).

On the other hand, (5.7) does not allow for the application of a Dirichlet boundary condition. Thus the Dirichlet condition (5.4) will have to be specified by deleting one of the algebraic equations obtained from the weak form, and replacing it with an equation like:

$$\sum_{i=1}^{N} a_i \phi_i(0) = g \tag{5.11}$$

When a boundary condition does not appear explicitly in the weak form, it is said to be an essential condition. We refer to an expression such as (5.11) as a strong Dirichlet BC.

When applying the weak form using a Bubnov-Galerkin method, we generate N equations for the N unknown a_i. If we have E essential boundary conditions, however, we will have to remove E of these equations to have as many equations as unknowns. In other words, we will not test the residual with E of the possible w_i. This begs the question which of the w_i do we reject as test functions? For inspiration, we consider an exact solution of the problem, obtained by testing the residual using delta functions placed at an inifinite number of points in the domain (figure 5.5). If we are specifing the solution at a point p on the boundary, it is clear that the w_i we should reject is the delta function which

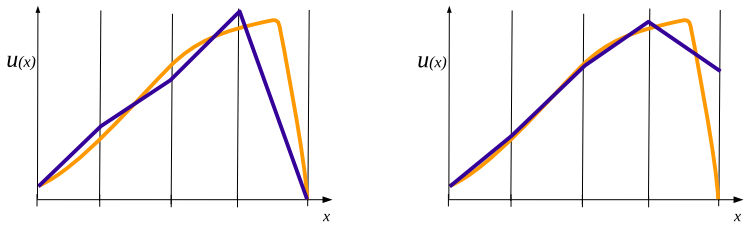


Figure 5.6: Discrete solutions obtained for a problem with a strong gradient near the right boundary using strong (left) and weak (right) Dirichlet conditions

tests the residual of the PDE at p. In other words, there is little value in including an equation which tests the residual at a point where the behaviour of the solution is already specified by an essential boundary condition. Thus *when applying essential boundary conditions, the w_i we will reject are ones which are non-zero on the boundary.*

In the following section, it will be seen that the replacement of a test of the residual with an algebraic relation directly specifying the Dirichlet condition $u = g$ resembles the finite-difference approaches we considered previously, in which the finite-difference approximation of the PDE on a Dirichlet boundary was dropped in favour of the direct expression of the boundary condition. However, we will also demonstrate how a Neumann boundary condition can be applied weakly using the weak form (5.7). This approach has no finite-difference counterpart.

Even when a boundary condition is natural to the weak form employed, we could choose to treat it as an essential condition and apply it strongly. The advantage of using a weak condition, however, is that can potentially improve the numerical solution by balancing interior discretisation errors with errors in the reproduction of the boundary condition. An example is shown in figure 5.6.

A final requirement for the Galerkin method to work is that the weak form considered remains integrable. This is not a trivial requirement. For example, if $u_x = \frac{1}{x^2}$, and w is chosen as x, the second definite integral in (5.7) would evaluate to ∞. For a Bubnov-Galerkin discretisation of (5.7), we can define the set of suitable functions as those which have square-integrable derivatives, defined as:

$$\int_0^1 (\phi_x)^2 dx \ < \ \infty \tag{5.12}$$

Note that square integrability of the derivative can be a stronger restriction than integrability of the function itself.

Summarising, the basic requirements for the weak form, w and ϕ in the Galerkin method are:

- The chosen combination of functions and weak form must produce a non-

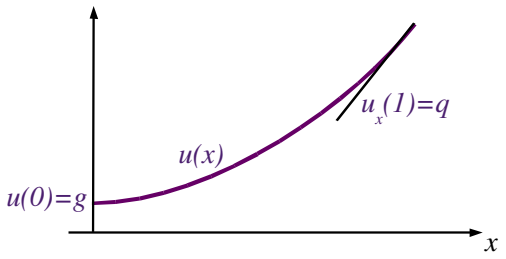


Figure 5.7: Solution of the sample problem

trivial algebraic system

- When E essential boundary conditions are applied, E w must be rejected. These should be w that are non-zero on the essential boundary
- The weighting and basis functions must be chosen such that the weak form is integrable

5.6 An example spectral method

We will now proceed to find an approximate solution to the sample problem of section 5.2, which has the form illustrated in figure 5.7. We will use a spectral method, that is, one employing basis functions defined over the complete domain. The basis functions will be chosen from the "modal p-type expansion" defined in [18]:

$$\begin{aligned}\phi_1(x) &= 1-x \\ \phi_2(x) &= x \\ \phi_3(x) &= x-x^2 \\ \phi_4(x) &= -2x+6x^2-4x^3\end{aligned}$$

These functions allow the interpolation of polynomial solutions of up to third order, and are illustrated in figure 5.8. We will express our approximate solution as:

$$\hat{u}(x) = \sum_{i=1}^{4} a_i \phi_i(x) \tag{5.13}$$

and employ a Bubnov-Galerkin method for which $w_i = \phi_i$.

5.6.1 The basic system

We can obtain a system for the four unknowns in the problem by testing the integrated-by-parts weak form of the equations (5.7) with $w_{1\to4}$. Noting that

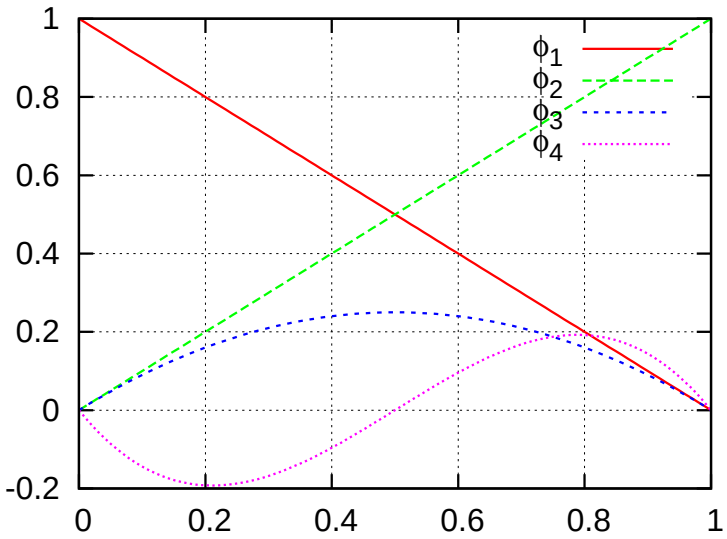


Figure 5.8: Modal basis functions

only $w_1 = \phi_1$ is non-zero at $x = 0$, and only $w_2 = \phi_2$ is non-zero at $x = 1$, this results in the following basic system of equations:

$$w_1(0)u_x(0) - \int_0^1 w_{1_x} \sum_{i=1}^{N} a_i \phi_{i_x} dx = \int_0^1 w_1 f dx \tag{5.14}$$

$$w_2(1)u_x(1) - \int_0^1 w_{2_x} \sum_{i=1}^{N} a_i \phi_{i_x} dx = \int_0^1 w_2 f dx \tag{5.15}$$

$$- \int_0^1 w_{3_x} \sum_{i=1}^{N} a_i \phi_{i_x} dx = \int_0^1 w_3 f dx \tag{5.16}$$

$$- \int_0^1 w_{4_x} \sum_{i=1}^{N} a_i \phi_{i_x} dx = \int_0^1 w_4 f dx \tag{5.17}$$

5.6.2 Dirichlet boundary condition

We can start by applying the Dirichlet boundary condition at $x = 0$. Examining the solution basis functions, we see they are all zero at $x = 0$ except for ϕ_1, which has a value of 1. Expressing the Dirichlet boundary condition using (5.13) we obtain the essential condition $a_1 = g$. Based on the discussion in the previous section, the test with $w_1 = \phi_1$ will be removed from consideration as w_1 is the only weighting function that is non-zero on the the Dirichlet boundary. As we already know a_1, however, and have only three remaining unknowns (a_2, a_3 and

a_4), a test with w_1 will not be required. We then have:

$$a_1 = g \tag{5.18}$$

$$w_2(1)u_x(1) - \int_0^1 w_{2_x} \sum_{i=1}^{N} a_i \phi_{i_x} dx = \int_0^1 w_2 f dx \tag{5.19}$$

$$- \int_0^1 w_{3_x} \sum_{i=1}^{N} a_i \phi_{i_x} dx = \int_0^1 w_3 f dx \tag{5.20}$$

$$- \int_0^1 w_{4_x} \sum_{i=1}^{N} a_i \phi_{i_x} dx = \int_0^1 w_4 f dx \tag{5.21}$$

5.6.3 Neumann boundary condition

The Neumann condition is natural for this weak form, so we can apply it directly by noting $u_x = q$ at $x = 1$. We then have

$$a_1 = g \tag{5.22}$$

$$w_2(1)q - \int_0^1 w_{2_x} \sum_{i=1}^{N} a_i \phi_{i_x} dx = \int_0^1 w_2 f dx \tag{5.23}$$

$$- \int_0^1 w_{3_x} \sum_{i=1}^{N} a_i \phi_{i_x} dx = \int_0^1 w_3 f dx \tag{5.24}$$

$$- \int_0^1 w_{4_x} \sum_{i=1}^{N} a_i \phi_{i_x} dx = \int_0^1 w_4 f dx \tag{5.25}$$

5.6.4 The final system

We now complete the system of equations by substituting the assumed solution (5.13) into the system and noting for a Bubnov-Galerkin method $w_i = \phi_i$.

$$\begin{aligned}
a_1 &= g \\
\phi_2(1)q - \int_0^1 \phi_{2_x} \left(a_1\phi_{1_x} + a_2\phi_{2_x} + a_3\phi_{3_x} + a_4\phi_{4_x}\right) dx &= \int_0^1 \phi_2 f dx \\
- \int_0^1 \phi_{3_x} \left(a_1\phi_{1_x} + a_2\phi_{2_x} + a_3\phi_{3_x} + a_4\phi_{4_x}\right) dx &= \int_0^1 \phi_3 f dx \\
- \int_0^1 \phi_{4_x} \left(a_1\phi_{1_x} + a_2\phi_{2_x} + a_3\phi_{3_x} + a_4\phi_{4_x}\right) dx &= \int_0^1 \phi_4 f dx
\end{aligned}$$

The above can be expressed as a matrix equation for the unknowns a_i as:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ K_{21} & K_{22} & K_{23} & K_{24} \\ K_{31} & K_{32} & K_{33} & K_{34} \\ K_{41} & K_{42} & K_{43} & K_{44} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} g \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} \tag{5.26}$$

where, for example, all terms like:

$$K_{23} = -\int_0^1 [\ \phi_{2_x}(x)\ \phi_{3_x}(x)\ ]\ dx \tag{5.27}$$

$$F_2 = \int_0^1 [\ \phi_2(x)\ f(x)\ ]\ dx - \phi_2(1)q \tag{5.28}$$

can be evaluated using known data. With the exception of the first row, which contains an essential condition, each row in the K matrix corresponds to a test of the weak form with a single weighting function. K is often referred to as the *stiffness matrix*, due to the role it plays in structural problems [6].

For the system to have a solution, the $\phi_i(x)$ must be linearly independent. This means that it is not possible to express one $\phi_i(x)$ as a linear combination of the others. The definition of K_{23} given by (5.27) also demonstrates that if one has an orthogonal basis, i.e. one for which:

$$\int_\Omega [\ \phi_i(x)\ \phi_j(x)\ ]\ d\Omega = 0; \qquad i \neq j \tag{5.29}$$

the resulting K matrix will be diagonal, and trivial to invert. When non-orthogonal bases are used, those which are closer to being orthogonal will tend to produce K matrices with increased diagonal dominance.

5.6.5 Observations and results

Spectral solutions to the sample problem with $g = 0, q = \frac{1}{2}$ and $f = x$ obtained using increasing numbers of basis functions are shown in figure 5.9. In this case the exact solution is given by $u(x) = g + qx + \frac{1}{6}(x^3 - 3x)$, shown using diamond symbols in the plot. The spectral solution converges quickly, and becomes exact when $\phi_2(x), \phi_3(x)$ and $\phi_4(x)$ are used. This occurs since once $\phi_4(x)$ is added, the solution function space is rich enough to represent the cubic variation of the exact solution without error. This phenomena is known as *superconvergence*.

5.7 An example finite-element method

When a domain has a complex shape, it can be difficult to define global functions appropriate for a spectral method. Furthermore, many realistic problems include sudden local changes in the boundary conditions, or in the governing equations themselves. To deal with these issues it is easiest to use basis functions which are only non-zero on limited parts of the domain. This is the concept behind the finite-element method. The basic procedure can be summarised as:

[6] Note that in the construction of K there is an analogy with the finite-difference example considered in section 3.1.4. In that case, to apply a Dirichlet boundary condition the first equation corresponding to the PDE evaluated at the boundary point was removed and replaced with an expression specifying the solution at the point. Thus the "test" of the PDE was not performed at the boundary, since it would lead to redundant information. The analogy does not hold for the method used to apply the Neumann boundary condition, however.

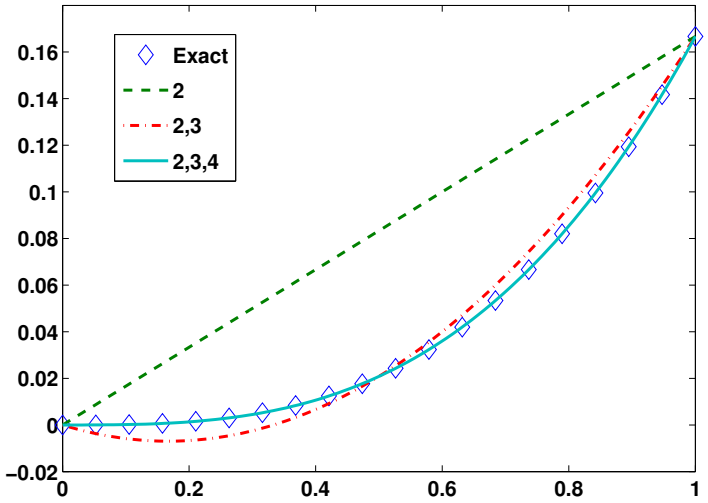


Figure 5.9: Spectral solution using an increasing number of basis functions from the modal p-type expansion

1. Divide the domain into elements
2. Define basis functions which span a small number of elements
3. Compute per-element contributions to the weak form
4. Assemble the element contributions into a global matrix
5. Apply the boundary conditions and solve the resulting system

We will consider each of these steps in turn by solving our sample problem.

Step 1

We begin by splitting the domain into three elements. These are chosen to have an equal width of $h = 1/3$, although unequal element sizes could also be used. For complex domains, division into elements is normally done using dedicated grid-generation software.

Step 2

We define four basis functions, from the group of local piecewise linear functions in figure 5.1(b). For this one-dimensional problem, we choose to have the functions span two elements, with the exception of $\phi_1(x)$ and $\phi_4(x)$, which are truncated and exist only in the boundary elements (figure 5.10).

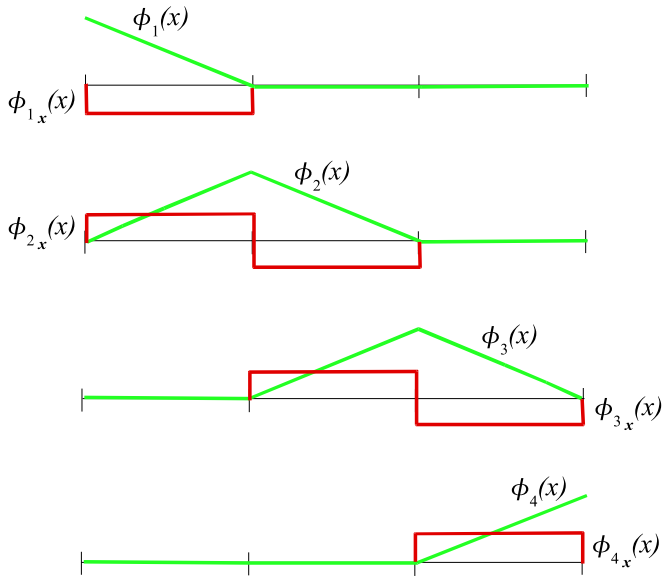


Figure 5.10: Basis functions (green) for a three-element discretisation of the sample problem. The derivatives of the basis functions are shown in red.

Step 3

Since grid generation codes produce a representation of the problem domain in terms of a group of elements, it makes sense to assemble the system of equations by looping over elements and computing their individual contributions to the K and F matrices. This of course can be done by evaluating the integrals in physical coordinates on an element-by-element basis.

However, most large finite-element codes use a limited number of element types (e.g. triangle, hexahedron, etc..) to which all elements in a mesh can be topologically related. Since some quantities remain fixed for a given element type, it makes sense to consider a master element in Cartesian coordinates and relate individual elements to the master element using a coordinate transform.

Returning to our one-dimensional problem, each of the elements in the mesh can be treated generically if we consider local versions of the basis functions defined on the master element shown in figure 5.11. In the figure ϕ_L indicates the portion of the basis function centred on the left which is contained in the element, while ϕ_R indicates the portion of the basis function centred on the right. For the case shown, the master element starts at $\xi = 0$ and ends at $\xi = 1$. When we compute the contribution from a given element in the mesh, it will have some arbitrary width h. Therefore we need to evaluate the integral in a way that accounts for this. If the element in physical space has a length h and starts at x_1, then $\xi = \frac{(x-x_1)}{h}$. x derivatives can then be obtained from the chain

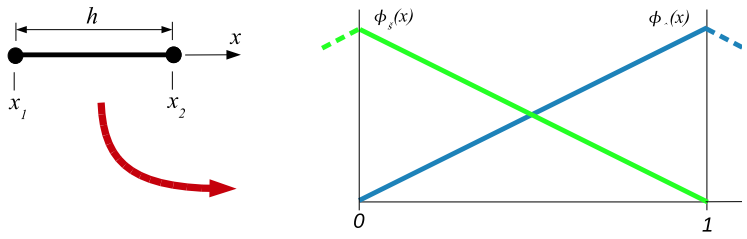


Figure 5.11: Physical element (left) and master element (right) with two basis functions

rule, so that $\phi_x = \phi_\xi \xi_x$, where $\xi_x = \frac{1}{h}$. To change integration coordinates, we note that given the physical length is h and the transformed length is 1, so that $dx = hd\xi$. The element matrix, K_e, and vector, F_e are then given by:

$$Ke = \begin{bmatrix} \frac{1}{h}\int_0^1 \phi_{L_\xi}(\xi)\ \phi_{L_\xi}(\xi)\ d\xi & \frac{1}{h}\int_0^1 \phi_{L_\xi}(\xi)\ \phi_{R_\xi}(\xi)\ d\xi \\ \frac{1}{h}\int_0^1 \phi_{R_\xi}(\xi)\ \phi_{L_\xi}(\xi)\ d\xi & \frac{1}{h}\int_0^1 \phi_{R_\xi}(\xi)\ \phi_{R_\xi}(\xi)\ d\xi \end{bmatrix} \tag{5.30}$$

$$Fe = \begin{bmatrix} -h\int_0^1 \phi_L(\xi)\ f(\xi)\ d\xi \\ -h\int_0^1 \phi_R(\xi)\ f(\xi)\ d\xi \end{bmatrix} \tag{5.31}$$

In simple cases the integrals which define the element matrices and vectors can be evaluated analytically. In practice, however, quadrature is often used, i.e.:

$$\int_\Omega f(x)\ dx \approx \sum_{ip=1}^{N_{ip}} k_{ip}\ f(x_{ip}) \tag{5.32}$$

where N_{ip} is the number of quadrature (integration) points used in the interval and k_{ip} is the weight associated with a given quadrature point. Using quadrature reduces implementation complexity and increases flexibility, allowing, for example, non-analytic input data to be used. The number and placement of quadrature points can be chosen to exactly integrate polynomials up to a given order.

Step 4

The next step is to add the per-element contributions to the global matrix and vector. Each row in the global system corresponds to a test with a certain weighting function ϕ_i. Since ϕ_i normally spans more than one element, more than one element will contribute to a given row. On the other hand, a given element will be normally touched by more than one test function, so integration of the weak forms active within an element will result in a local 'element" matrix and vector. Constructing the global matrix and vector by adding the contributions of the element matrices and vectors is called *assembly*.

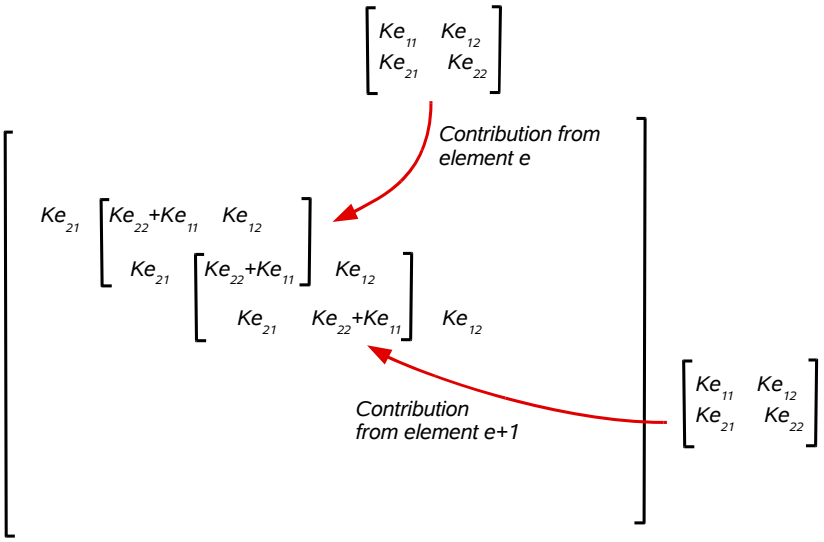


Figure 5.12: Overlapping element arrays in global matrix assembly

Consider element 2. Its ϕ_L is actually ϕ_2 and its ϕ_R is ϕ_3. The first row of its element matrix thus corresponds to row 2 of the global matrix and its first column to column 2. It also makes contributions to row 3. In the next element (element 3), ϕ_L is actually ϕ_3 and ϕ_R is ϕ_4. Element 3 thus accounts for the the second half of the equation associated with the weighting function ϕ_3. The result is that the element matrices must be added to the global matrix in an overlapped fashion, as shown in figure 5.12. A similar overlapping occurs in the right-hand side vector.

Since we chose our functions to span only two elements, using many elements will result in a matrix which is sparse and banded. In other words, it will be filled mostly with zeros except along the diagonal, and in row-column combinations corresponding to adjacent elements. Sparse matrices can be solved very efficiently using iterative or specialised direct techniques. For the PDE considered the stiffness matrix will also be symmetric. This is a consequence of our choice of the integrated-by-parts weak form combined with a Bubnov-Galerkin method, which leads to symmetry between the weighting functions and the solution basis functions. Symmetric matrices are generally easier to solve than unsymmetric ones.

Step 5

Using the same arguments as in the previous section, $a_1 = g$, and w_1 is redundant. We therefore eliminate the first row of the K matrix, and then eliminate the first column by inserting the known value for a_1 in each equation and

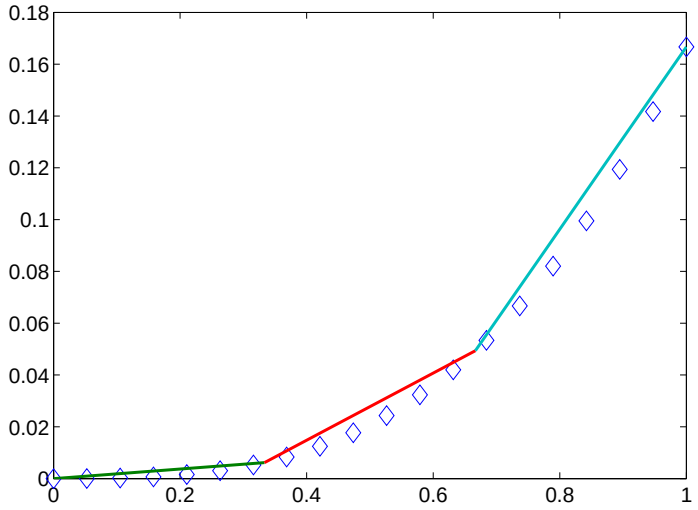


Figure 5.13: Finite-element solution of the sample problem using three elements

transferring the result to the right-hand side. Finally, we need to consider the Neumann boundary condition at $x = 1$. This can be done by adding $\phi_4(1)q$ to F_4. We can then solve the remaining system.

Results

The results for a three-element discretisation of our sample problem with $g = 0, q = \frac{1}{2}$ and $f = x$ is shown in figure 5.13. Remarkably, the solution is nodally exact. This type of superconvergence occurs due to the combination of piecewise linear basis functions with this particular weak form (see [15] for details). Convergence rates for finite-element discretisations of more general problems are discussed in the next section.

Contrast with the spectral approach

Note that the global matrix obtained in the finite-element test problem could have been obtained by using a spectral method with functions which are zero everywhere except in the vicinity of two elements, as shown in the upper two plots of figure 5.1 (b). However, performing integration element-wise, followed by global assembly, has certain advantages. If a spectral approach is used, every integral requires evaluation over the complete domain, which is generally expensive when quadrature is used. Secondly, in a parallel computing environment, all functions and unknowns need to be present on all processors. thirdly, the

boundary terms would have to be retained on all functions. In contrast, with a finite-element method integration is only performed locally. Furthermore, information exchanged across processors can be limited to that associated with the elements touching the inter-processor boundaries. Finally, only the elements adjacent to the physical boundaries need to be involved in the application of boundary conditions.

5.8 Convergence rates

For problems where superconvergence does not occur, the convergence rate of spectral and finite-element discretisations is normally given by:

$$|e| = \left(\int_\Omega (u - \hat{u})^2 \, d\Omega \right)^{1/2} \leq Ch^p \tag{5.33}$$

where C is a constant determined by the discretisation and problem data, h is a measure of the domain (spectral) or element (FEM) size, and p depends on the basis functions chosen for the discretisation.

For the piecewise-linear functions used in the previous section, normally $p = 2$. For the spectral method of section 5.6, the value of p is typically the order of the highest polynomial in the basis plus one.

It is possible to combine the spectral and finite-element methods, for example by using the basis (5.13) within each element of a group of elements which span the domain. Such *spectral-element methods* also converge with a rate given by (5.33). They can be refined either by introducing more elements (h refinement) or adding more basis functions (p refinement).

Since (5.33) is exponential in p, p refinement can be more effective than h refinement in spectral-element methods, especially when the solution is "smooth" (changes gradually between elements). When the solution is "rough", (changes rapidly between elements), h refinement tends to be more effective. h refinement may also be less expensive for a given number of degrees of freedom, as p refinement can lead to a reduction in conditioning of the assembled matrices, making them more expensive to solve (see [18] for details).

5.9 FEM in multiple dimensions

The finite-element method in multiple dimensions works with the same steps as in the one-dimensional case, but with some additional procedures associated with the division of the domain into elements, and the evaluation of integrals and gradients on elements of varying geometry. In this section we discuss the main issues associated with division of the domain into elements. Appendix D describes two approaches to dealing with arbitrary element geometries in multiple dimensions.

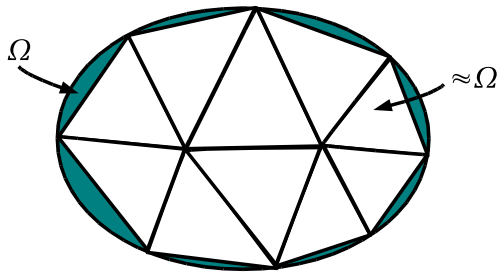


Figure 5.14: Approximate domain implied by straight element edges

5.9.1 Division of the domain into elements

Classical discretisation methods such as the finite-difference method consider the boundary of the domain to be defined by a sequence of points. This provides no definition of the functional form of the boundary, and thus many grid generation packages represent the boundaries of the domain using a series of straight-line segments. Consequently, it is common for finite-element methods to use elements with edges defined by straight-line segments, resulting in approximations of the physical domain, as shown in figure 5.14.

However, the finite-element method is not inherently limited to this representation. Curved element boundaries, typically expressed in terms of polynomials, can also been used. These are normally combined with higher-order representations of the interior solution, although this is not strictly necessary. For problems which are sensitive to boundary geometry (e.g. the flow around a curved surface) large increases in solution accuracy have been observed simply by improving the representation of the domain boundary. An example is shown in figure 5.15, where the piecewise-linear geometry representation on the left shows artificial jumps in the isodensity contours as the boundary is approached.

Recently it was realised that the limitations associated with current grid-generation packages could be bypassed if finite-element programs worked directly with the interpolation functions used in the initial computer-aided design (CAD) process [9]. These are typically non-uniform rational B-splines (NURBS). Employing them directly with a finite-element method allows an exact representation of the boundary, independent of the number of elements used to discretise the domain (figure 5.16). Having an exact boundary definition accessible also makes adaptation easier, as the exact coordinates of new boundary nodes are easily generated. A related approach is to use standard polynomial approximations for the solution, but an exact NURBS representation for the boundary. This ensures that the polynomial order of the interpolation is maintained in physical space, so that the optimal order of convergence is obtained [27, 28].

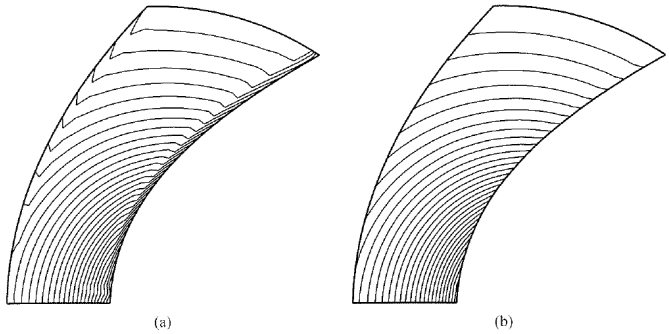


Figure 5.15: Isodensity contours of a Ringleb flow. (a) discretised with piecewise linear solution and geometry representations, (b) discretised with piecewise linear solution and quadratic geometry representations (from [2]).

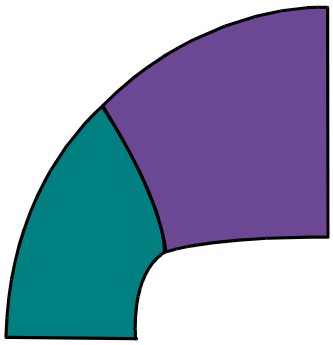


Figure 5.16: A two-element discretisation of a curved domain (from [9])

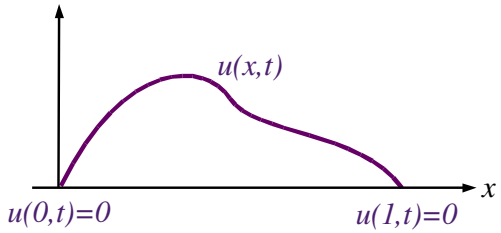


Figure 5.17: Solution of the unsteady sample problem at a time t

5.10 Unsteady problems

There are two common approaches to discretising unsteady problems using spectral and finite-element methods. In the first, known as the semi-discrete approach, the time derivatives are initially treated as additional unknowns and the spectral or finite-element method is only used to discretise the problem in space. The time derivatives are then discretised using any appropriate method, including those based on finite-difference approximations. The second approach to unsteady problems is known as the fully-discrete approach, in which both time and space are discretised using the same spectral or finite-element method. Both approaches are described in the following sections in the context of the problem below:

$$u_t - u_{xx} = 0 \qquad \text{on} \quad \Omega \tag{5.34}$$

$$u(0,t) = 0 \tag{5.35}$$

$$u(1,t) = 0 \tag{5.36}$$

which has a solution with the form illustrated in figure 5.17.

5.10.1 Semi-discrete approach

In the semi-discrete approach, the time derivatives are initially left as continuous variables in the formulation. This results in a system of ordinary differential equations which can be integrated using conventional time-marching techniques. Consider the discretisation of our sample unsteady problem. We begin by assuming:

$$\hat{u}(x,t) = \sum_{i=1}^{N} a_i(t)\ \phi_i(x) \tag{5.37}$$

and then derive a weak form of the PDE by considering the MWR applied in the spatial domain:

$$\int_\Omega w\ u_t\ d\Omega\ +\ \int_\Omega w_x\ u_x\ d\Omega = 0 \tag{5.38}$$

where we have used the fact that $w = 0$ on both Dirichlet boundaries as specified by (5.35) and (5.36). (5.37) can be substituted into (5.38), to obtain a system of the form:

$$M_{ij} \frac{da_j}{dt} + K_{ij}\, a_j = 0 \tag{5.39}$$

where the stiffness matrix, K_{ij}, has elements determined by expressions similar to (5.27) or (5.30), and the *mass matrix* M_{ij} is defined by:

$$M_{ij} = \int_\Omega \phi_i(\xi)\, \phi_j(\xi)\, d\Omega \tag{5.40}$$

Equation (5.39) is a system of ODEs which can be integrated using standard time-marching techniques. For example, using the explicit Euler method would result in:

$$a_j^{n+1} = a_j^n + (\Delta t) M_{ij}^{-1} K_{ij}\, a_j^n \tag{5.41}$$

where Δt is the time step and n is indicates the n^{th} time level. In this case, using an orthogonal basis is advantageous, since then M_{ij} is diagonal, and is thus trivial to invert.

5.10.2 Fully-discrete approach

In the fully-discrete approach, we simply consider time as an extra dimension, and apply a spectral or finite-element discretisation in the multidimensional sense. For a finite-element discretisation of our sample problem, we would assume a solution of the form:

$$\hat{u}(x,t) = \sum_{i=1}^{N} a_i\, \phi_i(x,t) \tag{5.42}$$

and discretise (5.38) on the space-time domain Q shown in figure 5.18. To do so we apply the MWR on the space-time domain Q (figure 5.19) (a slab of width Δt):

$$\int_Q w\,(u_t - u_{xx})\, dQ = 0 \tag{5.43}$$

and integrate by parts:

$$\int_\Omega \left(wu\Big|_0^{\Delta t} - \int_0^{\Delta t} w_t\, u\, dt \right) dx - \int_0^{\Delta t} \left(wu_x\Big|_0^1 - \int_\Omega w_x\, u_x\, dx \right) dt = 0$$

Applying the Dirichlet BCs at $x = 0$ and $x = 1$, we are left with

$$\int_{\Omega^{n+1}} w\, u\, dx - \int_{\Omega^n} w\, u\, dx - \int_Q w_t\, u\, dQ + \int_Q w_x\, u_x\, dQ = 0 \tag{5.44}$$

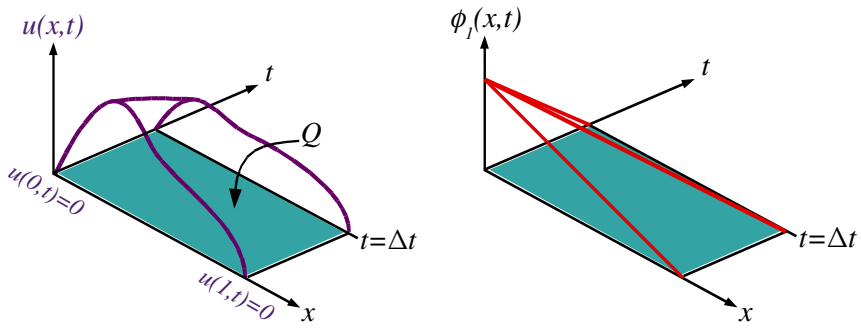


Figure 5.18: A space-time domain solution (left) and basis function (right)

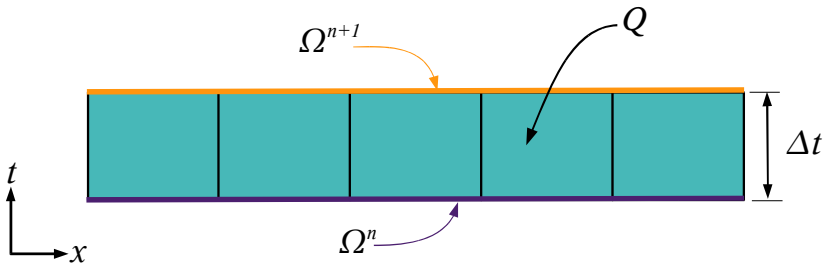


Figure 5.19: A Space-time slab, Q. Its lower boundary is given by the spatial domain Ω at time step n, and its upper boundary by Ω at time step $n+1$.

Figure 5.20: Space-time slab for an expanding spatial domain

There is usually no need to consider more than one time step in the time direction due to causality (the past does not depend on the future). Q is normally referred to as a *space-time slab*. On the lower boundary, Ω^n, the initial condition is applied as a Dirichlet condition. By virtue of the fact that the problem is parabolic in the time direction, no boundary condition is required on Ω^{n+1}. The solution at the next times step is just the computed solution on Ω^{n+1}, which is used as an initial condition for the next time step.

Space-time methods are flexible, in that they provide a straightforward way to deal with changing domains (figure 5.20). They also provide some additional choices for adaptation, such as increasing the number of time basis functions locally, so that time accuracy is enhanced in only a portion of the domain. Space-time methods are relatively expensive, however, if p-refinement is used globally to increase the order accuracy, as this has a direct impact on the size of the matrices which must be solved within a time march loop. Furthermore, their stability analysis using Fourier analysis techniques can be laborious, typically requiring the use of symbolic manipulation programs.

5.11 Further Developments

Historically, finite-element methods were first applied to structural problems, but the last few decades have seen their application to a wide range of PDEs. In certain cases, this has required the development of techniques which go beyond those we have just discussed.

A case in point is the application of finite-element methods to problems involving advection, typically encountered in fluid mechanics. The standard Bubnov-Galerkin approach is insufficient in this regard, as it tends to produce oscillatory solutions in the presence of advective phenomena. This problem was successfully overcome by employing weighting functions which differ from the solution basis functions, known as the *Petrov-Galerkin method*. It was later shown that this was comparable to introducing additional terms to a Bubnov-Galerkin method which account for unresolved components of the solution, leading to the concept of *variational multiscale methods*. These have since been applied to a range of challenging problems.

A number of other successful approaches to finite-element discretisation have also been developed. These include *least-squares methods* and *discontinuous Galerkin methods*. The interested reader is encouraged to investigate their respective advantages by consulting the latest scientific literature.

Chapter 6

Time march methods

For unsteady problems, one is usually concerned with how well certain transient components of the solution are reproduced. As shown in the previous chapters, however, a variety of different numerical solution methods can be derived using different approaches to space and time discretisation. These can differ substantially in their inherent stability, and ability to reproduce the behaviour of the exact solution. This section will describe techniques which can be used to analyse numerical solution methods for unsteady problems, and anticipate their behaviour. Although the approaches considered here are strictly only valid for linear PDEs and linear discrete operators, in practice their results have been found to be useful in the interpretation of a wide range of numerical schemes.

6.1 Accuracy of transient computations

It is normal when designing numerical methods to establish their order of accuracy. For finite difference methods, for example, this can be done by the substitution of Taylor series expansions into the differential operator, as is done in the consistency check explained at the end of chapter 3. Yet the order of accuracy may be insufficient information for deciding on a time march method. In unsteady problems, one is also interested in the absolute accuracy with which transient phenomena of a certain wavelength are represented. Fortunately we can re-use the results of the Fourier analysis introduced in chapter 3 to provide such information.

6.1.1 The exact amplification factor

In order to evaluate the accuracy of a numerical scheme for solution components of a certain wavelength, exact values are required for reference. These can can be obtained by considering the exact behaviour of the amplification factor implied by the original PDE. This procedure is demonstrated below for two example PDEs, the linear advection and linear diffusion equations.

The linear advection equation

In the Fourier analysis procedure, we considered the generic discrete solution component:

$$u_i^n = Ue^{at}e^{\mathrm{I}k_m x} \tag{6.1}$$

An expression for the exact evolution of this component for the linear advection equation can be obtained by substituting (6.1) directly into $u_t + cu_x = 0$:

$$ae^{at}e^{\mathrm{I}k_m x} + c\mathrm{I}k_m e^{at}e^{\mathrm{I}k_m x} = 0 \tag{6.2}$$

this implies that $a = -\mathrm{I}ck_m$, so that the exact discrete solution should be:

$$u_i^n = Ue^{-\mathrm{I}ck_m t}e^{\mathrm{I}k_m x} = Ue^{\mathrm{I}k_m(x-ct)} \tag{6.3}$$

The amplification factor is defined by $\rho = u_i^{n+1}/u_i^n$, which then has the exact value:

$$\rho_e = e^{a\Delta t} = e^{-\mathrm{I}ck_m\Delta t} = e^{-\mathrm{I}\alpha\beta} \tag{6.4}$$

where $\alpha = \frac{c\Delta t}{\Delta x}$, and $\beta = k_m\Delta x$ with $k_m = \pi m/L$ as defined in chapter 3. When advancing by a single time step, (6.1) is therefore multiplied by the factor $\rho_e = e^{-\mathrm{I}\alpha\beta}$. Since in this case $|\rho_e| = 1$, the magnitude of the Fourier component will not change. However, its phase (angle) should will by the value $-\alpha\beta$ each time step.

The linear diffusion equation

The procedure above can also be performed for the linear diffusion (or heat) equation, $u_t - \nu u_{xx} = 0$. Substituting $u_i^n = Ue^{at}e^{\mathrm{I}k_m x}$ gives $a = -\nu k_m^2$. The exact discrete solution is then:

$$u_i^n = Ue^{-\nu k_m^2 t}e^{\mathrm{I}k_m x} \tag{6.5}$$

which shows that the magnitude of each Fourier component decreases in time, and that the higher wave numbers are damped at a higher rate. The corresponding exact amplification factor is:

$$\rho_e = e^{-\nu k_m^2\Delta t} = e^{-g\beta^2} \tag{6.6}$$

where $g = \frac{\nu\Delta t}{\Delta x^2}$. Once again, the solution is multiplied by ρ_e when advancing by a single time step. Since in this case ρ_e is real, there will be no change in the phase of the wave during a time step, but only a decrease in its magnitude.

6.1.2 Definition of amplitude error

We can now provide a definition of the amplitude error which arises due to an approximate numerical scheme as:

$$er_a = |\rho_e| - |\rho| \tag{6.7}$$

When examining linear advection, the deviation of $|\rho|$ from 1 is thus an indication of amplitude error, while for the linear diffusion equation it is a deviation from the exact decay rate.

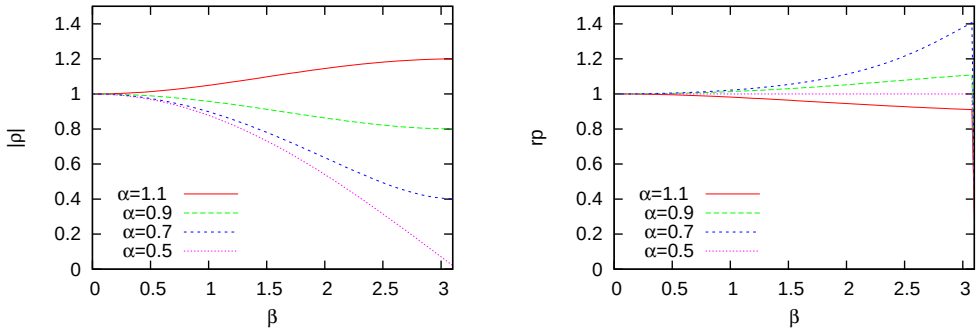


Figure 6.1: Amplification factor and relative phase of the explicit in time, upwind in space scheme

6.1.3 Definition of phase error

Similarly, we can provide a definition for phase error as:

$$er_p = \angle\rho_e - \angle\rho \tag{6.8}$$

where $\angle\rho$ indicates the angle of the complex number ρ, i.e. $\tan^{-1}(\rho_{imaginary}/\rho_{real})$. When dealing with the linear advection equation, it is more common to examine the relative phase:

$$rp = \frac{\angle\rho}{\angle\rho_e} \tag{6.9}$$

In this case, $rp > 1$ indicates the approximated wave component is moving faster than it should, while $rp < 1$ indicates that the approximated wave is lagging behind.

6.1.4 Errors of some example methods

We now examine the phase and amplitude errors obtained by Fourier analysis of some simple-finite difference approximations for the linear advection equation. We start with the explicit Euler in time, first-order upwind in space scheme considered at the end of chapter 3. Plots of $|\rho|$ and rp are shown for this method in figure 6.1. As can be seen from the plot of $|\rho|$, this method is unstable for $\alpha = \frac{c\Delta t}{\Delta x} > 1$, a result we derived previously. For $\alpha = \frac{c\Delta t}{\Delta x} < 1$, the magnitude of the amplification factor is low at higher frequencies, meaning that high-frequency components of the solution are quickly damped. The phase errors are relatively low through the frequency range, however. This behaviour can be seen in the numerical solution shown in figure 6.2. After several time steps the highest-frequency components of the initial square-wave solution have been removed, but the speed of the square wave is reasonably matched.

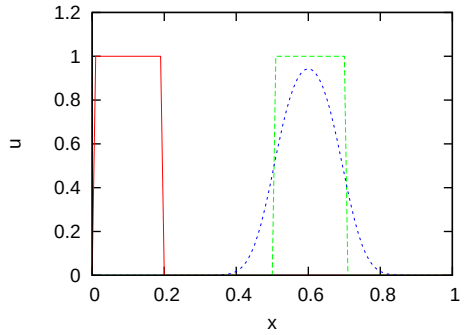


Figure 6.2: Comparison of the exact and computed solution from the explicit in time, upwind in space scheme at $\alpha = \frac{c\Delta t}{\Delta x} = 0.5$

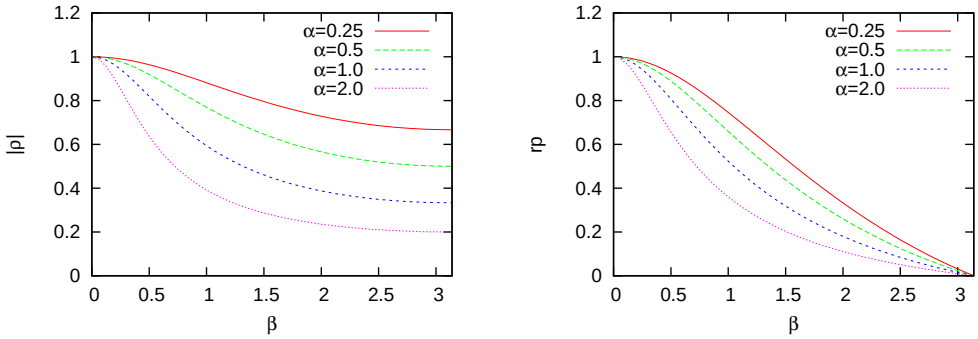


Figure 6.3: Amplification factor and relative phase of the implicit in time, upwind in space scheme

The next example is an implicit Euler in time, upwind in space scheme. $|\rho|$ and rp are shown for this method in figure 6.3. In this case the amplification factor is also low in the mid-frequency range. In addition, the phase error is appreciable (indicated by the large deviations of relative phase from unity). This behaviour can also be seen in the solution shown in figure 6.4. Only the lowest frequencies have survived, and even these lag slightly behind the square-wave solution (note that the frequencies for which phase error is highest have been strongly damped).

Finally we consider an implicit Euler in time, central in space scheme. The corresponding $|\rho|$ and rp plots are given in figure 6.5. In this case the damping occurs only in the medium-frequency range, while the phase error again increases with frequency. This is reflected by the solution in figure 6.6, which shows a "coming apart" of the signal, since the highest modes survive while having significant phase error. This results in high-frequency oscillations.

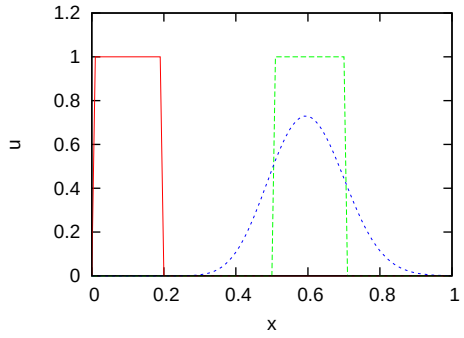


Figure 6.4: Comparison of the exact and computed solution from the implicit in time, upwind in space scheme at $\alpha = \frac{c\Delta t}{\Delta x} = 0.5$

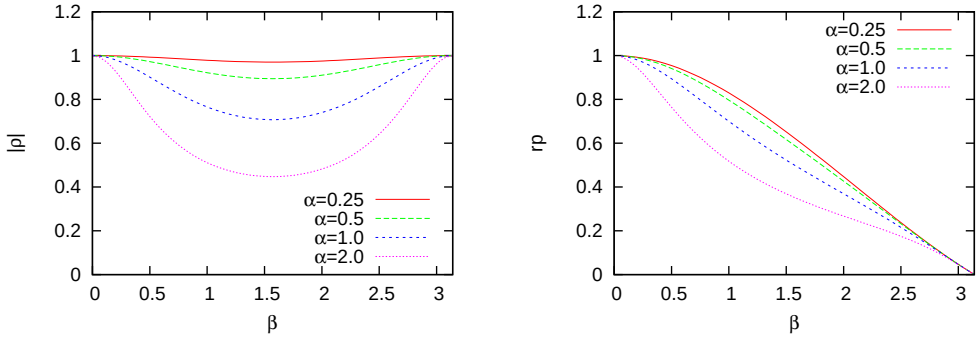


Figure 6.5: Amplification factor and relative phase of the implicit in time, central in space scheme

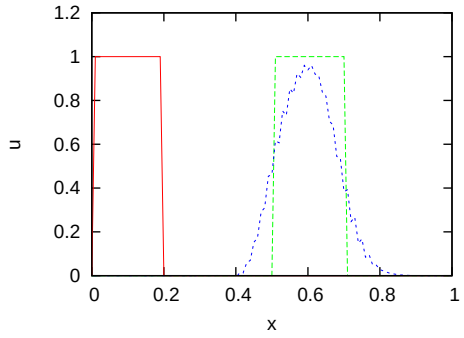


Figure 6.6: Comparison of the exact and computed solution from the implicit in time, central in space scheme at $\alpha = \frac{c\Delta t}{\Delta x} = 0.5$

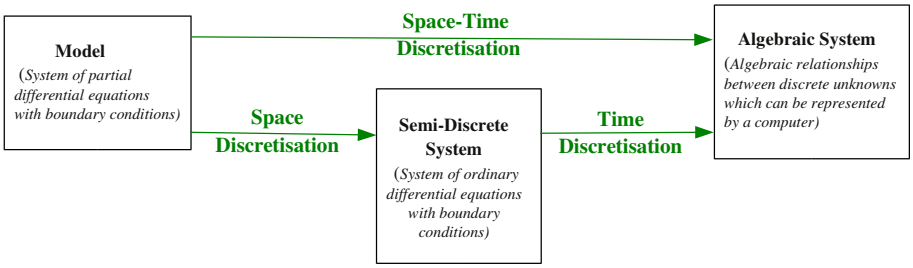


Figure 6.7: The fully and semi-discrete approaches to discretisation

6.2 General analysis of linear methods

6.2.1 From PDE to algebraic system

As discussed in the previous chapter, there are two basic approaches to constructing methods for unsteady problems. These are shown diagrammatically in figure 6.7. Starting from a partial differential equation, one may choose to implement a space-time discretisation, which immediately generates an algebraic system. This is known as the *fully-discrete* approach. Alternatively, one may first leave the time derivatives as additional unknowns, and apply a spatial discretisation only, which leads to a system of ordinary differential equations (ODEs). A time-march method is then applied to the system of ODEs, which again produces a system of algebraic equations. This second *semi-discrete* approach is more popular, as it can be useful to combine the same spatial discretisation with different time-march methods for different problems.

We now provide some notation associated with the semi-discrete and fully-discrete approaches. In the semi-discrete approach, the first step produces a system of ordinary differential equations of the form:

$$\frac{\partial \vec{u}}{\partial t} = [A]\vec{u} - \vec{f} \tag{6.10}$$

For example, a centred spatial discretisation of the linear advection equation with periodic BCs can be written:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \quad \Rightarrow \quad \frac{\partial \vec{u}}{\partial t} = -\frac{c}{2\Delta x}\begin{bmatrix} 0 & 1 & & & & -1 \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & & & & \\ 1 & & & & -1 & 0 \end{bmatrix}\vec{u} \tag{6.11}$$

while for the linear diffusion equation, a centred discretisation with periodic

BCs gives:

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} \quad \Rightarrow \quad \frac{\partial \vec{u}}{\partial t} = \frac{\nu}{\Delta x^2} \begin{bmatrix} -2 & 1 & & & & 1 \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & & & & \\ 1 & & & & 1 & -2 \end{bmatrix} \vec{u} \quad (6.12)$$

The second step in the semi-discrete approach is to choose a time-march procedure and apply it to the system of ODEs. This will result in a final algebraic system of the form:

$$\vec{u}^{n+1} = [C]\vec{u}^n - \vec{q} \quad (6.13)$$

For example, an explicit Euler method applied to either the linear advection or diffusion discretisations given above will result in:

$$\begin{aligned} \vec{u}^{n+1} &= \vec{u}^n + \Delta t \frac{\partial \vec{u}^n}{\partial t} - \vec{q} \\ &= ([I] + \Delta t[A])\vec{u}^n - \vec{q} \end{aligned} \quad (6.14)$$

A fully-discrete approach, on the other hand, bypasses the generation of systems of the form (6.10) and proceeds directly to those of the form (6.13). In the following sections we will analyse the relationships between the semi- and fully-discrete forms, with the purpose of understanding how to choose a time march for a given spatial discretisation.

To minimise complexity, we will assume f and therefore q are fixed in time. Since these then do not affect the transient response of our algorithms we may take them to be zero and omit them from the analysis. We will also limit our examples to those with boundary conditions which are periodic in space. The analysis procedures introduced, however, do not depend on this assumption. If desired, different types of boundary condition operators or spatially-varying interior discretisations could be included using the same procedures, although this might necessitate the use of symbolic manipulation or numerical software to assist with the analysis.

6.2.2 The exact solution of the semi-discrete system

The semi-discrete system obtained after the application of a spatial discretisation technique itself has an exact solution. This can be found by using the eigensystem of the matrix $[A]$, defined by the problem:

$$([A] - \lambda_m[I])\,\vec{x}_m = 0, \qquad m = 1, 2, ...M \quad (6.15)$$

If $[A]$ has a complete set of linearly-independent eigenvectors, then using the matrix of right-hand eigenvectors,$[X]$ defined by:

$$[X] = [\vec{x}_1, \vec{x}_2, ...] \quad (6.16)$$

we can diagonalise the system with the following transformation:

$$[X]^{-1}[A][X] = [\Lambda] \quad = \quad \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & . & \\ & & & \lambda_M \end{bmatrix} \tag{6.17}$$

where $[\Lambda]$ is a diagonal matrix containing the eigenvalues $\lambda_m, m = 1, 2, ...M$. We can use this transformation to find the exact solution of the semi-discrete system by pre-multiplying (6.10) by $[X]^{-1}$ and substituting (6.17) to obtain:

$$[X]^{-1}\frac{\partial \vec{u}}{\partial t} = [X]^{-1}[A][X][X]^{-1}\vec{u} \tag{6.18}$$

where we have assumed that the vector f is zero for clarity [1]. We now define new variables $\vec{w} = [X]^{-1}\vec{u}$. This allows us to re-express the system as:

$$\frac{\partial \vec{w}}{\partial t} = [\Lambda]\vec{w} \tag{6.19}$$

which, since $[\Lambda]$ is diagonal, is a decoupled system of first-order ODEs. This has the exact solution:

$$w_m(t) = C_m e^{\lambda_m t} \qquad m = 1, 2...M \tag{6.20}$$

From the definition of $\vec{w}$, we can then write the solution $\vec{u}$ as:

$$\vec{u}(t) = \sum C_m e^{\lambda_m t}\vec{x}_m \tag{6.21}$$

from which it can be seen that the transient solution u can be thought of as the combined response of the discrete system's eigenvectors, each of which has a temporal behaviour defined by its corresponding λ_m.

The decoupling of the equations is extremely useful conceptually, as we can now analyse our system by considering what happens to individual equations from (6.19) of the form:

$$\frac{dw_m}{dt} = \lambda_m w_m \tag{6.22}$$

A special case: A fixed finite-difference operator with periodic BCs

In the spatial discretisations given in section 6.2.1 for the linear advection and diffusion equations, periodic boundaries were assumed, and the same finite-difference discretisation was used for all nodes in the mesh. The resulting $[A]$

[1] In general f contains specified boundary and source-term values, but when fixed these do not influence the transient behaviours considered here. Note that the discretisation operators used near the boundary are contained within the matrix $[A]$.

matrices, defined by (6.11) and (6.12) are examples of *circulant matrices* i.e., those of the form:

$$\begin{bmatrix} b_o & b_1 & b_2 & b_3 \\ b_3 & b_o & b_1 & b_2 \\ b_2 & b_3 & b_o & b_1 \\ b_1 & b_2 & b_3 & b_o \end{bmatrix}$$

As shown in [20], the eigenvectors of a circulant matrix are always identical and can be expressed as:

$$\vec{x}_m = (x_j)_m = e^{\mathrm{I}(2\pi jm/M)} \qquad \begin{matrix} j = 0, 1, ...M-1 \\ m = 0, 1, ...M-1 \end{matrix} \tag{6.23}$$

where j denotes the node index. While the eigenvalues can be expressed in terms of matrix coefficients as:

$$\lambda_m = \sum_{j=0}^{M-1} b_j e^{\mathrm{I}(2\pi jm/M)} \tag{6.24}$$

Note that (6.24) indicates that our previous decomposition of the numerical solution into Fourier components has a more general interpretation. For a periodic problem with a constant finite-difference operator, the eigenvectors of the semi-discrete system are in fact Fourier modes. Correspondingly, we may think of our numerical discretisation as a coupled system which describes the evolution of the nodal values of u, or a decoupled system which describes the evolution of sets of sines and cosines.

λ_m for example discretisations with periodic BCs

For general cases the eigensystem of the semi-discrete system are often computed using a numerical package. This is usually necessary for discretisations with non-periodic boundary operators, a higher-order finite-element discretisation, or a finite-difference discretisation which uses approximations that change depending on the region of the domain considered. In such cases the eigenvectors are general functions rather than Fourier modes (although they often contain modes which resemble Fourier modes to some extent).

For the special case of a periodic domain with a fixed finite-difference operator, however, we can compute the semi-discrete eigenvalues from (6.24). Eigenvalues for the central linear advection and diffusion discretisations of section 6.2.1 are shown plotted on the complex λ plane in figure 6.8. In general, finite-difference operators corresponding to advective phenomena tend to produce semi-discrete eigenvalues on the imaginary axis, while those corresponding to diffusive phenomena tend to occupy the negative real axis.

Inspection of the expression for the general solution of the semi-discrete system (6.20), reveals that the semi-discrete solution implied by the discretisations

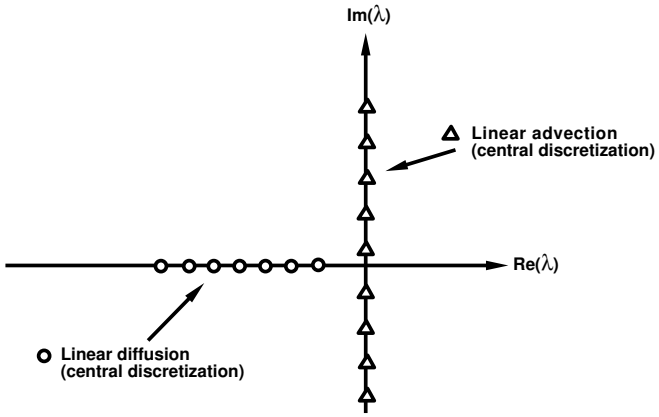


Figure 6.8: Semi-discrete eigensystems for central discretisations

considered above behaves in a manner similar to the solutions of the partial differential equations which they approximate. In the case of the linear diffusion equation, the negative real value of λ results in a transient solution which decays in time, mimicking the exact diffusion process. In contrast, the pure imaginary λ's of the linear advection discretisation will result in the unattenuated propagation of the initial conditions, as occurs in the exact solution.

Correspondence with the exact behaviour of the semi-discrete system does not occur for every consistent discretisation, however. For example, the semi-discrete eigenvalues for a first-order upwind discretisation:

$$\frac{du_i}{dt} = -\frac{c(u_i - u_{i-1})}{\Delta x} \tag{6.25}$$

and a first-order downwind discretisation:

$$\frac{du_i}{dt} = -\frac{c(u_{i+1} - u_i)}{\Delta x} \tag{6.26}$$

appear plotted in figure 6.9. As the eigenvalues of the upwind discretisation include both imaginary and negative real components, its semi-discrete solution will produce an unphysical decay of the initial condition as it is propagated. On the other hand, the semi-discrete solutions produced by the downwind discretisation will amplify in time. Although it may be possible to define a time integration technique to successfully integrate a downwind discretisation in the interior of the domain, in practice this is overly restrictive. Furthermore downwind discretisations are clearly incorrect for boundaries, as they do not respect the characteristic directions of information propagation. In general, spatial discretisations which lead to stable semi-discrete systems of ODEs, for which

$$Re(\lambda_m) < 0 \tag{6.27}$$

are preferred when there is no physical process (e.g. chemistry) which might result in positive growth rates.

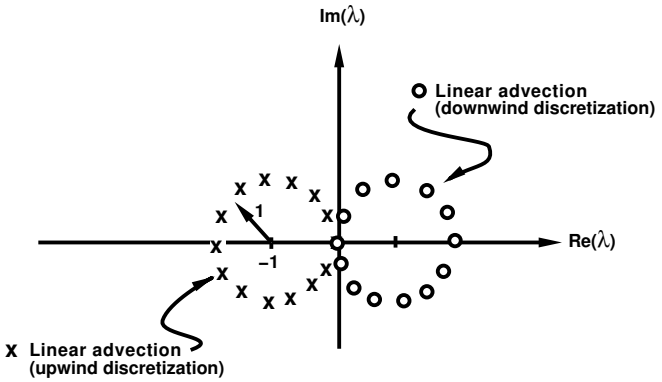


Figure 6.9: Semi-discrete eigenvalues for one-sided discretisations

6.2.3 The exact solution of the fully-discrete system

The fully-discrete system (for $f = 0$) is written:

$$\vec{u}^{n+1} = [C]\vec{u}^n \tag{6.28}$$

Provided that $[C]$ has a complete set of linearly-independent eigenvectors, we can employ the same transformation used for the semi-discrete system to diagonalise the full-discrete system. This results in:

$$w_m^{n+1} = \sigma_m w_m^n \qquad m = 1, 2...M \tag{6.29}$$

where σ_m are the eigenvalues of the fully-discrete matrix, $[C]$, and $\vec{w}^n = [Y]^{-1}\vec{u}^n$ where $[Y]$ is the matrix of right eigenvectors of $[C]$. (6.29) indicates that the σ_m are in fact amplification factors, as they encapsulate the changes in a given spatial mode w_m^n from time step n to $n+1$. Consequently, for the scheme to be stable, i.e. to prevent w_m^n from growing without bound, we require:

$$\mid \sigma_m \mid \leq 1 \qquad m = 1, 2, ...M \tag{6.30}$$

Furthermore , we can define the amplitude and phase errors for an arbitrary $[C]$ as:

$$er_a = |\rho_e| - |\sigma_m| \tag{6.31}$$

and:

$$er_p = \angle\rho_e - \angle\sigma_m \tag{6.32}$$

For general numerical schemes, one can plot phase and amplitude errors ordered by the frequency of their associated modes to obtain diagrams similar to those considered in section 6.1.4. In the case of periodic BCs and a fixed finite-difference discretisation, $[C]$ will again be circulant, the w^n will be Fourier modes, and the σ_m will be identical to the amplification factors predicted by Fourier analysis at the same frequencies.

6.2.4 The relation between λ_m and σ_m

The analysis in the previous sections provides descriptions of the behaviour of the semi and fully discrete systems. These are related to each other by the chosen time march. Consequently, it is the time march which establishes the relation between the λ_m and σ_m.

For example, consider the application of the explicit Euler time march to the system (6.10) with $f = 0$:

$$\begin{aligned} \vec{u}^{n+1} &= ([I] + \Delta t[A])\vec{u}^n \\ &= [C]\vec{u}^n \end{aligned}$$

The eigenvalues of $[C]$, denoted by σ_m are given by:

$$([C] - \sigma_m[I])\,\vec{y}_m = 0, \qquad m = 1, 2, ...M \tag{6.33}$$

or:

$$\left((1-\sigma_m)[X]^{-1}[I][X] + \Delta t[X]^{-1}[A][X]\right)\vec{y}_m = 0, \tag{6.34}$$

from which:

$$\sigma_m = 1 + \lambda_m \Delta t \tag{6.35}$$

This is the same relationship obtained by applying the time march directly to the uncoupled equation (6.22). Applying the explicit Euler time march to (6.22), for example, results in:

$$w_m^{n+1} = w_m^n + \Delta t \lambda_m w_m^n = (1 + \lambda_m \Delta t) w_m^n \tag{6.36}$$

Which confirms (6.35). Similarly, for the implicit Euler method we have:

$$w_m^{n+1} = w_m^n + \Delta t \lambda_m w_m^{n+1} \tag{6.37}$$

In this case the $\lambda - \sigma$ relation is:

$$\sigma_m = \frac{1}{1 - \lambda_m \Delta t} \tag{6.38}$$

Determining the λ_m -σ_m relation using an equation from the decoupled system becomes particularly convenient when the time march is more complex. Consider the second-order accurate MacCormack predictor-corrector scheme:

$$\tilde{u}^{n+1} = u^n + \Delta t \frac{du^n}{dt} \qquad \text{(predictor)} \tag{6.39}$$

$$u^{n+1} = \tfrac{1}{2}\left(u^n + \tilde{u}^{n+1} + \Delta t \frac{d\tilde{u}^{n+1}}{dt}\right) \qquad \text{(corrector)} \tag{6.40}$$

For the decoupled ODE, $\tilde{w}_m^{n+1} = (1 + \lambda_m \Delta t) w_m^n$, so that

$$w_m^{n+1} = \frac{1}{2}\left(w_m^n + (1 + \lambda_m \Delta t) w_m^n + \lambda_m \Delta t (1 + \lambda_m \Delta t) w_m^n\right) \tag{6.41}$$

which gives the relation:

$$\sigma_m = 1 + \lambda_m \Delta t + \frac{(\lambda_m \Delta t)^2}{2} \tag{6.42}$$

At this point it is worthwhile to examine the expressions for σ_m obtained so far. Re-writing the exact solution to the semi- and fully- discrete problems (equations (6.20) and (6.29)), we have, for a given m:

$$w_m(t) = C_m e^{\lambda_m t} \qquad\qquad w_m^{n+1} = \sigma_m w_m^n$$

Comparing the ratio of the semi-discrete solutions $w_m(t + \Delta t)/w_m(t)$ to the ratio of fully discrete solutions w_m^{n+1}/w_m^n reveals that σ_m should approximate $e^{\lambda_m \Delta t}$. We can see that this is the case for each of the time marches considered using the series expansion for the exponential $e^k = 1 + k + k^2/2! + k^3/3! + ...$ (for the implicit Euler case, we also need to consider the binomial theorem $(1 + \epsilon)^{-1} = 1 - \epsilon + \epsilon^2/2 + ...$). In fact, every consistent time march will have a σ_m root which approximates $e^{\lambda_m \Delta t}$ to the order of the time march plus one.

The previous examples involved only two time levels. These methods are therefore referred to as self-starting, as one can march directly from a single initial condition at a given time. In contrast, time marches with employ more than two time levels (larger stencils in time) require information at more than one time level to start. This information is normally provided by other time march methods which are self-starting when the computation is initiated.

A time march which employs more than two levels is the leapfrog method:

$$u^{n+1} \quad = \quad u^{n-1} + 2\Delta t \frac{du^n}{dt} \tag{6.43}$$

Noting that $\sigma_m = w_m^{n+1}/w_m^n$ the $\lambda - \sigma$ relation in this case is:

$$\sigma_m^2 - 2\Delta t \lambda_m \sigma_m - 1 = 0 \tag{6.44}$$

This expression produces two sigma roots, only one of which is an approximation for $e^{\lambda_m \Delta t}$. The additional roots generated by non-self-starting methods are normally referred to as "spurious roots". Spurious roots do not correspond to physical modes, so they do not play a role in accuracy analysis. Non-physical spurious modes must be kept stable, however, so spurious roots are considered in stability analysis. In practice, if a non-self-starting method is started properly using a self-starting method, the initial magnitude of the spurious modes will be very small. They will become even smaller in time provided that the amplitudes of the spurious roots are less than one. Consequently, non-self-starting methods can still provide effective time-marching techniques.

6.2.5 Choosing a time march

As shown in the previous sections, the final behaviour of a method is described by the eigenvalues of the fully-discrete matrix, σ_m. The relationship between the

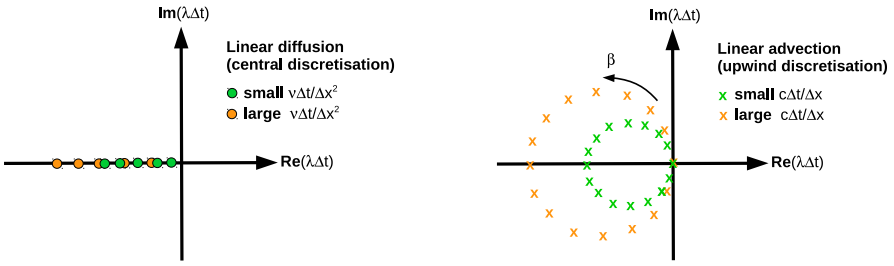


Figure 6.10: Semi-discrete eigenvalues for varying ratios of $\frac{\nu\Delta t}{\Delta x^2}$ and $\frac{c\Delta t}{\Delta x}$

semi-discrete eigenvalues, λ_m which are produced by a given spatial discretisation and the final σ_m is set by the chosen time march. We can therefore choose a time march or modify a spatial discretisation to provide the most favourable σ_m. In this context it is useful to plot the eigenvalues of the spatial discretisation along with contours of the $\lambda - \sigma$ relation on the complex $\lambda\Delta t$ plane. Examples of the former are shown in figure 6.10, which also illustrate how the eigenvalue spectra are scaled via input parameters. For the case of linear diffusion, increasing ν or Δt expands the spectra to the left, while for linear advection increasing c or Δt expands the 2D region occupied by the eigenvalues. Refining the mesh (decreasing Δx) produces similar effects. These variations have consequences for the performance of the discretisation.

In figure 6.11, the contour for $|\sigma| = 1$ for the explicit Euler time march is shown superimposed over different semi-discrete eigenvalue spectra. For the central discretisation of the diffusion equation, sufficiently small values of $\frac{\nu\Delta t}{\Delta x^2}$ will position the λ_m inside inside the circle, and their corresponding σ_m will have a magnitude less than 1. This complete scheme is thus conditionally stable. The same is true for the upwind discretisation of the linear advection equation, which is stable for sufficiently small $\frac{c\Delta t}{\Delta x}$.

On the other hand, a central discretisation of the linear advection equation leads to λ_m which are always outside of the $|\sigma| = 1$ contour. This scheme is therefore unconditionally unstable. However, we can now visualise how a stable scheme can be constructed. If we add artificial dissipation, the λ_m of the central scheme will be pulled to the left (into an elliptical circuit, as shown in figure 6.12). Such a scheme can be integrated with the explicit Euler method provided that $\frac{c\Delta t}{\Delta x}$ is small enough to ensure the ellipse lies within the $|\sigma| = 1$ contour.

In figure 6.13, left, contours for different values of $|\sigma|$ are shown for the explicit Euler time march, along with eigenvalue spectra for an upwind discretisation of the linear advection equation for two CFL numbers. The value of $|\sigma|$ for a given semi-discrete eigenvalue is determined by its position on the $\lambda\Delta t$ plane, where the local value of σ is in turn determined but the $\lambda - \sigma$ relation of the time march. Each of the semi-discrete eigenvalues (λ_m) is associated with an eigenvector with a given wavenumner, β. Plotting the values $|\sigma|$ for

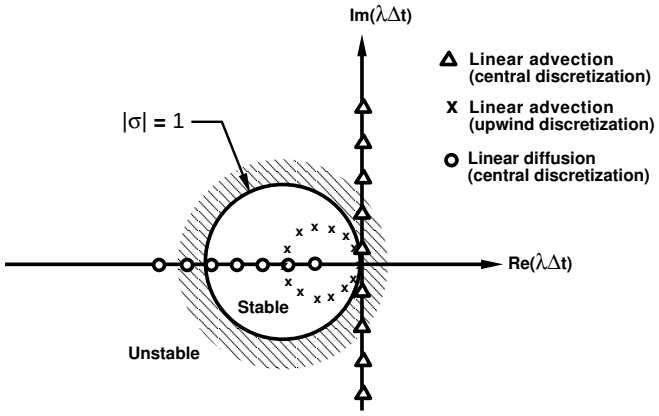


Figure 6.11: Stability region for the explicit Euler time march

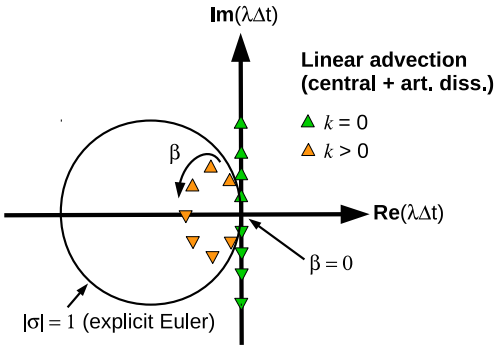


Figure 6.12: Modification of the eigenvalue spectrum of a central discretisation for the linear advection equation using artificial dissipation ($k > 0$).

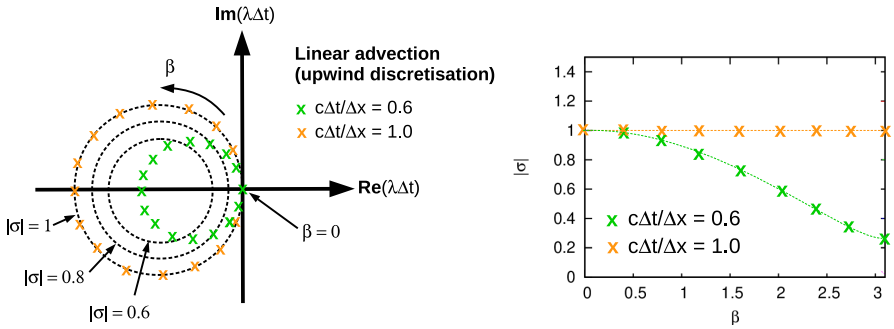


Figure 6.13: Amplification factors for an upwind discretisation of linear advection with an explicit Euler time march with changing $\frac{c\Delta t}{\Delta x}$.

each eigenvalue vs this β results in plots similar to those produced by Fourier analysis (figure 6.13, right).

Tuning a time march with free parameters

Consider an explicit low-storage Runge-Kutta scheme of the form:

$$\begin{aligned}
u_1^{n+1} &= u^n + \alpha^1 \Delta t \left(\frac{\partial u^n}{\partial t} \right) \\
u_2^{n+1} &= u^n + \alpha^2 \Delta t \left(\frac{\partial u_1^{n+1}}{\partial t} \right) \\
&\vdots \\
u_k^{n+1} &= u^n + \alpha^k \Delta t \left(\frac{\partial u_{k-1}^{n+1}}{\partial t} \right) \\
&\vdots \\
u^{n+1} &= u^n + \alpha^K \Delta t \left(\frac{\partial u_{K-1}^{n+1}}{\partial t} \right)
\end{aligned}$$

Where the number of stages K, and the stage coefficients, $\alpha^1 \rightarrow \alpha^K$ can be specified. One approach to choosing a time march is to employ such a general form, and then tune the K and α values to produce the required order of accuracy and phase and amplitude error characteristics.

$|\sigma_m| = 1$ contours for $K = 1$ to $K = 4$ appear plotted in figure 6.14. For larger K the method is seen to be conditionally stable for low-dissipation discretisations of the linear advection equation. Changing the α coefficients can have significant impact on where low amplification factors are encountered. Contours of constant amplification factor are shown for two $K = 4$ schemes

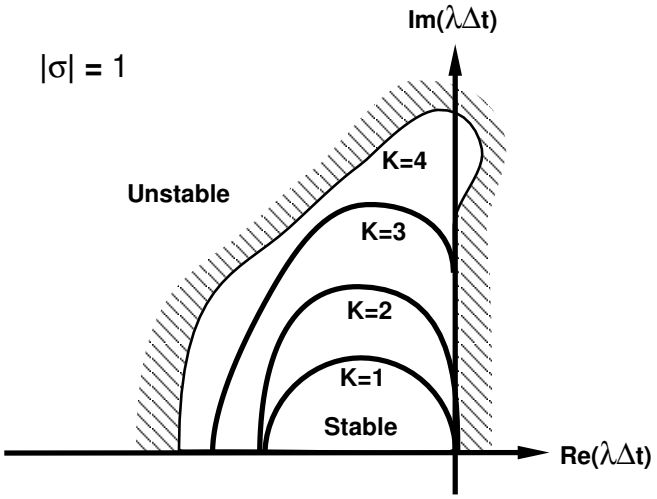


Figure 6.14: Stability regions for the low-storage Runge-Kutta time march

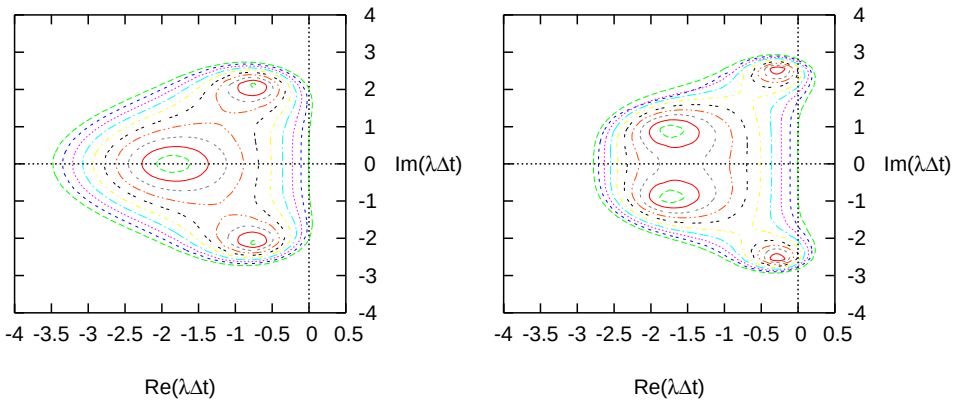


Figure 6.15: $|\sigma|$ = constant contours for four-stage low-storage Runge-Kutta time marches with different α^k. The contour values range from 0.1 to 1.0 with a 0.1 increment.

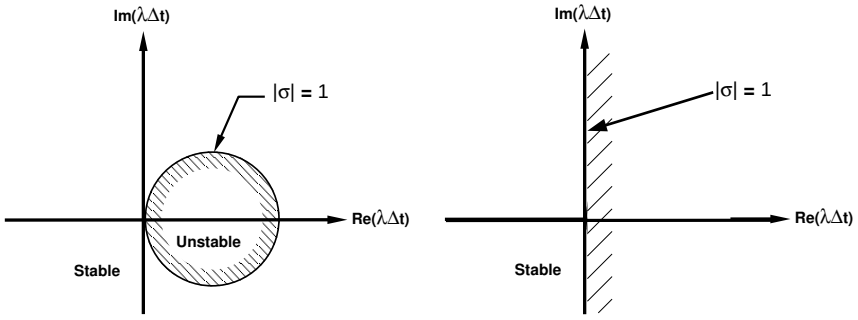


Figure 6.16: Stability regions for the Implicit Euler (left) and trapezoidal (right) time marches

with different alpha coefficients in figure 6.15. A common strategy is to tune the time march so that the highest frequencies, which tend to have high phase errors, are also highly damped.

Implicit versus explicit time marches

The $|\sigma| = 1$ contour for the implicit Euler method is shown in figure 6.16 (left). In this case the σ_m are far from 1 for all the λ_m considered up to now. Although such combinations are stable, their low values of σ_m imply so much amplitude error that they are generally unsuitable for application to advective and diffusive phenomena. There are alternative implicit methods which produce $|\sigma| = 1$ contours which are close to the imaginary axis and provide much better accuracy. A subset of these are stable for all λ_m which have negative real components. An example is the second-order accurate trapezoidal time march, defined by:

$$u_i^{n+1} = u_i^n + \frac{\Delta t}{2}\left(\frac{du_i^{n+1}}{dt} + \frac{du_i^n}{dt}\right) \tag{6.45}$$

for which the stable region is shown in figure 6.16 (right).

Implicit methods are more expensive than explicit methods, so a fair comparison is with explicit methods which are applied for more time steps, but have the same computational work. The choice between them depends on the λ_m which are of most interest to the user. Typically the time step for an explicit method is set by the need to keep the modes with the largest $|\lambda_m|$ within the stable range of the time march. These are normally associated with the highest-frequency components of the solution. Such a time step can be much smaller, however, than that required to adequately resolve a low-frequency transient phenomena which might be of interest. These are typically modes with small $|\lambda_m|$. As a mesh is refined, this mismatch may become much greater. If the time step required for stability becomes so small that an excessive number of time steps will be required to reproduce the desired low-frequency transient, then an

implicit time march will be preferred. An early indication of the necessity for an implicit time march is given by the stiffness of the semi-discrete system, defined by:

$$C_r = \frac{\max(|\lambda_m|)}{\min(|\lambda_m|)} \tag{6.46}$$

Referring to equation (6.20), the C_r thus compares the largest and smallest time scales of the semi-discrete system. C_r values of 10^4 or higher indicate a "stiff" system of ODEs, for which implicit time marches should definitely be considered.

Chapter 7

Iterative solvers

When applied to elliptic problems, or when used with implicit time marches, the finite-difference and finite-element methods introduced in chapters 3 and 6 typically lead to large, sparse systems of linear equations. Naturally these can be solved with generic techniques from linear algebra, but often the work required for such methods scales poorly with the number of unknowns. Since problems with tens of millions of degrees of freedom are not uncommon, several efficient alternatives have been developed. These typically exploit the sparseness of the matrix to produce favourable scaling of work with the number of unknowns.

Solvers for linear algebraic systems can be divided into two categories, direct and iterative. In a direct solver, a process with a fixed number of operations is carried out on the given system of equations to arrive at a solution. In an iterative solver, the solution is initially guessed, and the system of equations is used to repeatedly improve the guess until it comes within a specified tolerance of the exact solution. The first category will be briefly reviewed in the next section. Afterwards we will focus on iterative solvers, as these are the most widely used for large sparse systems.

7.1 A brief overview of direct solvers

As mentioned above, a direct solver follows a certain algebraic recipe for solving a system of equations, which is independent of the exact numerical values of the matrix coefficients. One of the most common examples of a direct solution algorithm is *Cramer's rule*. This is popular as it is easy to remember and apply to 2×2 systems. Unfortunately the operation count of Cramer's rule scales with $(N + 1)!$, where N is the number of unknowns. This makes its use completely impractical once N is increased beyond 10 or so.

A far more common direct solution algorithm is called *Gaussian elimination*. This approach uses substitution to proceed from a full matrix to one which is upper triangular (figure 7.1). Once this is achieved, solutions can be found quickly by initially solving for the last value in the vector using the last equation,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} \qquad \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} \\ & u_{22} & u_{23} & u_{24} & u_{25} \\ & & u_{33} & u_{34} & u_{35} \\ & & & u_{44} & u_{45} \\ & & & & u_{55} \end{bmatrix}$$

Figure 7.1: Initial matrix (left) and final upper triangular form (right)

then substituting that value into the second-to-last equation, and continuing. In the standard version of Gaussian elimination, the procedure is not dependent on the values of the matrix coefficients, and thus does not exploit the sparseness of the matrix. It scales with N^3, but is quite practical for systems of a few hundred equations. In fact, it is often preferred for systems of this size or smaller, since provided that the manipulation of the matrix concerned is not sensitive to round-off error, Gaussian elimination will provide reliable solutions with minimum user intervention.

Gaussian elimination can also be implemented in a way that takes advantage of a particular sparseness pattern. A famous example is the *Thomas algorithm* for tridiagonal systems, which solves problems with matrices of the form:

$$\begin{bmatrix} b_1 & c_1 & & & & \\ a_1 & b_2 & c_2 & & & \\ & a_2 & b_3 & c_3 & & \\ & & a_3 & b_4 & c_4 & \\ & & & a_4 & b_5 & c_5 \\ & & & & a_5 & b_6 \end{bmatrix} \tag{7.1}$$

Such matrices are common in second-order 1D finite-difference approximations, for example when Dirichlet or first-order Neumann boundary conditions are used. The Thomas algorithm for solving tridiagonals is extremely efficient, with a required effort which scales with N.

The development of efficient direct solvers remains an active area of research. The potential advantage lies in their robustness, since they can in principal produce solutions reliably, even when the matrix is ill conditioned. There have been some successes in the development of direct solvers for sparse systems which are more complex than tridiagonal ones. For certain two-dimensional elliptic problems, for example, N scaling with direct solvers has also been achieved. Many finite-difference and finite-element discretisations, however, lead to matrix structures for which no efficient direct solvers exist. As a result, the use of iterative solvers is more common than the use of direct solvers for the solution of large sparse systems.

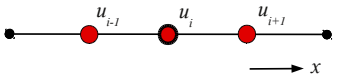


Figure 7.2: Finite-difference stencil for the 1D model problem

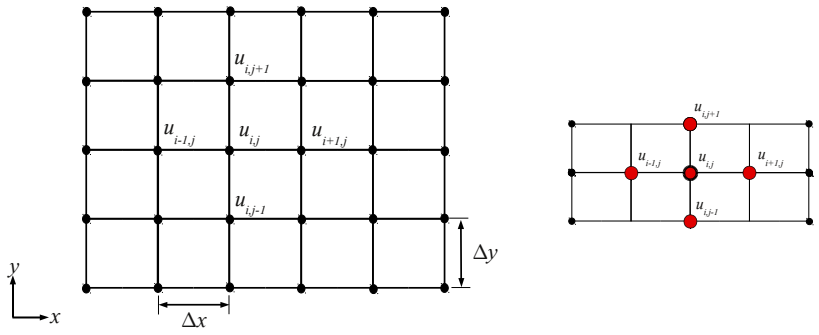


Figure 7.3: Domain and finite-difference stencil for the 2D model problem

7.2 Iterative solvers: model problems

To illustrate the factors which affect the design of iterative solvers, we will focus on the solution of the Laplace equation, $\nabla^2 u = 0$, first on a 1D domain with $h = \Delta x =$ constant, and then on a simple 2D domain with $h = \Delta x = \Delta y =$ constant, as shown in figures 7.2 and 7.3. We will consider central second-order finite-difference discretisations, so that for the 1D problem we have:

$$\left(\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}\right) = 0 \tag{7.2}$$

which requires the solution of $[A]\vec{u} = 0$ where $[A]$ has the following structure:

$$[A] = \frac{1}{h^2}\begin{bmatrix} \ddots & \ddots & \ddots & & & & \\ & 1 & -2 & 1 & & & \\ & & 1 & -2 & 1 & & \\ & & & 1 & -2 & 1 & \\ & & & & \ddots & \ddots & \ddots \end{bmatrix} \tag{7.3}$$

For the 2D problem we will have:

$$\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}\right) + \left(\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}\right) = 0 \tag{7.4}$$

which leads to an $[A]$ of the form:

$$[A] = \frac{1}{h^2}\begin{bmatrix} \ddots & \ddots & & & & \ddots & & & & \\ & 1 & \cdots & 1 & -4 & 1 & \cdots & 1 & & \\ & & 1 & \cdots & 1 & -4 & 1 & \cdots & 1 & & \\ & & & 1 & \cdots & 1 & -4 & 1 & \cdots & 1 & \\ & & & & \ddots & & & & \ddots & & \ddots \end{bmatrix} \tag{7.5}$$

Although the 1D problem could be solved efficiently using the Thomas algorithm, we will consider it for the purposes of demonstrating how iterative methods work in the simplest possible setting. The 2D problem differs considerably from the 1D problem in that it has a large bandwidth, which increases as the mesh is refined.

7.3 Point iteration methods

7.3.1 Jacobi iteration

Iterative methods are based on using the algebraic system to improve on a guess for the solution. The simplest of these is the point Jacobi method, which updates the solution at each point independently based on the previous data. It does so by using each row of the matrix to update the value at one point (normally the point in the row with the largest matrix coefficient). Consider the point i in our 1D model problem. We can correct an initial guess for its value by rewriting (7.2) as an equation for u_i^{n+1}:

$$u_i^{n+1} = \frac{1}{2}(u_{i+1}^n + u_{i-1}^n) \tag{7.6}$$

Where in this case n is the iteration number. Thus the values on the right are all from the previous guess for the solution, while the value on the left is the value for the new guess at i. For the two dimensional problem this would be:

$$u_{i,j}^{n+1} = \frac{1}{4}(u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n) \tag{7.7}$$

In chapter **??** it was pointed out that the Laplace equation acts as a smoothing operator. Equation (7.7) shows that on an equidistant mesh, the discrete version of the Laplace equation is actually trying to make the value at i, j equal to the average of its neighbours.

7.3.2 Gauss-Seidel iteration

When using Jacobi iteration, we base our update entirely on the data of the previous guess. Yet as we proceed through the solution vector, we are obtaining better data. Why not use this immediately in our update procedure? For example, let's assume that we are updating the solution vector in the order

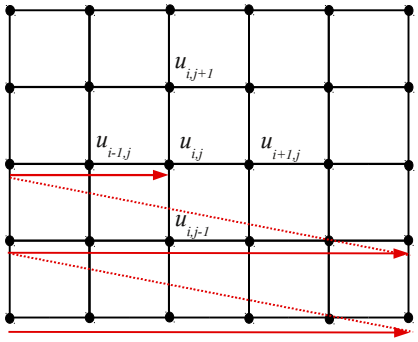


Figure 7.4: Update pattern for point iteration methods

shown in figure 7.4. Then when updating i, j we have new values available for the row below $(j-1)$ and all of the values of the current row which are to the left of i, j. We can thus make a more efficient update using:

$$\text{(1D)} \qquad u_i^{n+1} = \frac{1}{2}(u_{i+1}^n + u_{i-1}^{n+1}) \tag{7.8}$$

$$\text{(2D)} \qquad u_{i,j}^{n+1} = \frac{1}{4}(u_{i+1,j}^n + u_{i-1,j}^{n+1} + u_{i,j+1}^n + u_{i,j-1}^{n+1}) \tag{7.9}$$

7.3.3 Over and under relaxation

The Jacobi and Gauss-Seidel methods can be seen as giving corrections to the local solution of the form $u_i^{n+1} = u_i^n + \Delta u_i^n$ where for the 1D problem:

$$\Delta u_i^n = \frac{1}{2}(u_{i+1}^n + u_{i-1}^n) - u_i^n \tag{7.10}$$

Since Δu_i^n is assumed to move u_i^n in the direction of the exact solution, why not move there a little faster by adding a little more of the correction? On the other hand, if during iteration we keep overshooting the exact solution, why not add a little less? To provide these options we multiply Δu_i^n by the factor ω and re-arrange to obtain:

$$\begin{aligned} u_i^{n+1} &= u_i^n + \omega \Delta u_i^n \\ &= u_i^n + \frac{\omega}{2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \end{aligned} \tag{7.11}$$

If $\omega < 1$, we can expect slower convergence, but less tendency to overshoot. This is called *underrelaxation*. If $\omega > 1$, we can expect faster convergence. This is referred to as *overrelaxation*. The term *successive overrelaxation* (SOR) is used for to the fastest-converging combination, using $\omega > 1$ with the Gauss-Seidel method. The observant should notice that (7.11) is extremely similar to an explicit Euler time march being applied to the equation $u_t = u_{xx}$ where u_{xx} is evaluated with the same finite difference operator used in the model problem.

7.4 Analysis of iterative methods

7.4.1 Fourier analysis

In the previous chapters we used Fourier analysis to determine the behaviour of the numerical solution. We can also use it to determine the behaviour of the error, $\epsilon_i^n = u_i^n - u_i^*$, where u_i^* is the final converged solution to the system of equations. We begin by considering the 1D model problem solved with the point-Jacobi method including relaxation. Substituting the expression $u_i^n = \epsilon_i^n + u_i^*$ into (7.11) gives:

$$\epsilon_i^{n+1} = \epsilon_i^n + \frac{\omega}{2}(\epsilon_{i+1}^n - 2\epsilon_i^n + \epsilon_{i-1}^n) \tag{7.12}$$

since substituting u_i^* into (7.11) gives zero. Thus the evolution of the error is determined by the same equation as that used to update the solution. If we assume that the error is modified each iteration by an amplification factor $\rho = \epsilon_i^{n+1}/\epsilon_i^n$, then the evolution of an inital error distribution with wave number k, $Ee^{\mathrm{I}kx}$, can also be written:

$$\epsilon_i^n = \quad E\rho^n e^{\mathrm{I}kx} \tag{7.13}$$

This can be substituted into the method's error evolution equation to get

$$\begin{aligned} E\rho^{n+1}e^{\mathrm{I}kx} &= E\rho^n e^{\mathrm{I}kx} \\ &+ \frac{\omega}{2}\left(E\rho^n e^{\mathrm{I}k(x+\Delta x)} - 2E\rho^n e^{\mathrm{I}kx} + E\rho^n e^{\mathrm{I}k(x-\Delta x)}\right) \end{aligned} \tag{7.14}$$

Dividing by ϵ_i^n and defining $\beta = k\Delta x$ gives the amplification factor of the point-Jacobi method:

$$\rho = 1 + \frac{\omega}{2}\left(e^{\mathrm{I}\beta} - 2 + e^{-\mathrm{I}\beta}\right) \tag{7.15}$$

$$= 1 + \omega\left(\cos(\beta) - 1\right) \tag{7.16}$$

The amplification factor represents the ratio of the new error over the old error $\epsilon_i^{n+1}/\epsilon_i^n$ which must be less than 1 for convergence. ρ appears plotted for several values of ω in figure 7.5 (left). The interpretation of such a plot is quite different from those we made previously for time-march methods. Before, we wanted ρ to be close to ρ_e. Now we simply want $|\rho|$ to be as small as possible for all wavenumbers, so that the error is reduced to the smallest possible value during each iteration.

Examining figure 7.5, it is clear that combining overrelaxation with the point-Jacobi method is not an option, as then $|\rho| > 1$ and the iteration process will diverge. In fact, a slight amount of underrelaxation might be beneficial for cleaning up the high-wavenumber components of the error. This must be minimised, however, as $|\rho|$ at the lowest wavenumbers increases as ω is decreased.

The situation is much better for the relaxed Gauss-Seidel method:

$$u_i^{n+1} = u_i^n + \frac{\omega}{2}(u_{i+1}^n - 2u_i^n + u_{i-1}^{n+1}) \tag{7.17}$$

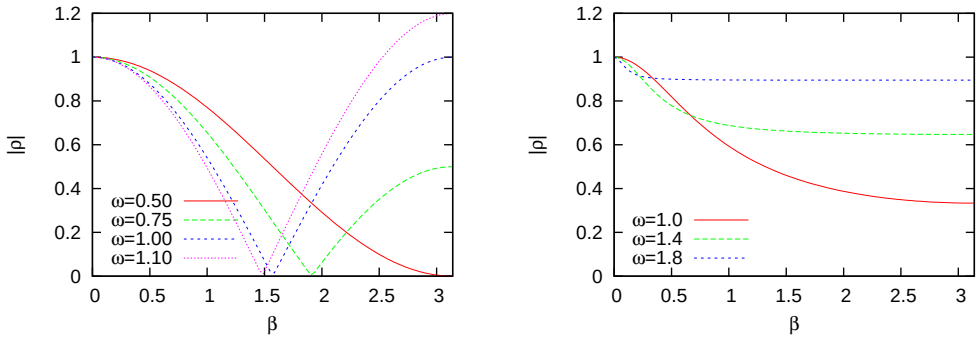


Figure 7.5: Amplification factors for the relaxed point-Jacobi (left) and Gauss-Seidel (right) methods with different values of ω

performing a similar analysis gives an amplification factor of:

$$\rho = \frac{2 + \omega\left(\cos(\beta) + \mathrm{I}\sin(\beta) - 2\right)}{2 - \omega\left(\cos(\beta) - \mathrm{I}\sin(\beta)\right)} \tag{7.18}$$

$|\rho|$ is plotted for the relaxed Gauss-Seidel method in figure 7.5 (right). In this case the basic $\omega = 1$ method has good high-wavenumber error reduction, while the low-wavenumber error reduction can be improved by increasing ω. Once ω exceeds 2, however, the method will become unstable.

7.4.2 Estimating the rate of convergence

The plot of amplification error versus β gives quite detailed information about a particular iterative solution method, but to compare solution methods, it is useful to have a single indicator of performance. This can be defined by noting it is typically the lowest wave numbers that have the lowest rates of error reduction, yet these are often the most important to eliminate. We can therefore define the *rate of convergence* of a method in terms of the the amplification factor for the lowest wave numbers. To make it more intuitive, we will also define the rate of convergence in terms how fast the order of magnitude of the error is decreased.

With every iteration, the magnitude of the error $|\epsilon|$ at a given wavenumber is reduced by the factor ρ. Therefore, after n iterations we can expect:

$$|\epsilon_i^n| = (\rho)^n |\epsilon_i^o| \tag{7.19}$$

where $|\epsilon_i^o|$ is the initial error. We are interested in how many orders of magnitude, m, the error is reduced after n iterations. This is given by:

$$\frac{|\epsilon_i^n|}{|\epsilon_i^o|} = 10^{-m} = (\rho)^n \tag{7.20}$$

so that:

$$-m\ln(10) = n\ln(\rho) \tag{7.21}$$

A method with a good rate of convergence should thus have a high value of m for a given n. We can measure this by defining the rate of convergence as a scaled ratio of m/n:

$$R_c = \frac{m\ln(10)}{n} = \ln\frac{1}{|\rho|} \tag{7.22}$$

As mentioned previously, a good approximation for the solution requires low error at the lowest wavenumbers (these correspond to the largest features in the solution). At the same time, these tend to be associated with the largest $|\rho|$. Therefore ρ in (7.22) used to define the rate of convergence are the lowest non-zero wavenumbers of the discretisation[1].

As an example we will compute the rate of convergence of the 2D Laplace operator (7.4) with unrelaxed Jacobi iteration:

$$u_{i,j}^{n+1} = \tfrac{1}{4}(u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n) \tag{7.23}$$

In this case we assume the error has the form:

$$\epsilon_{i,j}^n = E\rho^n e^{\mathrm{I}(k_x x + k_y y)} \tag{7.24}$$

where E is a constant, k_x and k_y are the wavenumbers of the error in the x and y directions and ρ is the amplification factor associated with the assumed wave numbers. With $\beta_x = k_x\Delta x$ and $\beta_y = k_y\Delta y$ this leads to:

$$\begin{aligned}
\rho &= \frac{1}{4}\left(e^{\mathrm{I}\beta_x} + e^{-\mathrm{I}\beta_x} + e^{\mathrm{I}\beta_y} + e^{-\mathrm{I}\beta_y}\right) && (7.25)\\
&= \frac{1}{2}\left(\cos(\beta_x) + \cos(\beta_y)\right) && (7.26)\\
&\approx 1 - \frac{\beta_x^2 + \beta_y^2}{4} \qquad \text{for small } \ \beta_x, \beta_y && (7.27)
\end{aligned}$$

where for the last line, an approximation for low wave numbers was obtained using the series expansion for cosine:

$$\cos(\beta) = 1 - \frac{\beta^2}{2!} + \frac{\beta^4}{4!} + ... \tag{7.28}$$

We can use two other series expansions:

$$\begin{aligned}
\frac{1}{1-x} &= 1 + x + x^2... && (7.29)\\
\ln(1+x) &= x - \frac{x^2}{2} + \frac{x^3}{3}... && (7.30)
\end{aligned}$$

[1]The zero-wavenumber mode corresponds to a spatially-constant error component, which will be zero if a Dirichlet boundary condition is applied at any point on the boundary

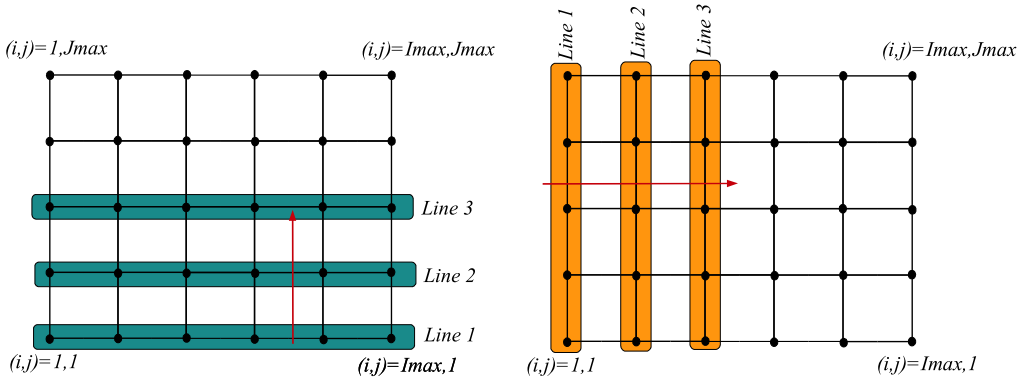


Figure 7.6: Line iteration methods. Updates with j = constant lines (left) and with i = constant lines (right)

to express the rate of convergence as:

$$R_c \approx \frac{\beta_x^2 + \beta_y^2}{4} = \frac{h^2(k_x^2 + k_y^2)}{4} \tag{7.31}$$

This indicates that if we decrease our mesh spacing h by 2, thereby introducing four times the number of unknowns (in 2D), we can expect a drop in the rate of convergence by a factor of 4, meaning 4 times as many iterations will be required to reach a desired tolerance. Since the number of equations which must be updated is also increased by a factor of 4, the work required for Jacobi iteration scales with N^2, where N is the number of unknowns. This scaling is not as good as that achieved by the Thomas algorithm for tridiagonals, but the point Jacobi method can be applied to any matrix and scales considerably better than Gaussian elimination. Finally, the Gauss-Seidel, and especially overrelaxed Gauss-Seidel methods can provide much faster convergence for a given N than the point Jacobi method, although these also scale with N^2.

7.5 Line iteration methods

Since the rate of convergence is normally determined by the speed at which error is removed from the lowest wavenumbers, it makes sense to try to increase the physical length scales we consider when computing an update. This can be done by simultaneously solving for all of the values on a single line of the mesh, as illustrated in figure 7.6, and repeating the process on the following lines. Each line update requires the solution of a system of equations, but for the sample problem considered these will be tridiagonal, allowing us to leverage the excellent performance of the Thomas algorithm.

7.5.1 Line Jacobi

In line Jacobi, one proceeds from line to line as shown in figure 7.6, using data from previous iteration from the lines which are not being solved. Consider the j = constant sweep shown in the left of the figure. For each line, a system of equations is assembled containing only the values on that line as unknowns. For a j = constant line, the solution vector of the line sub-problem will refer to the values of $u_{i=1,Imax}$ for that j. In the Laplace operator the immediate neighbours in i will appear as unknowns whereas the $u_{i,j+1}$ and $u_{i,j-1}$ values will be taken from the solution at the previous iteration. This results in the system of equations defined by:

$$u^{n+1}_{i+1,j} - 4u^{n+1}_{i,j} + u^{n+1}_{i-1,j} = -(u^{n}_{i,j+1} + u^{n}_{i,j-1}) \qquad i = 2, Imax - 1 \quad (7.32)$$

where the left-hand side values contribute to a tridiagonal matrix and those on the right-hand side are known. Additional equations are then added for $i = 1$ and $i = Imax$ to apply the boundary conditions and complete the tridiagonal system.

As an alternative to considering j = constant lines, one can also consider i = constant lines, as shown in figure 7.6 (right). Methods which first update one set of lines and then the other are known as *alternating direction implicit* (ADI) methods.

7.5.2 Line Gauss-Seidel

Line Gauss-Seidel proceeds in exactly the same way as line Jacobi, except that the most recent values for the previous line are used on the right-hand side. As with point Gauss-Seidel, this results in an improvement in the convergence rate. Furthermore, line Gauss-Seidel can be used in combination with overrelaxation to further increase performance. An overrelaxed line would start by solving for the tentative update to the solution (indicated by a long tilde):

$$\widetilde{u^{n+1}_{i+1,j}} - 4\widetilde{u^{n+1}_{i,j}} + \widetilde{u^{n+1}_{i-1,j}} = -(u^{n}_{i,j+1} + u^{n+1}_{i,j-1}) \qquad i = 2, Imax - 1 \quad (7.33)$$

and then apply the overrelaxed correction to obtain the actual update:

$$u^{n+1}_{i,j} = u^{n}_{i,j} + \omega \left(\widetilde{u^{n+1}_{i,j}} - u^{n}_{i,j} \right) \qquad i = 2, Imax - 1 \quad (7.34)$$

This can be combined with an ADI approach for maximum effectiveness.

7.6 Modern iterative methods

Point and line iterative methods have seen wide use, but there has been considerable progress over the last few years developing alternative iterative techniques which have improved scaling of work required versus the number of unknowns.

One very effective approach, known as *multigrid*, considers the problem simultaneously on meshes with different levels of refinement. This improves the convergence of low-wavenumber features in particular, since when considered on a coarse mesh, such features become relatively high in wavenumber. Multigrid has been applied to a wide range of problems, and more recently algebraic versions have emerged. The interested reader is referred to [7, 30].

Another important line of development of iterative methods is Krylov subspace methods, which attempt to find approximate solutions which minimise the residual of the system [26]. Popular variants include the *conjugate gradient* and *generalised minimum residual* (GMRES) method, both of which can be extremely effective for certain classes of large sparse systems. Students interested in this broad and important area are encouraged to follow the master-level courses in scientific computing offered by the department of Applied Mathematics.

Bibliography

[1] *AIAA Guide for the Verification and Validation of Computational Fluid Dynamics Simulations*. Number G-077-1998e. AIAA Standards Series, 1998.

[2] T. J. Barth. Numerical methods for gasdynamic systems on unstructured meshes. In Kroner D., M. Ohlberger, and Rohde. C., editors, *An introduction to recent developments in theory and numerics for conservation laws*, Lecture Notes in Computational Science and Engineering, pages 195–285. Springer-Verlag, 1998.

[3] E. B. Becker, G. F. Carey, and J. T. Oden. *Finite Elements, An Introduction*. Prentice-Hall, 1981.

[4] Pavel Bochev and James Hyman. Principles of mimetic discretizations of differential operators. In Douglas N. Arnold, Pavel B. Bochev, Richard B. Lehoucq, Roy A. Nicolaides, Mikhail Shashkov, Douglas N. Arnold, and Fadil Santosa, editors, *Compatible Spatial Discretizations*, volume 142 of *The IMA Volumes in Mathematics and its Applications*, pages 89–119. Springer New York, 2006.

[5] S. C. Brenner and R. Scott. *The Matematical Theory of Finite Element Methods*. Springer, 2002.

[6] Franco Brezzi, Konstantin Lipnikov, and Valeria Simoncini. A family of mimetic finite difference methods on polygonal and polyhedral meshes. *Mathematical Models and Methods in Applied Sciences*, 15(10):1533–1552, 2005.

[7] W. L. Briggs. *A multigrid tutorial*. SIAM, 2000.

[8] Alexander H.-D. Cheng and Daisy T. Cheng. Heritage and early history of the boundary element method. *Engineering Analysis with Boundary Elements*, 29(3):268 – 302, 2005.

[9] J.A. Cottrell, T.R.J. Hughes, and Y. Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons, 2009.

[10] R. Dwight. *AE2212 Part 1: Applied Numerical Analysis: Lecture Notes*. Delft University of Technology, 2012.

[11] K. Fidkowski and D. Darmofal. Review of output-based error estimation and mesh adaptation in computational fluid dynamics. *AIAA Journal*, 49(4):674–694, 2011.

[12] C. F. Gerald and P. O. Wheatley. *Applied Numerical Analysis - Seventh Edition*. Pearson Addison Wesley, 2004.

[13] Thomas J. R. Hughes, Gerald Engel, Luca Mazzei, and Mats G. Larson. The continuous galerkin method is locally conservative. *Journal of Computational Physics*, 163(2):467 – 488, 2000.

[14] Thomas J.R. Hughes and Garth N. Wells. Conservation properties for the galerkin and stabilised forms of the advection-diffusion and incompressible navier-stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 194(9-11):1141 – 1159, 2005.

[15] TRJ Hughes. *The finite element method: linear static and dynamic finite element analysis*. Dover Publications, 2000.

[16] M. Y. Hussaini, B. van Leer, and J. van Rosendale. *Upwind and High-Resolution Schemes*. Springer-Verlag, 1997.

[17] C. Johnson. *Numerical solution of partial differential equations by the finite element method*. Cambridge University Press, 1987.

[18] G. Karniadakis and S. Sherwin. *Spectral/hp element methods for CFD*. Oxford University Press, 1999.

[19] A. Loeven. *Efficient uncertainty quantification in computational fluid dynamics*. PhD thesis, Delft University of Technology, 2010. ISBN 978-90-8891-171-2.

[20] H. Lomax, T. H. Pulliam, and D. W. Zingg. *Fundamentals of Computational Fluid Dynamics*. Springer-Verlag, 2001.

[21] L. G. Margolin and M. Shashkov. Finite volume methods and the equations of finite scale: A mimetic approach. *International Journal for Numerical Methods in Fluids*, 56(8):991–1002, 2008.

[22] Jan Nordstrom and Mark H. Carpenter. High-order finite difference methods, multidimensional linear problems, and curvilinear coordinates. *Journal of Computational Physics*, 173(1):149 – 174, 2001.

[23] W. L. Oberkampf, T. G. Trucano, and C. Hirsch. Verification, validation, and predictive capability in computational engineering and physics. *Applied Mechanics Reviews*, 57(5):345–384, 2004.

[24] P. J. Roache. Code verification by the method of manufactured solutions. *Journal of Fluids Engineering*, 124(1):4–10, 2002.

[25] C. J. Roy. Review of code and solution verification procedures for computational simulation. *Journal of Computational Physics*, 205(1):131–156, 2005.

[26] Y. Saad. *Iterative methods for sparse linear systems (2nd edition)*. Springer (and online), 2003.

[27] R. Sevilla, S. Fernández-Méndez, and A. Huerta. NURBS-enhanced finite element method (NEFEM). *International Journal for Numerical Methods in Engineering*, 76(1):56–83, 2008.

[28] R. Sevilla, S. Fernández-Méndez, and A. Huerta. Comparison of high-order curved finite elements. *International Journal for Numerical Methods in Engineering*, in print, 2011.

[29] J. F. Thompson, Z. U. A. Warsi, and C. W. Mastin. *Numerical Grid Generation: Foundations and Applications*. North-Holland, 1985.

[30] P. Wesseling. *An Introduction to Multigrid Methods*. R.T. Edwards, 2004.

Appendix A

Notation

A.1 Analysis

$\forall$	for all
$\in$	in
$\exists$	there exists
$A \subset M$	A is a subset of M.
$\|\mathbf{x}\|$	The norm of $\mathbf{x}$
$\|\mathbf{x}\|$	The Euclidean norm of $\mathbf{x}$, defined as $\sqrt{\mathbf{xx}}$.

A.2 Symbols

w	weighting function $\equiv$ test function
ϕ	basis function $\equiv$ interpolation function

A.3 Abbreviations

ODE	Ordinary differential equation
PDE	Partial differential equation
MWR	Method of weighted residuals
FEM	Finite element method

A.4 Function Spaces

$C^n(\Omega)$	Space of functions which have continuous derivatives up to order n in domain Ω
$L_2(\Omega)$	Space of functions which are square-integrable in domain Ω
$H^n(\Omega)$	Space of functions which are square integrable, and which have derivatives up to order n which are square integrable, on domain Ω
$H_0^n(\Omega)$	Subspace of $H^n(\Omega)$ which only includes functions whose values are zero on the boundary of Ω

Appendix B

Finite-volume methods

Both finite-difference and finite-element methods have been widely applied to fluid dynamic problems. Finite-difference methods are simple to design and analyse, but their application to complex geometries can be awkward, particularly when considering higher orders of accuracy on irregular or unstructured meshes. Finite-element methods deal easily with such issues, but can require more computational effort due to the cost of integration. For fluid dynamics problems there is a third approach, however, which retains some of the flexibility of finite-element methods but is relatively simple to implement, and relatively fast in execution. This approach, known as the finite-volume method, is currently the most popular for engineering fluid dynamics.

The finite-volume method is a natural approach to the discretisation of *conservation laws*, such as the mass, momentum and energy equations encountered in fluid dynamics. Finite-volume methods mimic these laws by *exactly*[1] conserving mass, momentum or energy, no matter what the size of the mesh. This property is known as *discrete conservation*, and has been shown to enhance both stability and accuracy, particularly when computing the dynamics of discontinuities (such as shock waves) on coarse meshes. When formulated in conservative variables, finite-element methods also have the discrete conservation property [13, 14], but finite-difference methods require special treatment to achieve it [22].

Although finite-volume methods were initially developed by physical analogy (which partly accounts for their popularity), their close connection to discrete conservation links them to a larger class of methods designed to exactly reproduce key mathematical properties of their underlying physical models. These are known as *mimetic methods* and are currently receiving considerable attention from the numerical analysis community [4, 6, 21].

[1] To machine precision

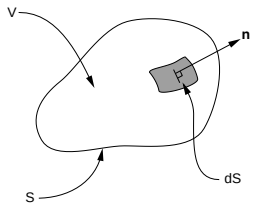


Figure B.1: A control volume of finite size V

B.1 Finite-volume methods for fluid dynamics

The starting point for the derivation of finite-volume methods is the integral form of the conservation laws. These can be derived by considering the conservation of mass, momentum and energy within a volume of finite size, such as that shown in figure B.1. For such a volume this results in:

$$\frac{\partial}{\partial t}\int_V U \; dV \quad + \quad \int_S \vec{F}\cdot\vec{n} \; dS = 0 \tag{B.1}$$

where U is known as the *state vector* defined by:

$$U = \begin{bmatrix} \rho \\ \rho\vec{u} \\ \rho E \end{bmatrix} \tag{B.2}$$

which contains the unknown conservative variables density (ρ), momentum ($\rho\vec{u}$), and total specific energy (ρE). $\vec{F}$ is the vector containing the fluxes of these quantities through the boundaries of the volume. Basically, equation (B.1) can be read as:

The time rate of change of a quantity within the control volume is equal and opposite to the flux of the quantity out of the control volume

For the equations of fluid flow, in the mass conservation equation $\vec{F}$ is simply the conserved variable ρ times the local velocity vector. In the momentum and energy equations, $\vec{F}$ also includes pressure and viscous terms.

A finite-volume method is obtained by applying (B.1) directly to the finite-sized cells which make up the computational mesh. Referring to figure B.2, if we define our unknowns, $\tilde{U}_{ij}$,as the volume average of U in each cell:

$$\tilde{U}_{ij} = \frac{1}{V_{ij}}\int_{V_{ij}} U \; dV_{ij} \tag{B.3}$$

we can apply (B.1) to each cell to obtain the following set of semi-discretised

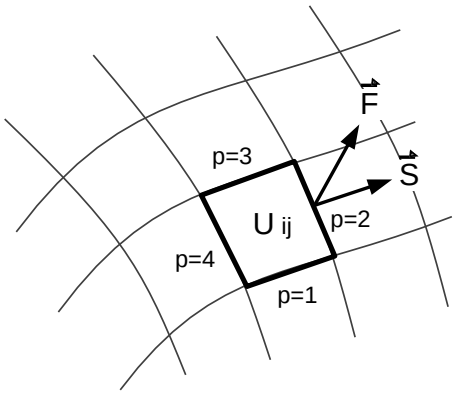


Figure B.2: Cell-centred finite volume discretisation

equations:

$$\frac{\partial}{\partial t}\tilde{U}_{ij}V_{ij} + \sum_{p=1}^{P} \vec{F}_p \cdot \vec{S}_p = 0 \tag{B.4}$$

where P is the number of faces which define the cell, (four in figure B.2), and $\vec{S}_p$ is the area vector of a face:

$$\vec{S}_p = \vec{n}_p \cdot \mid S_p \mid \tag{B.5}$$

where $\vec{n}$ is the face normal vector and $\mid S_p \mid$ is the face area. In general $\vec{F}_p$ can be a function of several of the $\tilde{U}_{ij}$ values in the vicinity of the face p. If we consider a face perpendicular to the i-direction of a structured mesh, for example (figure B.3), a typical form for $\vec{F}_p$ would be:

$$\vec{F}_p = \vec{F}_p(\tilde{U}_{i-m,j}, ...\tilde{U}_{i,j}, ...\tilde{U}_{i+m,j}) \tag{B.6}$$

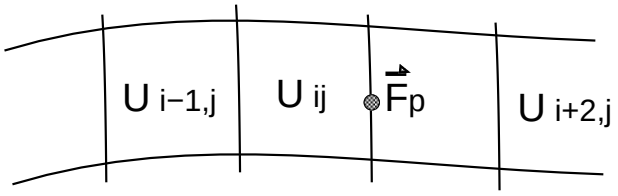


Figure B.3: Flux evaluation on a structured mesh

Different finite-volume methods can be categorised in terms of the choice of control volume and the method used for flux evaluation, as described in the following sections.

B.1.1 Choice of control volume

In order to maintain the discrete conservation property, the choice of control volume for the evaluation of $\tilde{U}_{ij}$ is subject to the following conditions:

- Each volume must be closed ($\int \vec{n} \cdot dS = 0$)
- The value of flux, $\vec{F}_p$, evaluated at any surface must be independent of the volume under consideration
- The volumes must cover the entire domain
- Any overlapping face must be common to two control volumes

There remain, however, a number of ways to define a control volume. In addition to the *cell-centred* discretisation of figure B.2, it is possible to define *vertex-centred* methods, such as that shown in figure B.4, where cells surrounding the vertices are used to define control volumes. Volume definitions which do not follow the grid lines are also possible (figure B.5). Finally, it is also easy to define control volumes on unstructured meshes. These may be either cell-centred or vertex centred, as shown in figure B.6.

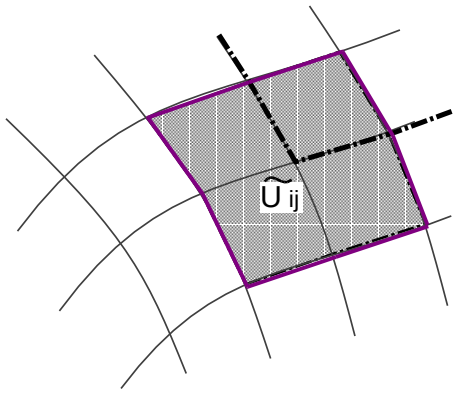


Figure B.4: Vertex-centred finite-volume discretisation

B.1.2 Choice of flux evaluation

To complete a finite-volume discretisation we need to express the surface fluxes $\vec{F}$ as a function of the unknown volume averaged values, $\tilde{U}_{ij}$. The simplest approach is to use the average the $\tilde{U}_{ij}$ values on either side of the face to compute $\vec{F}$. This leads to a scheme similar to central differencing, and typically requires the addition of artificial dissipation to the flux definition to prevent oscillations in the presence of advection. Alternatively, the fluxes can be based on values of $\tilde{U}_{ij}$ taken from one side of the face, corresponding to an upwind

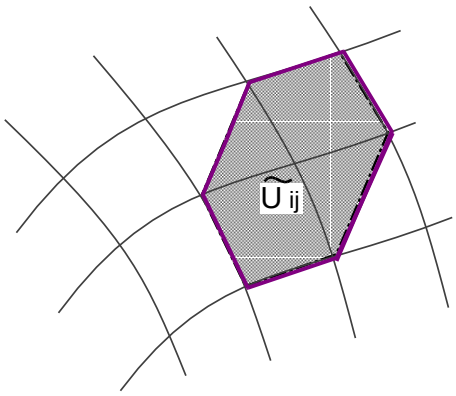


Figure B.5: Mc Donald finite-volume discretisation

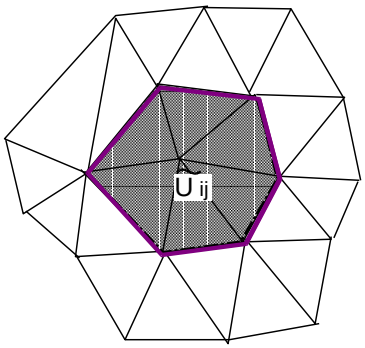


Figure B.6: Unstructured vertex-centred finite-volume discretisation

scheme. This is a natural approach when the propagation direction of solution information clear. Upwinding is effective if only first-order accuracy is desired, but care must be taken in the design of higher-order schemes, particularly on unstructured meshes. This topic has received considerable attention from those in the field of method design [16]. In any case, provided that the definition of flux on given face is unique, the finite-volume method will retain the discrete conservation property, for any flux evaluation method.

Appendix C

Common notation for spectral and finite-element methods

Requirements such as (5.12) arise frequently in the study of spectral and finite-element methods, so it makes sense to use compact notation to express them. In this appendix, definitions are provided for the most common notation encountered in the literature.

C.1 $C^n(\Omega)$

We say that a function is in $C^n(\Omega)$ when it has continuous derivatives up to order n in domain Ω. The functions in figure 5.1(b) are therefore in $C^0(\Omega)$. Often we omit the reference to Ω and just say the functions are in C^0, or are C^0 continuous.

C.2 $L^2(\Omega)$

A function $f(x)$ is in $L^2(\Omega)$ if it is square integrable in Ω:

$$\int_\Omega f^2 d\Omega < \infty \tag{C.1}$$

C.3 $H^n(\Omega)$

A function $f(x)$ is in $H^n(\Omega)$ if it and its derivatives up to order n are square integrable in Ω. For example, the following would be true for a function in H^1:

$$\int_\Omega \left[f^2 + f_x^2\right] d\Omega < \infty \tag{C.2}$$

$H^n(\Omega)$ defines the *Sobolev space* of functions, which is widely used in the analysis of partial differential equations.

C.4 $\{\cdot|....\}$

This notation is used to describe a particular set of functions, also known as a *function space*. The braces indicate that we are providing a set description. To the left of the | we give the notation for a member of the set, and to the right of the | we give the properties of the members. For example, we can define the set of suitable weighting functions for the weak form (5.7) as:

$$\mathcal{W} = \{w \mid w \in H^1,\ w(0) = 0\} \tag{C.3}$$

Since $\mathcal{W}$ describes a Sobolev space, but with some additional conditions, $\mathcal{W}$ may be referred to as a *Sobolev subspace*.

C.5 $(\cdot, \cdot)_\Omega$

This is a short form for symmetric, bilinear forms. Basically:

$$(a, b)_\Omega \equiv \int_\Omega (a\ b)\ d\Omega \tag{C.4}$$

By symmetric, we mean:

$$(a, b) = (b, a) \tag{C.5}$$

whereas bilinear refers to the following type of linearity for both a and b:

$$(c_1 a_1 + c_1 a_2, b) = c_1(a_1, b) + c_2(a_2, b) \tag{C.6}$$

C.6 A precise statement of the sample problem

Using the notation introduced in the previous section, we can now give a precise statement of the weak formulation (5.7) for the sample problem:

For $\mathcal{W} = \{w \mid w \in H^1,\ w(0) = 0\}$ and $\mathcal{U} = \{u \mid u \in H^1,\ u(0) = g\}$, find $u \in \mathcal{U}$ such that for all $w \in \mathcal{W}$:

$$w(1)q - (w_x, u_x)_\Omega = (w, f)_\Omega \tag{C.7}$$

In the above, we have substituted $w(0) = 0$ and $u_x(1) = q$, corresponding to the Dirichlet and Neumann boundary conditions.

Appendix D

Computations with arbitrary finite-element geometries

D.1 Isoparametric approach

Computing on elements with arbitrary geometries can be done using an *isoparametric approach*, where arbitrary elements in physical space are transformed into a simple master element of equivalent topology (figure D.1). Computations can then be performed in the coordinates of the master element. To take the physical geometry of an element into account, a generalised transformation between the master coordinates (ξ, η) and the physical coordinates (x, y) can be used.

When working with a master element, multidimensional basis functions can easily be defined by using *tensor products* of one-dimensional basis functions.

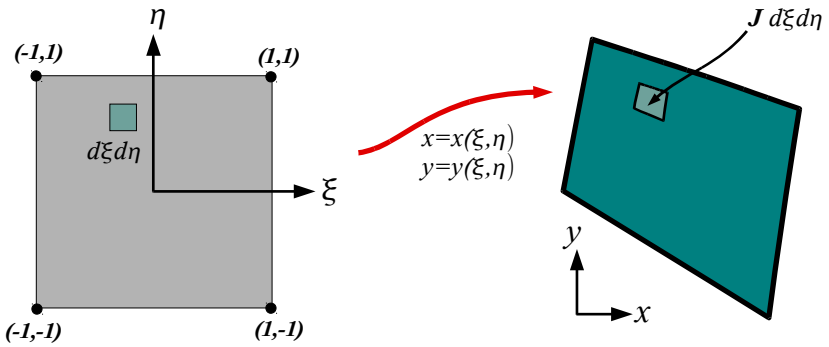


Figure D.1: A master element (left) and a physical element (right)

For example, consider the master element shown in the left half of figure D.1. A two-dimensional piecewise linear basis can be defined by using the following functions:

$$\phi_1(\xi,\eta) = \frac{1}{4}(1-\xi)(1-\eta), \quad \phi_2(\xi,\eta) = \tfrac{1}{4}(1+\xi)(1-\eta), \tag{D.1}$$
$$\phi_3(\xi,\eta) = \frac{1}{4}(1+\xi)(1+\eta), \quad \phi_4(\xi,\eta) = \tfrac{1}{4}(1-\xi)(1+\eta)$$

These are essentially products of the 1D piecewise linear basis we considered previously, expressed in ξ and η coordinates.

In order to compute element matrices and vectors in master coordinates, we need to relate the operations of differentiation and integration to equivalent operations in physical coordinates. This can be done by realising that the physical coordinates can be considered to be known functions when expressed in (ξ,η) coordinates, that is $x = x(\xi,\eta)$ and $y = y(\xi,\eta)$. The differential lengths in both coordinate systems can then be related to each other by the chain rule:

$$\begin{aligned} dx &= x_\xi\, d\xi + x_\eta\, d\eta \\ dy &= y_\xi\, d\xi + y_\eta\, d\eta \end{aligned} \tag{D.2}$$

where quantities such as $x_\xi = \frac{\partial x}{\partial \xi}, y_\eta = \frac{\partial y}{\partial \eta}$... are known as metrics of the transformation. (D.2) can be written in matrix format as:

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{bmatrix} \begin{bmatrix} d\xi \\ d\eta \end{bmatrix} \tag{D.3}$$

We could have also started by considering differential lengths on the master element to obtain:

$$\begin{bmatrix} d\xi \\ d\eta \end{bmatrix} = \begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} \tag{D.4}$$

By comparison, the second matrix of metrics is equivalent to the inverse of the first. This leads to:

$$\begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix} = J^{-1} \begin{bmatrix} y_\eta & -x_\eta \\ -y_\xi & x_\xi \end{bmatrix} \tag{D.5}$$

where $J = (x_\xi y_\eta - y_\xi x_\eta)$ is called the Jacobian of the transformation.

J represents the ratio of areas measured in the two coordinate systems. Specifically, an area of integration $d\xi\ d\eta$ on the master element corresponds to an area $J\ d\xi\ d\eta$ on the physical element. Correspondingly, integrals on the physical element (pe) are related to those on the master element (me) by:

$$\int_{\Omega_{pe}} f(x,y)\, dx\, dy = \int_{\Omega_{me}} f(\xi,\eta)\, J\, d\xi\, d\eta \tag{D.6}$$

In summary, (D.2), (D.5) and (D.6) can be used to relate the operations of differentiation and integration on the master element to those on the physical

element. Before using these relations, however, we need to determine the transformation metrics x_ξ, x_η, y_ξ and y_η. To do so we need explicit definitions for $x = x(\xi, \eta)$ and $y = y(\xi, \eta)$. A common approach is to construct a functional definitions using the solution basis:

$$x(\xi, \eta) = \sum_{i=1}^{N} b_i \, \phi_i(\xi, \eta)$$
$$y(\xi, \eta) = \sum_{i=1}^{N} c_i \, \phi_i(\xi, \eta) \tag{D.7}$$

although any appropriate basis can be used. The amplitudes of the coordinate basis functions, b_i and c_i, can be determined by requiring the interpolation to reproduce the physical coordinates of the element. If the physical elements have straight-line edges, and the basis (D.2) is used, the coefficients are simply given by $b_i = x_i$ and $c_i = y_i$, where x_i, y_i are the physical coordinates of the vertex where $\phi_i(\xi, \eta) = 1$ (note that for (D.2) all other ϕ are zero at this point).

Bringing everything together, the procedure for evaluating integrals and derivatives on arbitrary elements is:

1. Define $x(\xi, \eta)$ and $y(\xi, \eta)$ using physical element coordinates and (D.7)
2. Take derivatives of (D.7) to find $x_\xi, x_\eta, y_\xi, y_\eta$
3. Compute $J = (x_\xi y_\eta - y_\xi x_\eta)$
4. Use (D.5) to determine $\xi_x, \eta_x, \xi_y, \eta_y$
5. Evaluate the required solution derivatives by taking gradients in (ξ, η), then applying the chain rule (D.2)
6. Evaluate contributions to integrals using (D.6)

When integration is performed by quadrature, steps 2-6 above are repeated for each quadrature point. Note that all of the above operations are performed on the master element. The isoparametric approach is widely used since it is convenient from an implementation point of view, but complex transformations due to curved boundaries change the polynomial order of the interpolation in physical space. Maintaining the correct order of convergence (p in equation (5.33) then requires that the curvature of the elements be not too large. In practice this means that a minimum level of refinement in h is required before the optimal rate of convergence is obtained for a given p.

D.2 Physical coordinate approach

Basis functions can also be created and evaluated directly in physical coordinates. This has the advantage that the order of polynomial interpolation is preserved when elements with curved boundaries are used. To illustrate the

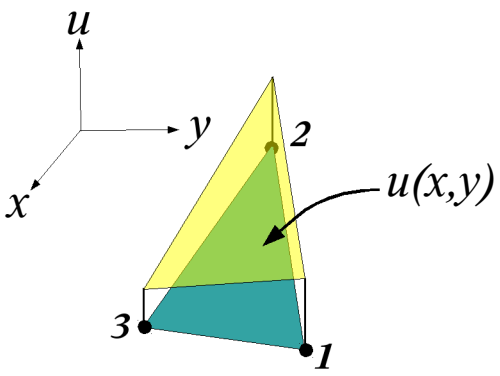


Figure D.2: Linear solution over a two-dimensional triangle element

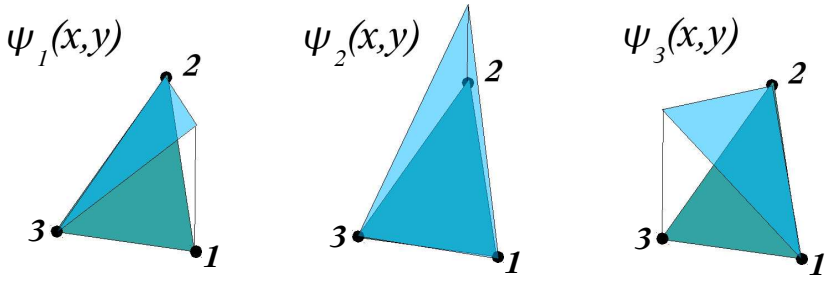


Figure D.3: Local functions for interpolating a linear solution within an element

procedure we will consider the construction of a piecewise linear basis for triangular elements with straight edges. As shown in figure D.2, a linear solution over a two-dimensional element e can be represented by the function:

$$u_e(x, y) = a + bx + cy \tag{D.8}$$

For convenience, we would like to express this as a combination of three "shape" functions, each of which has the value of the solution at one node and is zero at the others, as shown in figure D.3. This requirement can be expressed as:

$$u_1 = a + b\, x_1 + c\, y_1 \tag{D.9}$$
$$u_2 = a + b\, x_2 + c\, y_2 \tag{D.10}$$
$$u_3 = a + b\, x_3 + c\, y_3 \tag{D.11}$$

Solving this system for a, b and c leads to:

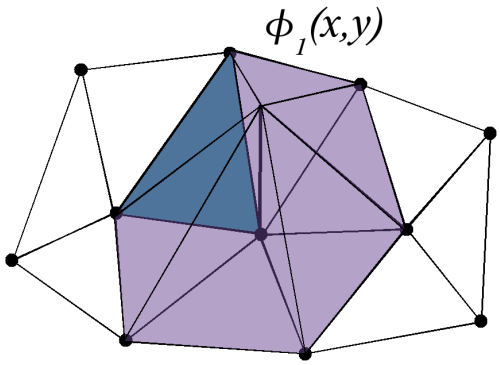


Figure D.4: Definition of a nodal basis function ϕ_i by aggregating shape functions from each surrounding element

$$a = \frac{1}{2A_e}\left[u_1(x_2y_3 - x_3y_2) + u_2(x_3y_1 - x_1y_3) + u_3(x_1y_2 - x_2y_1)\right] \quad \text{(D.12)}$$

$$b = \frac{1}{2A_e}\left[u_1(y_2 - y_3) + u_2(y_3 - y_1) + u_3(y_1 - y_2)\right] \quad \text{(D.13)}$$

$$c = \frac{1}{2A_e}\left[u_1(x_3 - x_2) + u_2(x_1 - x_3) + u_3(x_2 - x_1)\right] \quad \text{(D.14)}$$

where A_e is the area of the element, given by half the determinant of the system coefficient matrix:

$$A_e = \frac{1}{2}\left(x_2y_3 + x_1y_2 + x_3y_1 - x_3y_2 - x_1y_3 - x_2y_1\right) \quad \text{(D.15)}$$

Substituting the expressions for a, b and c into (D.8) and grouping terms in u_1, u_2 and u_3 gives:

$$u_e(x,y) \;=\; u_1\;\psi_1(x,y) \;+\; u_2\;\psi_2(x,y) \;+\; u_3\;\psi_3(x,y) \quad \text{(D.16)}$$

where the functions ψ_1, ψ_2, ψ_3 are given by:

$$\psi_1(x,y) = \tfrac{1}{2A_e}\left[(x_2y_3 - x_3y_2) + (y_2 - y_3)x + (x_3 - x_2)y\right] \quad \text{(D.17)}$$

$$\psi_2(x,y) = \tfrac{1}{2A_e}\left[(x_3y_1 - x_1y_3) + (y_3 - y_1)x + (x_1 - x_3)y\right] \quad \text{(D.18)}$$

$$\psi_3(x,y) = \tfrac{1}{2A_e}\left[(x_1y_2 - x_2y_1) + (y_1 - y_2)x + (x_2 - x_1)y\right] \quad \text{(D.19)}$$

The basis functions for the finite element method, $\phi_i(x,y)$ are then constructed by adding the ψ functions from surrounding elements which are non-zero at the vertex i, as shown in figure D.4.