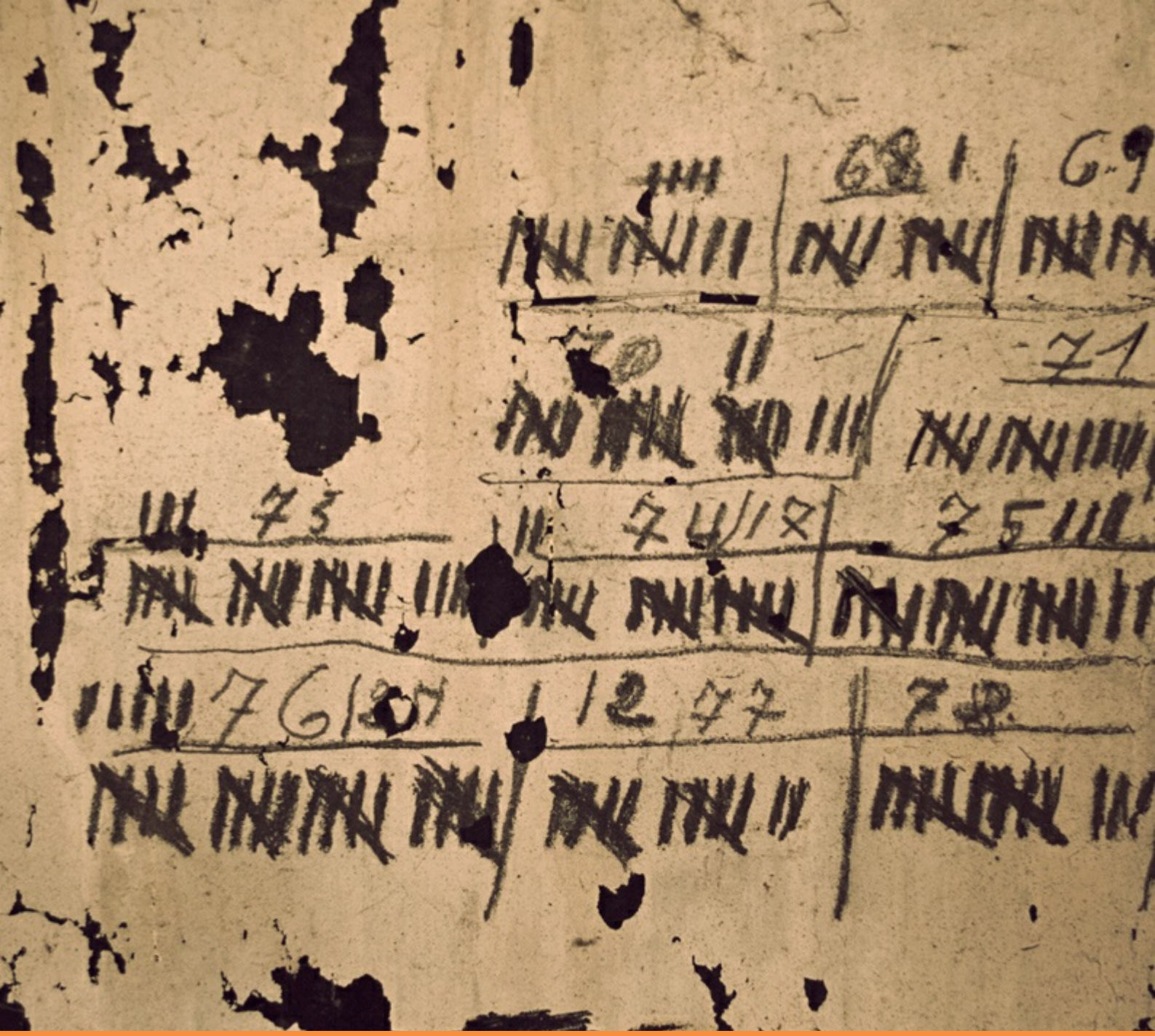

Introductory Finite Volume Methods for PDEs

Dr. L. Qian; Professor C. G. Mingham; Professor D. M. Causon

bookboon
The eBook company

Professor D. M. Causon, Professor C. G. Mingham and Dr. L. Qian

Introductory Finite Volume Methods for PDEs

Introductory Finite Volume Methods for PDEs

ISBN 978-87-7681-882-1

Contents

Preface

This material is taught in the BSc. Mathematics degree programme at the Manchester Metropolitan University, UK. The Finite Volume Method (FVM) is taught after the Finite Difference Method (FDM) where important concepts such as convergence, consistency and stability are presented. The FDM material is contained in the online textbook, 'Introductory Finite Difference Methods for PDEs' which is free to download from:

http://www.bookboon.com

It is recommended that the FDM text book is read before this book. This textbook is also freely downloadable from the above website.

The following chapters contain core material supported by pen and paper exercises together with computer-based exercises where appropriate. Supporting material, including worked solutions, and computer codes can be found at the companion website:

http://www2.docm.mmu.ac.uk/STAFF/C.Mingham/

Codes, with which the student can experiment, are written using Matlab. In the spirit of Open Source, it is hoped to reproduce these codes using Scilab (a Matlab clone, downloadable for free from http://www.scilab.org/).

The emphasis of this book is on a practical understanding of the basics of the FVM and a minimum of theory is given to underpin the method. Revision material is provided in appendices. It is recommended that anyone wishing to use the FVM to solve systems of equations for real world applications reads up on the underlying physics of the problem.

This book is intended for final year undergraduates who have knowledge of Calculus, vectors and introductory level computer programming.

1 Introduction

1.1 Introduction

The finite volume method (FVM) is an increasingly popular numerical method for the approximate solution of partial differential equations (PDEs). Most of the authors' research work has been based on the FVM (see http://www.scmdt.mmu.ac.uk/cmmfa/).

Compared to the classical finite difference method (FDM) the FVM has the following advantages:

1. Spatial discretisation is totally flexible: the mesh can accommodate to irregularly shaped boundaries to reduce geometric errors and the mesh can be refined locally to give more resolution in regions of particular interest.
2. Equations are presented in *integral form* which is often how they are derived from the underlying physical laws.
3. Because of 2) there is no need for dependent variables to be differentiable everywhere which means that a larger class of problems can be solved.
4. The FVM naturally conserves conserved variables when applied to PDEs expressing conservation laws since, as two neighboring cells share a common interface, the total flow of a conserved quantity out of one cell will be the same as that entering the other cell.

One disadvantage of the FVM over the FDM is that there is no easily accessible underlying theory (e.g. for formal accuracy). However a FVM on a uniform Cartesian mesh can be regarded as a FDM which then permits analysis based on Taylor series.

1.2 Obtaining the Integral Form from the Differential Form

The FVM can be used in 1D, 2D or 3D but a 2D treatment best encapsulates its essential elements and may be generalized easily to other dimensions and also to systems of equations. Consequently we will develop the FVM formulation using a single simple 2D PDE which we will interpret as describing the transport of a soluble pollutant in a uniform flow.

In *differential form* the 2D linear advection equation is:

$$\frac{\partial U}{\partial t} + v_x \frac{\partial U}{\partial x} + v_y \frac{\partial U}{\partial y} = 0 \tag{1.1a}$$

The independent variables are t (time, [s]) and x, y (space, [m]). The dependent variable is

U = U(t, x, y) (pollutant concentration, [kg/m^2]) and v_x and v_y are constant flow speeds [m/s] in the x and y directions respectively.

Given initial conditions,

$$U(0, x, y) = f(x, y) \tag{1.1b}$$

It is easy to show that the exact solution of (1.1a) is,

$$U(t, x, y) = f(x-v_x t, y-v_y t) \tag{1.1c}$$

The following FVM treatment requires that (1.1a) be written in a special form which we will call *finite volume form*. Let,

$$\underline{H} = U\,\underline{v} \tag{1.2a}$$

where,

$\underline{i}$ and $\underline{j}$ are the usual Cartesian basis vectors and,

$$\underline{v} = v_x \underline{i} + v_y \underline{j} \tag{1.2b}$$

is the flow velocity.

It is easy to see that units of the components of $\underline{H}$ are (kg/s)/m. $\underline{H}=\underline{H}(U)$ is called the *flux density* and is a vector field. The components of $\underline{H}$ measure the rate of mass flow through a line of unit length.

(1.1) can now be written in the *finite volume form*,

$$\frac{\partial U}{\partial t} + \nabla \cdot \underline{H} = 0 \tag{1.3}$$

where,

∇ is the well known vector differential operator, $\underline{i}\,\frac{\partial}{\partial x} + \underline{j}\,\frac{\partial}{\partial y}$.

In vector calculus the scalar quantity, $\nabla \cdot \underline{H}$, is often written div($\underline{H}$) and is the *divergence* of the vector field $\underline{H}$. A review of vectors and vector calculus is given in Appendices A and B respectively which should be read now if the reader is unfamiliar with this material.

The finite volume form (1.3) is the starting point for the finite volume method (irrespective of the particular form of $\underline{H}$).

Integrating (1.3) over an *arbitrary* (simply connected) region R in the xy-plane gives,

$$\iint_R \left(\frac{\partial U}{\partial t} + \nabla \cdot \underline{H} \right) \, dR = \iint_R 0$$

$$\therefore \iint_R \frac{\partial U}{\partial t} \, dR + \iint_R \nabla \cdot \underline{H} \, dR = 0 \tag{1.4a}$$

Using a form of Green's theorem (see Appendix B) the second integral in (1.4a) can be replaced by a *line integral* around the perimeter curve, S, of R giving,

$$\iint_R \frac{\partial U}{\partial t} \, dR + \oint_S \underline{H} \cdot \underline{n} \, ds = 0 \tag{1.4b}$$

Where $\underline{n}$ is the *outward* pointing *unit* normal vector to S at any point on S. Defining $\overline{U}$ as the average value of U over R, the first integral can be rewritten and (1.4b) becomes,

$$A \frac{\partial \overline{U}}{\partial t} + \oint_S \underline{H} \cdot \underline{n} \, ds = 0 \tag{1.4c}$$

where A is the area of R. Finally we rewrite (1.4c) to give,

$$\frac{\partial \overline{U}}{\partial t} = -\frac{1}{A} \oint_S \underline{H} \cdot \underline{n} \, ds \tag{1.5}$$

Equation (1.5) is a semi-*integral* form of (1.1a) and applies to *any* region in the xy-plane over which (1.1a) holds. In the FVM (1.5) is discretised spatially by first dividing up the computational region into a mesh consisting of a finite number of polygonal cells. Equation (1.5) is approximated over each cell to produce a finite volume scheme (FVS). The shapes of the cells are arbitrary and can be chosen to fit to the boundaries of the computational region (and any internal solid boundaries) for computational accuracy.

1.3 Finite Volume Meshes

Finite volume meshes are of two basic types: structured and unstructured. In a (2D) structured mesh cells are referenced by an ordered pair of indices, usually i and j (not to be confused with the standard Cartesian basis vectors $\underline{i}$ and $\underline{j}$). Cells in a structured 2D mesh necessarily have 4 sides. The benefit of using a structured mesh is that the indices of adjoining cells are automatically known and data may be held in 2D arrays efficiently. Furthermore a structured mesh permits the use of dimensionally split schemes (see the text, 'Introductory Finite Difference Methods for PDEs', in this series) which, when implemented on parallel computer architectures, may lead to significant decreases in compute time.

In an unstructured mesh cells are referenced by a single index. Unstructured 2D meshes usually consist of triangular cells. The locations of adjoining cells are *not* known automatically and must be stored which results in more complicated data structures. It could be argued that, for complex geometries, unstructured meshes are easier to generate than structured ones.

Figure 1 shows finite volume and (uniform) finite difference meshes for the same computational domain. The finite volume meshes fit the boundaries of the computational domain but in the finite difference mesh most mesh points do not lie on the boundaries which is a source of error. It is possible to change from Cartesian to curvilinear coordinates so that the computational domain transforms to a rectangular region which is then fitted exactly by a finite difference mesh but this requires a transformation of the PDE and introduces further complexity and errors. As we will see, the finite volume method remains referenced to the Cartesian coordinate system and flows through cell sides aligned in any direction are obtained via simple but powerful dot products.

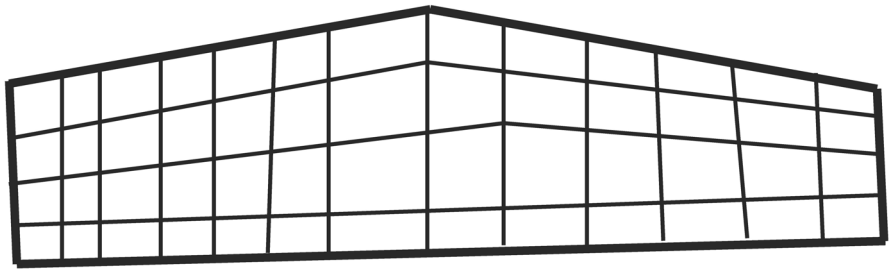

Figure 1.1a: Structured finite volume mesh.

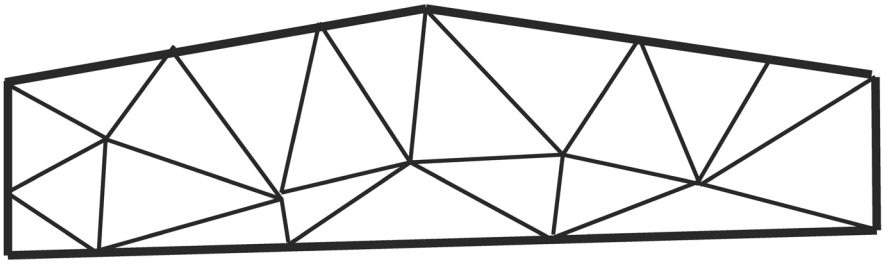

Figure 1.1b: Unstructured finite volume mesh.

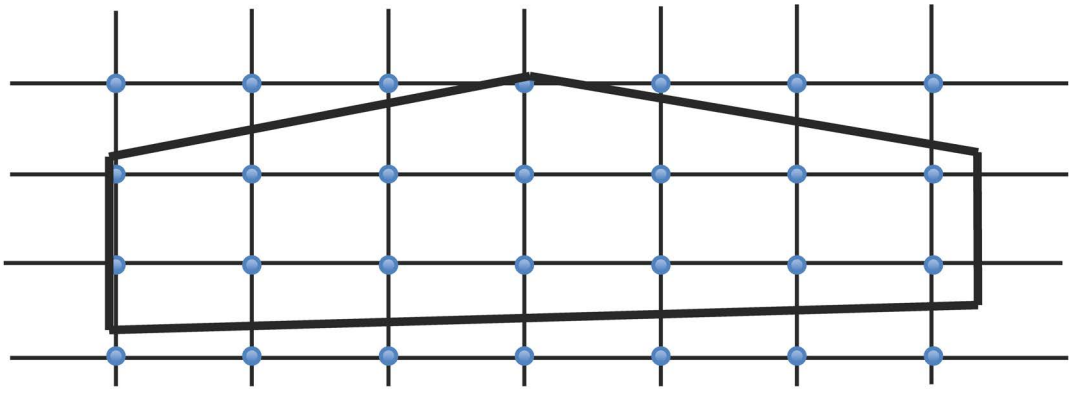

Figure 1.1c: Uniform finite difference mesh.

1.4 Discretising the Semi-Integral Equation

In the following development we will refer to a general cell which could be in a structured or an unstructured mesh. Later we will use a structured mesh for ease of indexing.

1.4.1 Cell Data

A FVS is created by discretising equation (1.5) over each cell in the computational mesh. Cells are indexed by subscript k. Let,

- A_k be the area of cell k
- U_k^n be the (approximation) to the mean value of U over cell k at time level n. U_k^n is located at the *centre* of the cell.
- $\underline{s}_k$ be the unique *outward* pointing *normal* vector to a side of cell k whose magnitude is the length of the side.

Notes:

1. In a structured 2D mesh each cell is a quadrilateral.
2. Cell areas are easily found from simple vector geometry (see Appendix A).
3. The cell centre is formally the cell centroid but a quick approximation to it is obtained by averaging the sum of respective x and y coordinates of the cell vertices.
4. The $\underline{s}$ vectors are called *side vectors*.

5. At a cell side common to two adjoining cells there are two side vectors which point *out* of their respective cells. Since they have the same length but point in opposite directions, one side vector is the negative of the other.

Figure 1.2 presents cell information at time level n for a general finite volume cell, k, with 4 sides (although a cell could have any number of sides). Mean flux density, $\underline{H}$, is labelled on a single side whose side vector is labelled $\underline{s}$. Other mean flux densities on cell sides are not shown although side vectors are shown but not labelled.

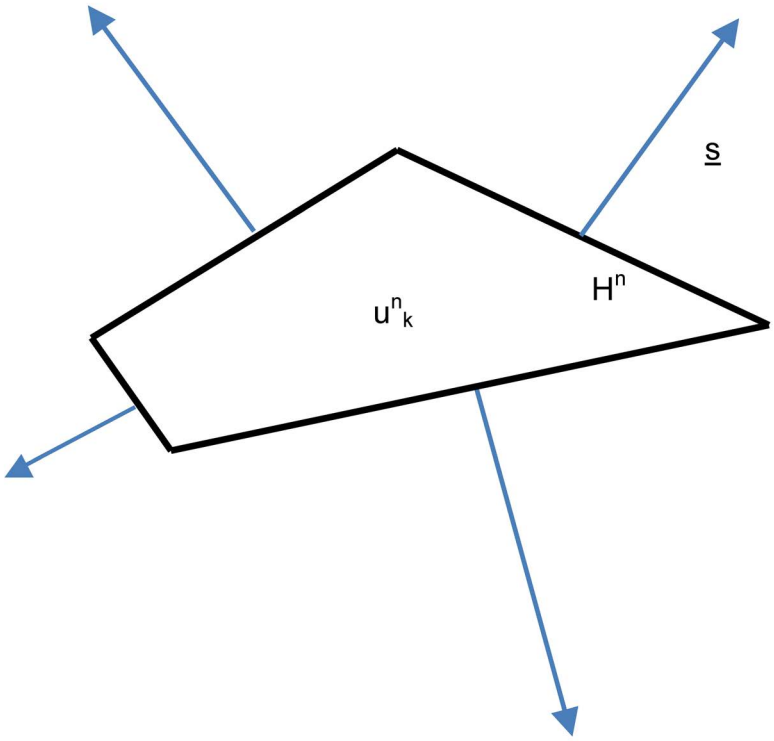

Figure 1.2: Finite volume cell k showing cell centre data u^n_k and flux density $\underline{H}^n$ on a side with side vector $\underline{s}$.

1.4.2 Spatial Discretisation

The spatial part of equation (1.5) is a line integral around the perimeter of cell k. This line integral gives the total flux *out* of cell k. Each side of cell k is a straight line so has the same normal vector at each point on it. $\underline{H}$ will, in general, vary over a cell side so we approximate it by a constant value therefore, by definition of the line integral, the total flux through a side of cell k is $\underline{H} \cdot \underline{s}$, where $\underline{s}$ is the side vector (see Appendix B) and $\underline{H}$ is the (constant) value of the flux density over the side. Hence the line integral is approximated by the sum of $\underline{H} \cdot \underline{s}$ terms over each cell side. $\underline{H}$ depends on U which is approximated at the centre of cell k at time level n by u^n_k. Let $\underline{H}^n$ be the (constant) flux at time level n on a side of cell k. We can now approximate the line integral of equation (1.5) over cell k by,

$$\oint_S \underline{H} \cdot \underline{ds} = \sum_{\text{sides}} \underline{H}^n \cdot \underline{s} \qquad (1.6)$$

(rather than introducing a complicated indexing system for cell sides it is assumed that the summation is over each side of cell k)

The spatial discretisation of equation (1.5) is therefore,

$$\frac{\partial \bar{U}}{\partial t} = -\frac{1}{A_k} \sum_{\text{sides}} \underline{H}^n \cdot \underline{s} \tag{1.7}$$

1.4.3 Time Discretisation

Time discretisation of (1.7) can take many forms. For simplicity we will use the first order forward difference approximation to the time derivative so (1.7) becomes,

$$\frac{u_k^{n+1} - u_k^n}{\Delta t} = -\frac{1}{A_k} \sum_{\text{sides}} \underline{H}^n \cdot \underline{s} \tag{1.8}$$

where Δt is the time step between time levels n and n+1 (to be determined).

Rearranging (1.8) gives the FVS,

$$u_k^{n+1} = u_k^n - \frac{\Delta t}{A_k} \sum_{\text{sides}} \underline{H}^n \cdot \underline{s} \tag{1.9}$$

1.5 Implementation

Equation (1.9) may be regarded as a **general finite volume scheme** (given the fixed time discretisation) for *any* 2D PDE that can be written in the finite volume form (1.3). **Different choices for interfaces fluxes in (1.9) give rise to different FV schemes.**

1.5.1 Time Step

(1.9) is an *explicit* scheme so the maximum value of Δt will be limited by *stability* considerations and will depend on cell area A_k and flow velocity $\underline{v}_k$ in cell k. Let the allowable time step in cell k be Δt_k then,

$$\Delta t = \min_k \left(\Delta t_k \right) \tag{1.10}$$

A formula for the time step will be given later.

1.5.2 Pseudo Code

A computer program to implement Equation (1.9) is described by the following pseudo code:

Step 1: Generate a FV mesh and store cell vertices, centres, areas and side vectors.
Step 2: Initialize the mesh (i.e. input cell centre u values) and parameters (e.g. run time).
Step 3: Input boundary conditions.
Step 4: Calculate the stable time step.

Step 5: Calculate flux densities at cell sides.

Step 6: For each cell evaluate (1.9) and update u.

Step 7: If run time is achieved output results and stop otherwise go to step 3.

1.5.3 Implementation on a Structured Mesh

On a structured mesh cells are indexed by (i, j) and each cell has 4 sides. We adopt the numbering convention that i indicates column number and j indicates row number. Useful notation is to use the subscript i+1/2, j to indicate the interface between cell (i, j) and cell (i+1, j) etc. Figure 1.3 shows cell variables in a structured mesh.

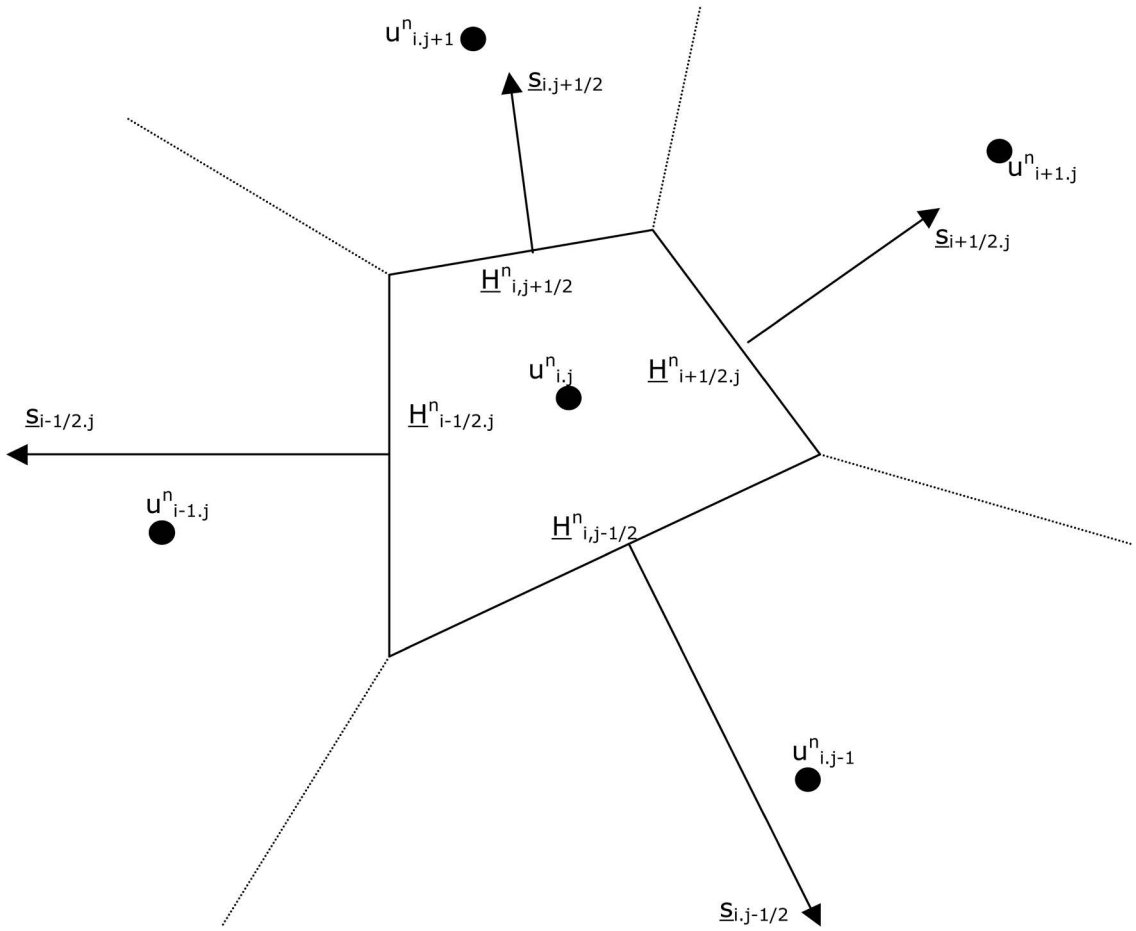

Figure 1.3: Finite volume cell (i, j) showing at time level n, flux densities, $\underline{H}^n$, on sides with side vectors, $\underline{S}$ and cell centre data, u^n .

Equation (1.9) can now be written,

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \frac{\Delta t}{A_{i,j}} \left(\underline{H}^n_{i+1/2,j} \cdot \underline{S}_{i+1/2,j} + \underline{H}^n_{i,j+1/2} \cdot \underline{S}_{i,j+1/2} + \underline{H}^n_{i-1/2,j} \cdot \underline{S}_{i-1/2,j} + \underline{H}^n_{i,j-1/2} \cdot \underline{S}_{i,j-1/2}\right) \quad (1.11)$$

Since the mesh is *structured* the 4 cells surrounding cell (i, j) are known *automatically*. It is only necessary to store lower left hand coordinates of a cell to describe the entire mesh. For example, if array entry X(i, j) contains the lower left hand x coordinate of cell (i, j) then X(i+1, j), X(i+1, j+1), X(i, j+1) contain the x coordinates of the remaining 3 vertices as we travel anticlockwise around the cell's perimeter (and similarly for the y coordinates stored in Y). This allows us to compute cell areas, centres and side vectors for each cell (which can be stored appropriately).

Notes:

1. If the structured mesh has NI cells in the i direction and NJ cells in the j direction then X and Y will be NI+1 by NJ+1. The arrays storing cell areas and centres will be NI by NJ. The array storing side vectors will be NI by NJ by 4 by 2 (since each cell has 4 side vectors which each have 2 components).
2. If storage is an issue then side vectors and even cell areas and centres may be computed when needed.
3. $\underline{s}_{i+1/2,j} = -\underline{s}_{i+1-1/2,j}$ so not all side vectors need be calculated.

Exercise 1

1. Show that (1.1c) is the general solution to (1.1a).
2. If possible, express the following PDEs in finite volume form and in each case write down the flux vector:

a) $$\frac{\partial U}{\partial t} + \frac{\partial U}{\partial x} = 2\frac{\partial U}{\partial y}$$

b) $\frac{\partial U}{\partial t} + \frac{\partial(-xU)}{\partial x} + \frac{\partial(yU)}{\partial y} = 0$

c) $\frac{\partial U}{\partial t} + 3\frac{\partial^2 U}{\partial x^2} + 2\frac{\partial^2 U}{\partial y^2} = 0$

d) $\frac{\partial U}{\partial t} + \frac{\partial(U^2)}{\partial x} + \frac{\partial(yU)}{\partial y} = 0$.

3. A 2D rectangular computational domain has its bottom left hand corner at (0, 0) and its top right hand corner at (16, 6).

a) Sketch the region.

b) Sketch the discretised region using a *uniform* structured FV mesh with 4 cells in the x (i) direction and 3 cells in the y (j) direction.

c) Sketch an *unstructured* FV mesh for the region using 12 cells.

d) State one advantage and one disadvantage of a structured mesh over an unstructured mesh.

4. Using your uniform structured FV mesh sketch from 3b) label each cell with appropriate (i, j) indices and calculate the 4 side vectors for cell (3,2) and label them as per Figure 1.3.

5. A 2D computational domain is described by the area in the first quadrant between two circles centred at the origin whose radii are 8m and 2m.

a) Sketch the region.

b) Sketch the discretised region using a uniform structured FV mesh with 4 cells in the circumferential (i) direction and 3 cells in the radial (j) direction

6. Using your uniform structured FV mesh sketch from 5b) label each cell with appropriate (i, j) indices and calculate the 4 side vectors for cell (3,2) and label them as per Figure 1.2.

7. A uniform rectangular Cartesian FV mesh consists of Δx by Δy quadrilateral cells. Find all 4 side vectors for cell (i, j) and label them correctly.

8. Show that side vectors for the common side between 2 adjoining cells are the negative of each other.

9. From what you know about explicit finite difference schemes discuss how you would expect the time step to be affected by cell size and flow velocity in an explicit FV scheme.

2 Finite Volume Schemes

2.1 Introduction

The finite volume method (FVM) applies to PDEs written in the form,

$$\frac{\partial U}{\partial t} + \nabla \cdot \underline{H} = Q \tag{2.1}$$

$\underline{H} = \underline{H}(U)$ is the flux (density) vector and Q is a *source* term (which was zero in Chapter 1). For simplicity we will focus our attention on 2D PDEs. Integrating (2.1) over a region R of area A with perimeter C and using a version of Green's theorem (Appendix B) gives,

$$A\frac{\partial \overline{U}}{\partial t} + \oint_C \underline{H} \cdot \underline{n}\ ds = A\ \overline{Q} \tag{2.2}$$

where bars denote mean values over R and $\underline{n}$ is the *outward* pointing *unit* normal vector at each point on C. Rearranging and dropping the overbars gives,

$$\frac{\partial U}{\partial t} = -\frac{1}{A}\oint_C \underline{H} \cdot \underline{n}\ ds + Q \tag{2.3}$$

Equation (2.3) is valid over any region R. The FVM tessellates R using a computational mesh of polygonal cells and approximates (2.3) on each cell. Discretising (2.3) over cell k using a first order forward difference in time gives,

$$\frac{u_k^{n+1} - u_k^n}{\Delta t} = -\frac{1}{A_k}\left(\sum_{\text{sides}} \underline{H}^n \cdot \underline{s}\right) + q_k^n \tag{2.4}$$

where,

- Δt is the time step between time levels n and n+1,
- u_k^n and q_k^n are the (approximations to) *mean* values of U and Q respectively at time level n located at the centre of cell k,
- A_k is the area of cell k
- the sum is taken around all sides of cell k where on a side, $\underline{H}^n$ is constant and $\underline{s}$ is the *outward* pointing normal vector whose length is the length of the side. $\underline{s}$ are called *side vectors* and the $\underline{H}^n$ are called *interface fluxes*.

In the following treatment we apply (2.4) on a *structured* mesh whose cells are indexed by (i, j) and for simplicity we take Q = 0.

Rearranging (2.4) gives,

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \frac{\Delta t}{A_{i,j}} \left(\sum_{\text{sides}} \underline{H}^{n} \cdot \underline{s} \right) \tag{2.5}$$

Adopting the convention from Chapter 1 of using ½ in subscripts to denote cell sides (interfaces) in a structured mesh (see Figure 1.3) (2.5) becomes,

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \frac{\Delta t}{A_{i,j}} \left(\underline{H}_{i+1/2,j}^{n} \cdot \underline{s}_{i+1/2,j} + \underline{H}_{i,j+1/2}^{n} \cdot \underline{s}_{i,j+1/2} + \underline{H}_{i-1/2,j}^{n} \cdot \underline{s}_{i-1/2,j} + \underline{H}_{i,j-1/2}^{n} \cdot \underline{s}_{i,j-1/2} \right) \tag{2.6}$$

Notes:

1. Equation (2.6) may be regarded as a general *explicit* finite volume scheme (FVS).
2. (2.6) is first order in time due to the time discretisation used but higher order time accuracy could be achieved by a multi-step approach.
3. A particular FVS based on (2.6) is constructed by *estimating* the interface fluxes (i.e. the $\underline{H}$ values on cell sides).
4. Since $\underline{H} = \underline{H}(U)$, interface flux estimation can be done by extrapolation of cell centre U values to interfaces or by direct extrapolation cell of cell centre $\underline{H}$ values to interfaces.

2.2 FVM on a Cartesian Mesh

Before implementing (2.6) on a general structured mesh it is instructive to apply it to the *special case* of a Cartesian mesh with constant cell sides of lengths Δx and Δy in the x and y directions respectively. Figure 2.1 shows cell (i, j) and relevant variables.

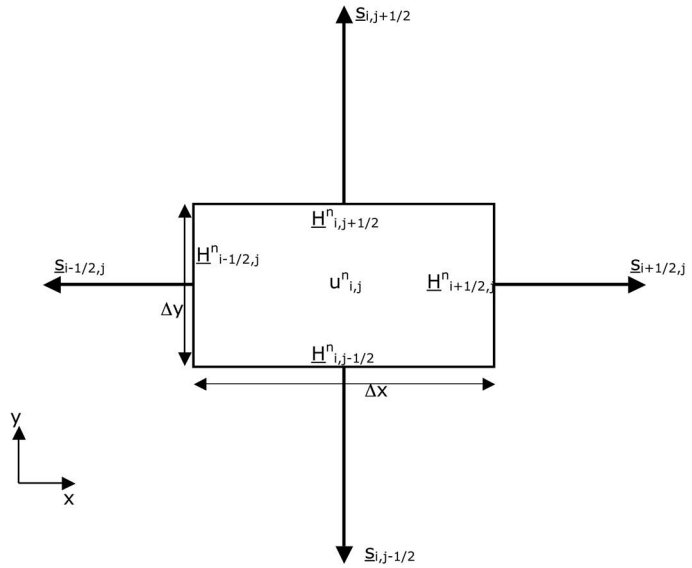

Figure 2.1: Cartesian cell (i, j) showing side vectors and variables.

Side vectors are clearly parallel to x and y axes and, from simple vector algebra (Appendix A), we have,

$$\begin{aligned} \underline{S}_{i,j-1/2} &= 0\ \underline{i} - \Delta x\ \underline{j}, \quad \underline{S}_{i+1/2,j} = \Delta y\ \underline{i} + 0\ \underline{j} \\ \underline{S}_{i,j+1/2} &= 0\ \underline{i} + \Delta x\ \underline{j}, \quad \underline{S}_{i-1/2,j} = -\Delta y\ \underline{i} + 0\ \underline{j} \end{aligned} \tag{2.7}$$

Hence (2.6) becomes,

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \frac{\Delta t}{\Delta x \Delta y}\left(\underline{H}_{i+1/2,j}^{n} \cdot (\Delta y\ \underline{i}) + \underline{H}_{i,j+1/2}^{n} \cdot (\Delta x\ \underline{j}) + \underline{H}_{i-1/2,j}^{n} \cdot (-\Delta y\ \underline{i}) + \underline{H}_{i,j-1/2}^{n} \cdot (-\Delta x\ \underline{j})\right) \tag{2.8}$$

We write $\underline{H}$ in terms of its components.

Let,

$$\underline{H} = F\ \underline{i} + G\ \underline{j} \tag{2.9}$$

therefore (2.8) becomes,

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \frac{\Delta t}{\Delta x \Delta y}\left(F_{i+1/2,j}^{n}\Delta y + G_{i,j+1/2}^{n}\Delta x - F_{i-1/2,j}^{n}\Delta y - G_{i,j-1/2}^{n}\Delta x\right) \tag{2.10}$$

which can be written as,

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \Delta t\left(\left[\frac{F_{i+1/2,j}^{n} - F_{i-1/2,j}^{n}}{\Delta x}\right] + \left[\frac{G_{i,j+1/2}^{n} - G_{i,j-1/2}^{n}}{\Delta y}\right]\right) \tag{2.11}$$

Notes:

1. In a Cartesian mesh cell sides are parallel to the x and y axes so the required fluxes normal to cell sides (i.e. the $\underline{H} \cdot \underline{s}$ terms) must be in the x and y directions. These are, as expected, the $\underline{i}$ and $\underline{j}$ components of $\underline{H}$, namely F and G as shown in (2.11).
2. In order to obtain a *specific* FVS from (2.11) the F and G values must be *estimated* in some way.
3. The terms in the square brackets in (2.11) are recognisable as *finite difference* approximations to $\frac{\partial F}{\partial x}$ and $\frac{\partial G}{\partial y}$ based on values of F and G on the interfaces between cell (i, j) and its neighbouring cells.
4. From Note 3 we may conclude that,

   'on a Cartesian mesh a finite volume scheme reduces to

   a finite difference scheme'

5. A finite difference scheme can be regarded as a special case of a finite volume scheme.

2.2.1 Specific Example: First Order Upwind (FOU) Scheme

We apply (2.11) to the 2D linear advection equation. In this equation $\underline{H} = F\,\underline{i} + G\,\underline{j}$ where, $F = (v_x\, U)\,\underline{i}$, $G = (v_y\, U)\,\underline{j}$. Simplifying, (2.11) becomes,

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \Delta t \left(v_x \left[\frac{u_{i+1/2,j}^{n} - u_{i-1/2,j}^{n}}{\Delta x} \right] + v_y \left[\frac{u_{i,j+1/2}^{n} - u_{i,j-1/2}^{n}}{\Delta y} \right] \right) \tag{2.12}$$

At time level n, $u_{i,j}^{n}$ is *known* and is at the centre of each cell (i, j). It remains to *estimate* $u_{i+1/2,j}^{n}, u_{i-1/2,j}^{n}, u_{i,j+1/2}^{n}, u_{i,j-1/2}^{n}$ which are on the four interfaces of cell (i, j) with its four neighbouring cells. Suppose the flow velocity components v_x and v_y are both positive. One reasonable estimation procedure is to take the interface u values from the neighbouring *upstream* known cell centre values (see Figure 2.2). This gives,

$$u_{i+1/2,j}^{n} \approx u_{i,j}^{n} \quad , \quad u_{i-1/2,j}^{n} \approx u_{i-1,j}^{n}, \quad u_{i,j+1/2}^{n} \approx u_{i,j}^{n}, \quad u_{i,j-1/2}^{n} \approx u_{i,j-1}^{n} \tag{2.13}$$

and (2.12) becomes,

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \Delta t \left(v_x \left[\frac{u_{i,j}^{n} - u_{i-1,j}^{n}}{\Delta x} \right] + v_y \left[\frac{u_{i,j}^{n} - u_{i,j-1}^{n}}{\Delta y} \right] \right) \tag{2.14}$$

This finite volume scheme is identical to the classical FOU finite difference scheme.

Notes:

1. The terms in the square brackets in (2.14) are finite difference approximations to $\frac{\partial U}{\partial x}$ and $\frac{\partial U}{\partial y}$ using first order backward differences.

2. (2.14) was derived on the basis of quantities averaged over cells and fluxes at cell interfaces and is consequently a finite volume scheme but is *indistinguishable* from a finite difference scheme derived from Taylor theory.
3. Interface flux estimation could be regarded as extrapolation of cell centre u values to cell interfaces using *gradients* of u. In this case it is *assumed* that the gradient of u in each cell is zero so that it takes the same value at the (downstream) interface as at the cell centre. More accurate estimation of interface fluxes can be achieved by first calculating a gradient (vector) for u in each cell from its surrounding values then using it to extrapolate cell centre u values to cell interface u values from which interface fluxes are then calculated.
4. This method produces a *single* value of H at an interface.

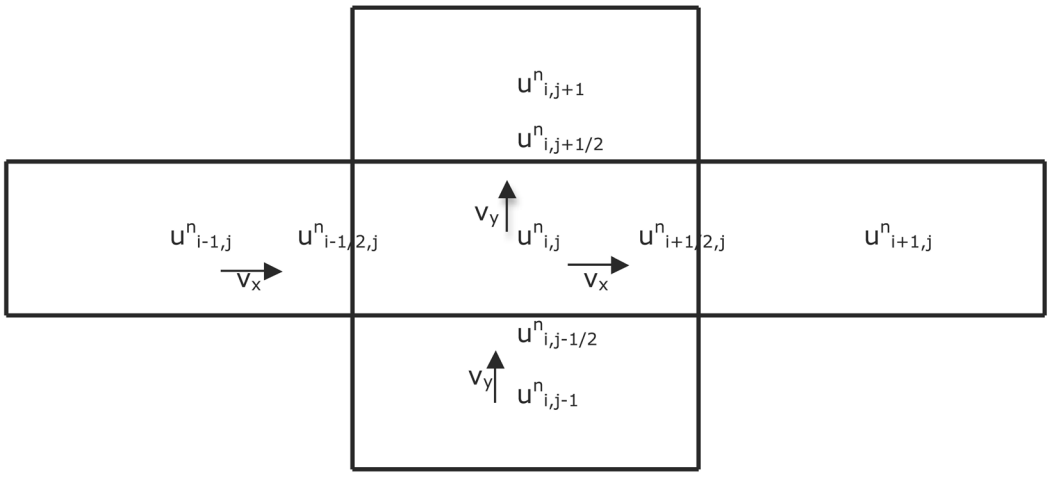

Figure 2.2: Interface values for an upwind 2D linear advection FVS

2.3 Finite Volume Schemes in 1D and 3D

We have focused on the FVM in 2D. The FVM can be used in 1D and 3D.

2.3.1 Finite Volume Schemes in 3D

FVS in 3D are beyond the scope of these notes although in principle it is simply a matter of replacing polygonal cells with polyhedral cells, areas by volumes, sides by surfaces and Green's theorem by Gauss' Divergence theorem. Constructing 3D meshes is not trivial and running 3D codes may required a great deal of computer power.

2.3.2 Finite Volume Schemes in 1D

The FVM also applies to 1D. If we use x to denote this single dimension then a 1D finite volume mesh consists of a *single row* of rectangular cells with constant height, Δy (y direction) as displayed in Figure 2.3.

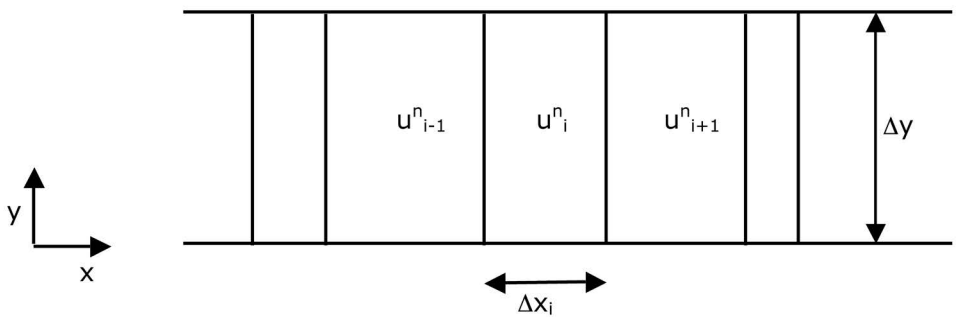

Figure 2.3: 1D finite volume mesh

In a 1D FVS cells and variables are indexed by a single index i. There is flow only in the x direction so $\underline{H} = F\,\underline{i}$ and thus (2.6) reduces to,

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{A_i}\left(F^n_{i+1/2}\,\underline{i}\cdot \underline{S}_{i+1/2} + F^n_{i-1/2}\,\underline{i}\cdot \underline{S}_{i-1/2}\right) \tag{2.15a}$$

For each cell the two side vectors (in the x direction) are $\Delta y\,\underline{i}$ and $-\Delta y\,\underline{i}$ and $A_i = \Delta x_i\,\Delta y$ so (2.15a) becomes,

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{\Delta x_i}\left(\underline{F}^n_{i+1/2} - \underline{F}^n_{i-1/2}\right) \tag{2.15b}$$

Notes:

1. (2.15b) can be regarded as a *finite difference* scheme in which fluxes are evaluated in between mesh points.
2. Δy plays no part in the final scheme (2.15b).
3. Δx may vary from cell to cell.
4. Cell centres lie on a straight line parallel to the x-axis so the scheme really is 1D.

2.3.3 Finite Volume Schemes in pseudo-1D

Rather than restricting our computational domain to a straight line parallel to the x-axis as in the previous section, we consider a curved line in a plane (which could represent, for example, a long, thin river). This domain, although existing in 2D, is essentially 1D since we only need a single variable to measure distance along it. We will refer to this case as pseudo-1D. The FVM copes with this case using a *single* row of cells as in the true 1D case. Figure 2.4 illustrates a typical geometry. There is *no flow* through the upper and lower cell sides (which we may regard as *solid boundaries*) so (2.6) becomes,

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{A_i} \left(\underline{H}^n_{i+1/2} \cdot \underline{s}_{i+1/2} + \underline{H}^n_{i-1/2} \cdot \underline{s}_{i-1/2} \right) \tag{2.16}$$

Notes:

1. Only a single index, i, is needed as there is only a single row of cells.
2. (2.16) shows the power of the FVM. There is no need to transform coordinates to fit the domain. Fluxes are projected into the correct direction by use of the dot product and everything is referenced to the standard Cartesian system.
3. We may solve a 2D problem on a structured mesh by *dimensional splitting*. We simply solve a sequence of pseudo-1D problems in the i and j directions in a similar way to the finite difference method approach. This approach is inherently parallelisable as each of the column (or row) computations can be done at the same time.

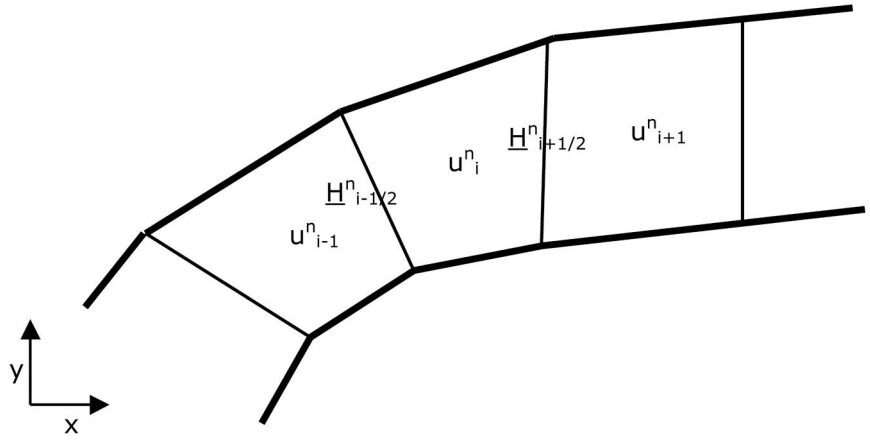

Figure 2.4: Psuedo-1D mesh showing variables and solid boundaries (thick lines).

2.4 Time Step Calculation for a Finite Volume Scheme

Equation (2.6) is our general FVS on a structured mesh. The scheme is explicit so (if we have studied finite difference schemes) we expect the time step, Δt, to be *limited* by stability considerations. Unlike the finite difference method (FDM), the FVM does not have a readily accessible stability theory so we use a heuristic approach to determine the maximum allowable time step. Any formula that we construct should make sense on a Cartesian mesh where the FVM and FDM coincide.

We argue that in a time step, Δt, information should not be able to propagate across more than one cell so it is reasonable that Δt depends directly on cell size and inversely on the components of 'flow velocity', $\underline{v}$, normal to cell faces.

There are two flow directions to consider namely the i direction and the j direction. The 'length' of cell (i, j) in the i direction is approximately its area, $A_{i,j}$, divided by a side length in the j direction. Using side (i+1/2, j), this length is $|\underline{s}_{i+1/2,j}|$. The component of flow velocity in the i direction is $\underline{v}.\underline{n}_{i+1/2,j}$ where $\underline{n}_{i+1/2,j}$ is the *unit* normal vector to cell side (i+1/2, j). Hence (since distance is the product of speed and time) the maximum time step, Δt_i, in the i direction is,

$$\Delta t_i \leq \frac{A_{i,j} / \left|\underline{s}_{i+1/2,j}\right|}{\left|\underline{v} \cdot \underline{n}_{i+1/2,j}\right|} \tag{2.17a}$$

By definition, $\underline{s}_{i+1/2,j} = |\underline{s}_{i+1/2,j}|\ \underline{n}_{i+1/2,j}$, so (2.17a) simplifies to,

$$\Delta t_i \leq \frac{A_{i,j}}{\left|\underline{v} \cdot \underline{s}_{i+1/2,j}\right|} \tag{2.17b}$$

Similarly the maximum time step, Δt_j, in the j direction is,

$$\Delta t_j \leq \frac{A_{i,j}}{\left|\underline{v} \cdot \underline{s}_{i,j+1/2}\right|} \tag{2.17c}$$

Hence a reasonable heuristic formula for the overall time step, Δt, is,

$$\Delta t \leq \min_{i,j}\left\{\min\left\{\frac{A_{i,j}}{\left|\underline{v} \cdot \underline{s}_{i+1/2,j}\right|}, \frac{A_{i,j}}{\left|\underline{v} \cdot \underline{s}_{i,j+1/2}\right|}\right\}\right\} \tag{2.17d}$$

which may be written more compactly as,

$$\Delta t \leq \min_{i,j}\left\{\frac{A_{i,j}}{\max\left\{\left|\underline{v} \cdot \underline{s}_{i+1/2,j}\right|, \left|\underline{v} \cdot \underline{s}_{i,j+1/2}\right|\right\}}\right\} \tag{2.17e}$$

Notes:

1. We have used the term 'flow velocity' to denote the speed of propagation of information. Flow velocity is not necessarily the actual velocity of the material flow. In general flow velocity varies in both time and space (hence the superscripts and subscripts on $\underline{v}$).
2. For each cell we estimate cell lengths and components of flow velocity in the i and j directions. These directions refer to the mesh and should not be confused with the x and y directions indicated by the Cartesian basis vectors $\underline{i}$ and $\underline{j}$.
3. Maximum absolute values are used to determine the maximum flow speed component.
4. In general cell areas vary so we take the *minimum* of the allowable time steps for each cell. Small cells lead to small time steps.
5. (2.17e) is a rough guide and can be tuned to a particular FVS by multiplying the right hand side by a constant which can be determined by numerical experiment.

If we apply (2.17e) to the 1D linear advection equation in the x direction with uniform (Δx by Δy) rectangular cells then,

$A_{i,j} = \Delta x \, \Delta y$, $\underline{v} = v_x \, \underline{i}$ and $\underline{s}_{i+1/2,j} = \Delta y \, \underline{i}$ so (2.17) becomes,

$$\Delta t \leq \frac{\Delta x}{|v_x|} \tag{2.18}$$

which makes sense because it says that information cannot travel across more than one cell in a single time step which agrees with our initial reasoning.

2.5 Finite Volume FOU 2D Scheme

We derive a particular FVS for the 2D linear advection equation. In this equation $\underline{H} = \underline{v}\, U$ where $\underline{v} = v_x \underline{i} + v_y \underline{j}$ with v_x and v_y constant. As in Section 2.2.1 we estimate U (and therefore $\underline{H}$) at cell interfaces by the *upwind* neighbouring cell centre values, u. We take the simplest case where flow speeds are always positive in the i and j directions, i.e.

$$\underline{v} \cdot \underline{S}_{i+1/2,j} \geq 0, \quad \underline{v} \cdot \underline{S}_{i,j+1/2} \geq 0, \quad \forall i, j \ . \tag{2.19}$$

Estimates of u at interfaces are given by (2.13) and the general FVS (2.6) becomes the FOU scheme,

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \frac{\Delta t}{A_{i,j}} \left(\underline{H}_{i,j}^{n} \cdot \underline{S}_{i+1/2,j} + \underline{H}_{i,j}^{n} \cdot \underline{S}_{i,j+1/2} + \underline{H}_{i-1,j}^{n} \cdot \underline{S}_{i-1/2,j} + \underline{H}_{i,j-1}^{n} \cdot \underline{S}_{i,j-1/2} \right) \tag{2.20a}$$

which may be written as,

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \frac{\Delta t}{A_{i,j}} \left(u_{i,j}^{n} \underline{v} \cdot \underline{S}_{i+1/2,j} + u_{i,j}^{n} \underline{v} \cdot \underline{S}_{i,j+1/2} + u_{i-1,j}^{n} \underline{v} \cdot \underline{S}_{i-1/2,j} + u_{i,j-1}^{n} \underline{v} \cdot \underline{S}_{i,j-1/2} \right) \tag{2.20b}$$

Notes:

1. Our upwind FVS was derived on the assumption (2.19) and may be adapted easily to the general case where flow components can be in any direction.
2. It is better to think of the scheme as (2.20a) rather than (2.20b) because the former explicitly refers to total flux through cell interfaces.
3. The upwind scheme may be regarded as extrapolating u from cell centres to cell interfaces in the direction of flow based on a *zero gradient* for u in each cell which gives a spatially first order accurate scheme. Higher spatial accuracy may be achieved by using (non-zero) gradients of u calculated from neighbouring cell centre u values.

2.6 Boundary Conditions

If we look at (2.20a) we see that values are needed in cells (i-1, j) and (i, j-1) and so, at the left and lower mesh boundaries when i=1 or j=1, we are referring to values in cells (0, j) or (i, 0) which we do not have. These cells are called ghost cells and values in them are called ghost values. We specify ghost values according to the boundary conditions in the problem in the same way as in the finite difference method. We will detail two common boundary conditions.

2.6.1 Transmissive (Zero Gradient) Boundary Condition

The idea of a transmissive boundary is that information can pass through and out of the domain. A simple way to implement this condition is by imposing a zero gradient (in the appropriate direction) for U at the boundary. This is illustrated by the following example.

Suppose that cell interface (1/2, j) is a transmissive boundary. By imposing a zero gradient on U (in the i direction) at (1/2, j) the value in the neighbouring ghost cell (0, j) is simply obtained from neighbouring interior cell (1, j). i.e.

$$u_{0,j}=u_{1,j}. \tag{2.21}$$

2.6.2 Solid (No Flow) Boundary Condition

Suppose that cell interface (1/2, j) is on a solid boundary. There are two ways of implementing a solid boundary condition which we present in the following examples.

2.6.2.1 Direct Method

In this method we calculate $\underline{H}.\underline{s}$ *directly* on the solid boundary. There is no flow through a solid boundary which means that at any point on a solid boundary the component of flow velocity, $\underline{v}$, normal to the boundary is zero. Hence at a solid boundary, $\underline{v} \cdot \underline{n} = 0$, where $\underline{n}$ is a unit vector normal to the boundary. So at solid cell interface (1/2, j),

$$\underline{v}_{1/2,j} \cdot \underline{n}_{1/2,j}=0. \tag{2.22a}$$

which is equivalent to,

$$\underline{v}_{1/2,j} \cdot \underline{s}_{1/2,j}=0. \tag{2.22b}$$

where $\underline{s}_{1/2,j}$ is the side vector.

In the FVS we must evaluate $\underline{H}_{1/2,j} \cdot \underline{s}_{1/2,j}$. This calculation is simplified by (2.22b). For example, in the linear advection equation $\underline{H} = U\underline{v}$ so,

$$\underline{H}_{1/2,j} \cdot \underline{s}_{1/2,j} = (u_{1/2,j}\,\underline{v}_{1/2,j}) \cdot \underline{s}_{1/2,j} = u_{1/2,j}\,(\underline{v}_{1/2,j} \cdot \underline{s}_{1/2,j}) = 0.$$

2.6.2.2 Indirect Method

An alternative *indirect* method is to copy all variables, *except for velocity*, from the neighbouring interior cell (1, j) into the ghost cell (0, j) and to deal with the velocity in the following way. Let $\underline{v}_{1,j}$ denote the velocity in cell (1, j) and write it as the sum of velocities normal and tangential to the solid interface (1/2, j). Let these velocities be $\underline{v}_n$ and $\underline{v}_t$ respectively. Then,

$$\underline{v}_{1,j} = \underline{v}_n + \underline{v}_t \tag{2.23a}$$

There are two cases:

Case 1: No Slip Boundary

Here the velocity on the solid boundary is zero. Therefore both normal and tangential components of velocity are zero. This suggests that the velocity in the ghost cell is,

$$\underline{v}_{0,j} = -\underline{v}_n + -\underline{v}_t = -\underline{v}_{1,j} \tag{2.23b}$$

Case 2: Slip Boundary

Here the normal velocity on the solid boundary is zero but the tangential velocity is unchanged as there is no friction. This suggests that the velocity in the ghost cell is,

$$\underline{v}_{0,j} = -\underline{v}_n + \underline{v}_t \tag{2.23c}$$

Notes:

1. There exist more sophisticated transmissive boundary condition treatments which are beyond the scope of these notes. A practical way of dealing with a problematic transmissive boundary is to enlarge the computational domain so that the transmissive boundary is a long way from the region of interest.
2. The direct method for implementing solid boundary conditions is mathematically elegant but computationally it requires special treatment (conditional statements) when looping through cells. The indirect method, though less elegant, may be more computationally efficient especially on vector or parallel computers.

2.7 Coding a Finite Volume Solver

Coding a FVS is best done using a structured code so that each part can be independently verified. The following is a rough guide to the steps used:

Step 1: Generate the mesh, store it.

Step 2: Read in the mesh and calculate and store cell areas, cell centres and side vectors. Input parameters (e.g. run time).

Step 3: Initialise cell centre values.

Step 4: Append ghost cells to arrays if required.

Step 5: Start time marching.

Step 6: Calculate the time step.

Step 7: Input boundary conditions.

Step 8: Calculate interface fluxes.

Step 9: Implement scheme and update results and time.

Step 10: If run time has been achieved output results otherwise go to Step 6.

Exercise 2

1. Obtain (2.3) from (2.1).
2. Explain how (2.4) is obtained from (2.3).
3. By representing cell sides in Figure 2.1 by vectors going around anticlockwise show that (2.7) is correct.
4. Obtain (2.8) from (2.6).
5. Obtain (2.10) from (2.8).
6. Obtain (2.11) from (2.10).
7. Obtain an upwind FVS from (2.12) when v_x is positive and v_y is negative.
8. Obtain a FVS from (2.12) by estimating interface fluxes from the average flux of the left and right (or top and bottom) values at neighbouring cell centres. Which finite difference scheme have you created? Will it work? Explain.
9. Obtain (2.14). Why is $\underline{H}$ unique at an interface?

10. Obtain (2.15b) from (2.6).

11. Obtain (2.16) from (2.6).

12. It is required to solve a problem using an explicit FVS on a 1D domain which is 100m long and is discretised using 50 uniform Cartesian cells. The maximum flow speed in the problem is 3m/s. Estimate the maximum stable time step. Critically assess your answer.

13. Figure 2.5 is a pseudo-1D mesh showing initial u values at the centres of cells i-1, i and i+1. The flux density vector $\underline{H} = \underline{v}\,U$. Initial values of $\underline{v}$ are, $\underline{v}_{i-1} = 2\underline{i} + 2\underline{j}$, $\underline{v}_i = 4\underline{i} + \underline{j}$, $\underline{v}_{i+1} = 3\underline{i} + 0.3\underline{j}$. Calculate the maximum allowable time step for the upwind FVS for this problem and, expressing everything in the correct notation, use the scheme to calculate the u value in cell i at this time (i.e. after one time step).

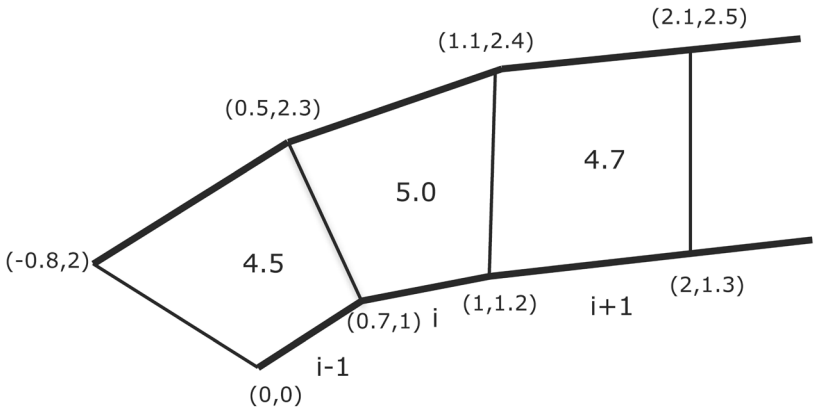

Figure 2.5: Pseudo-1D mesh with initial cell centre u values

14. Figure 2.6 is a small (test) FV mesh with 3 cells in the i direction and 2 cells in the j direction. Initial u values are given at cell centres.

    The following questions are designed to test your understanding of the 2D FVM. The answers can be used to verify a 2D FV code so you are advised to tabulate your results appropriately.

a) For each cell calculate cell areas and side vectors.
b) Given that flow velocity, $\underline{v} = 3\,\underline{i} + 4\,\underline{j}$ everywhere, estimate the maximum allowable time step for an explicit finite volume solver.
c) Given that flux density, $\underline{H} = \underline{v}\,U$, implement a single time step (using the time step and flow velocity from b)) of the FOU scheme to find u values in all cells where this is possible.

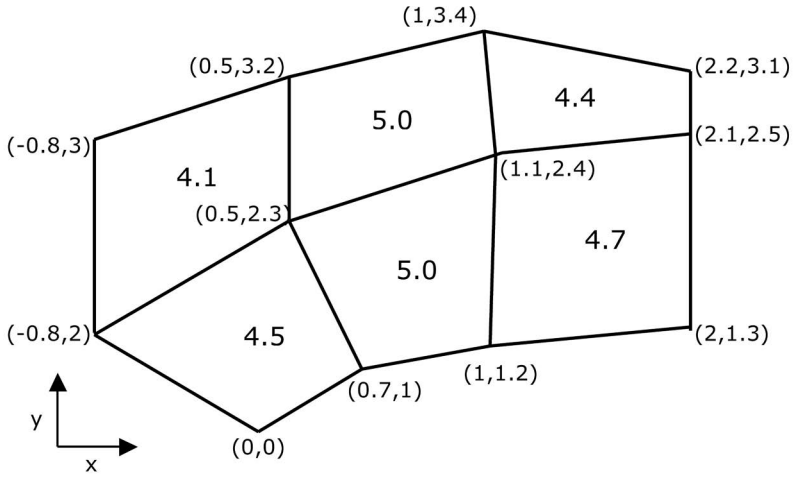

Figure 2.6: Test 2D mesh with initial cell centre u values.

15. In Figure 2.6 the leftmost cell side on each row corresponds to a transmissive boundary. Write down the indices of the boundaries and ghost cells and find u in each ghost cell.

16. In Figure 2.6 the leftmost cell side on each row corresponds to a slip solid boundary.

a) Using the correct indexing write down the total flux through each solid cell side.
b) Using the indirect method, find the values of u and $\underline{v}$ in the ghost cells and lable them appropriately.
c) Find u and $\underline{v}$ in the ghost cells for no-slip solid boundaries.

17) 'FVlinearadvectionFOU2D' is a code to implement the finite volume FOU scheme to solve the 2D linear advection equation. This code is written in a structured way so that each part can be verified easily.

a) Verify the code by reference to Q 14.
b) Implement the code on a uniform Cartesian mesh with a Gaussian initial profile based at the centre of the mesh. Use positive flow speeds.
c) Find the maximum allowable time step.
d) Comment critically on your results in the light of the exact solution.
e) Make your mesh non-Cartesian by adding small random numbers to the mesh coordinates. Plot your mesh. Re-run your program and comment on your results.
f) Alter the code so that it is a FOU scheme for any flow speeds.

3 Derivation of Equations

3.1 Introduction

In this chapter we will derive the 2D linear advection equation in both differential and integral forms by using an underlying law of nature. The mathematical techniques used to derive the equations extend naturally to other common (systems of) equations which describe a wide range of physical phenomena.

3.2 Conservation Laws

A *closed system* may be thought of as an isolated region of space whose boundaries prevent any interaction with the outside world. There are (classical) physical laws which state that the *total* amount of a particular quantity in a closed system remains constant; these laws are called *conservation laws*. Quantities which obey conservation laws are said to be *conserved quantities*. An example of a conservation law is the *conservation of mass* which states that 'the total mass in a closed system remains constant'. Other well known conserved quantities are momentum and energy. In the following analysis our conserved quantity will be *mass*.

Consider the total mass of some material in a *fixed* region of space which can have interactions with its surroundings (so is not a closed system). Since mass is a conserved quantity, the only way the total mass of the material in the region can change is if there is a *flow* of material across the *boundary* of the region. At any time material may flow into the region at some points on the perimeter and out of the region at other points on the perimeter. At a particular time if there is a *net outflow* of material across the entire perimeter then the mass of material inside the region will be decreasing (and vice versa) and its time derivative will therefore be negative at this time. If we adopt the convention that flow *out* of the region is considered positive (and therefore flow in to the region is negative) then our convention says that,

the rate of change of total mass in a region is the negative of the total mass flow rate out of the region. (3.1)

We will use these ideas to derive the well known 2D linear advection equation.

3.3 Control Volume Approach

We start with a *fixed* region of space commonly called a *control volume*. Since we are going to derive a 2D equation, the control volume is actually an arbitrary 2D region, R, in the xy-plane. Let C be the perimeter of R and let A be the area of R (see Figure 3.1).

3.3.1 Total Mass in R

We will consider how the *total* mass of material changes inside R. Let U(t, x, y) be the *density* of the material (kg/m^2) at time t at any point (x, y) in the xy-plane. The total mass inside R is therefore,

$$\iint_R U \, dR \tag{3.2a}$$

The mean value of U over R is denoted, $\bar{U}$, and (see Appendix B),

$$\bar{U} = \frac{1}{A}\iint_R U \, dR \tag{3.2b}$$

hence the total mass in R is,

$$\iint_R U \, dR = A \, \bar{U} \tag{3.2c}$$

The rate of change of total mass in R is,

$$\frac{\partial}{\partial t} \iint_R U \, dR \tag{3.3a}$$

and from (3.2c) this is,

$$\frac{\partial}{\partial t}\left(A \, \bar{U}\right) = A \, \frac{\partial}{\partial t} \, \bar{U} \tag{3.3b}$$

3.3.2 Flux

The flow of a quantity has attributes of size *and* direction and is therefore a *vector* quantity which is commonly called the *flux*. The *total* flow through a boundary is a scalar quantity and is also referred to as the (total) flux.

Let $\underline{H}(t, x, y)$ be the flux *density* vector. We are considering the flow of mass in 2D so the $\underline{i}$ and $\underline{j}$ components of $\underline{H}$ have units of kg/s/m. For the 2D linear advection equation (1.1a), $\underline{H} = U \, \underline{v}$ where $\underline{v}$ is the flow velocity. In order to calculate the *total flux* through a boundary at a particular time consider mass flow in 2D as shown in Figure 3.1.

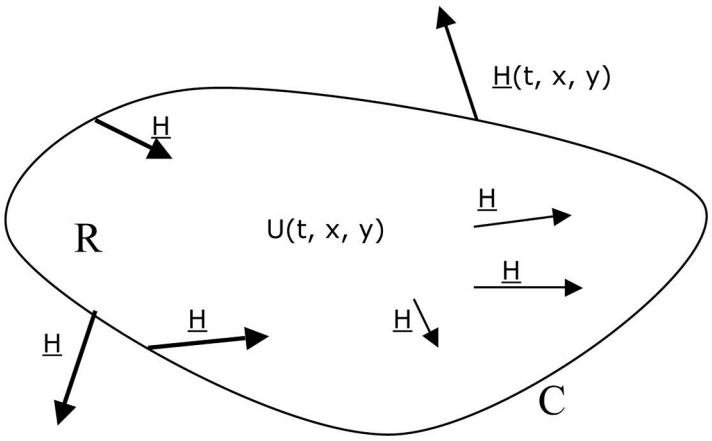

Figure 3.1: Control volume showing flux density $\underline{H}$ and density U.

At a particular time $\underline{H}$ is a *vector* at every point (x, y). Since mass is a conserved quantity the only flow which affects the total mass inside R at a particular time is the flow through its boundary, C, at that time. Hence the only values of $\underline{H}$ which concern us are its values at points on C. At a point on C the vector $\underline{H}$ can point in any direction. The outward flux at point (x, y) on C is the component of $\underline{H}$ *normal* to C. This component is (see Appendix A),

$$\underline{H} \cdot \underline{n} \tag{3.4a}$$

where $\underline{n}$ is the unique *outward* pointing *unit normal* vector to C at (x, y).

3.4 Deriving the Integral Form of the 2D Linear Advection Equation

Approximating $\underline{H}$ by a *constant* value on a small segment of C of length Δs, the *total flux* through the segment is approximately,

$$\underline{H} \cdot \underline{n} \ \Delta s \tag{3.4b}$$

Therefore the *total* flux through C is approximately,

$$\sum_C \underline{H} \cdot \underline{n} \ \Delta s \tag{3.4c}$$

where the sum is over all segments that make up C.

In the limit as $\Delta s \to 0$ the total flux through C is,

$$\oint_C \underline{H} \cdot \underline{n} \; ds \tag{3.4d}$$

where C is traversed in an anticlockwise direction (which accords with our convention that outward flow is positive).

We can now write statement (3.1) mathematically using Equations (3.3b) and (3.4d) to give,

$$\frac{\partial}{\partial t} \iint_R U \; dR = -\oint_C \underline{H} \cdot \underline{n} \; ds \tag{3.5a}$$

Taking the derivative under the integral sign gives,

$$\iint_R \frac{\partial U}{\partial t} \; dR = -\oint_C \underline{H} \cdot \underline{n} \; ds \tag{3.5b}$$

which is Equation (1.4b), an integral form of the 2D linear advection equation. Hence we have derived the 2D linear advection equation by using a control volume approach in conjunction with the conservation of mass.

Notes:

1. (3.5b) does not need U to be spatially differentiable unlike the differential form of the 2D linear advection equation (1.1a).
2. (3.5b) applies to any 2D region so we can apply it directly to a cell in a mesh (in this case in a structured mesh). If we replace the double integral by $A \frac{\partial}{\partial t} \bar{U}$ and discretised this derivative (by the first order forward difference) and express the line integral as a sum over the sides of the cell we generate a general finite volume scheme (1.9) which is,

$$u_k^{n+1} = u_k^n - \frac{\Delta t}{A_k} \sum_{\text{sides}} \underline{H}^n \cdot \underline{s} \tag{3.6}$$

3. From our derivation we see that the 'H dot s' terms in (3.6) represent the total flow through each cell side where $\underline{H}$ is the constant flux density on the cell side and $\underline{s}$ incorporates the normal vector and the side length. (i.e. $\underline{s} = \underline{n}$ multiplied by side length).

3.5 Deriving the Differential Form of the 2D Linear Advection Equation

The differential form of the 2D linear advection equation (1.1a) may be derived by applying the conservation of mass to a rectangular control volume whose sides are parallel to the x and y axes and then shrinking the region down to a single point (x, y) whereupon expressions become the required partial derivatives. We use a more elegant method based upon the integral equation (3.5b).

Rearranging (3.5b) gives,

$$\iint_R \frac{\partial U}{\partial t}\, dR + \oint_C \underline{H} \cdot \underline{n}\, ds = 0 \tag{3.7a}$$

Assuming that the components of $\underline{H}$ are suitably spatially differentiable we can use our version of Green's theorem in reverse (see Appendix B,) to rewrite the line integral in (3.7a) as a double integral to give,

$$\iint_R \frac{\partial U}{\partial t}\, dR + \iint_R \nabla \cdot \underline{H}\, dR = 0 \tag{3.7b}$$

Putting the integrands together gives,

$$\iint_R \left(\frac{\partial U}{\partial t} + \nabla \cdot \underline{H} \right) dR = 0 \tag{3.7c}$$

Since R is an arbitrary region the integrand must be identically zero, hence,

$$\frac{\partial U}{\partial t} + \nabla \cdot \underline{H} = 0 \tag{3.8}$$

In the 2D linear advection equation (1.1a), $\underline{H} = U\,\underline{v}$, where the $\underline{i}$ and $\underline{j}$ components of $\underline{v}$ are constants speeds v_x and v_y. Writing out (3.8) with this definition of $\underline{H}$ gives,

$$\frac{\partial U}{\partial t} + v_x \frac{\partial U}{\partial x} + v_y \frac{\partial U}{\partial y} = 0 \tag{3.9}$$

which is the 2D linear advection equation as required.

Note:

Our derivation did not rely on the *specific* form of $\underline{H}$ until the end. In our derivation flow across C was by advection with constant flow velocity. In reality flow velocity may not be constant and diffusion may also be an important process. In these cases derivation of the integral equations from conservation of mass is exactly the same as previously but with a different expression for $\underline{H}$.

Exercise 3

1. At time zero the total mass in a closed system is 100kg. What is the total mass in the system after a million years? Why?

2. Why isn't the total mass in a lecture theatre constant?

3. At time zero a bucket with a hole in it contains 20kg of water. Water is poured into a bucket at 10 kg/s and water flows out of the hole at 5 kg/s. What is the rate of change of the mass of water in the bucket? What is the mass of water in the bucket after 2 seconds?

4. A 2D (mass) flux density is given by the vector field, $\underline{H} = K\,\underline{i}$ where K is a constant measured in kg/s/m.

a) draw the vector field $\underline{H}$

b) Calculate the total mass flow across a line of length L which is,

1. parallel to the x axis,
2. parallel to the y axis,
3. at 45 degrees to the x axis.

5. The density, U, of material over the xy-plane is constant at 10 kg/m^2. Calculate the total mass of material within a rectangular 10m by 5m region. Hence verify (3.2a).

6. The mean value, $\bar{U}$, of the density of material over a rectangular 2m by 5m region, R, of the xy-plane is 20 kg/m^2. Calculate the total mass of material within R. Hence verify (3.2c).

7. Write down the units of the variables in Equations (3.2a, b, c) and show that they are consistent. Write down the units of rate of change of total mass.

8. Check that the units of (3.4b) (and hence (3.4c,d) and (3.5)) are correct.

9. Derive (3.6) from (3.5b).

10. Show that (3.9) follows from (3.8).

4 Further Finite Volume Schemes

4.1 Introduction

In Chapter 1 we saw that the general finite volume scheme (using first order forward differencing in time) on a structured mesh is,

$$J_{i,j}^{n+1} = u_{i,j}^{n} - \frac{\Delta t}{A_{i,j}} \left(\underline{H}_{i+1/2,j}^{n} \cdot \underline{S}_{i+1/2,j} + \underline{H}_{i,j+1/2}^{n} \cdot \underline{S}_{i,j+1/2} + \underline{H}_{i-1/2,j}^{n} \cdot \underline{S}_{i-1/2,j} + \underline{H}_{i,j-1/2}^{n} \cdot \underline{S}_{i,j-1/2} \right) \quad (4.1)$$

So far we have presented only one finite volume scheme, namely the FOU scheme of Chapter 2. In this chapter we derive some more schemes for the linear advection equation (although the principles apply to any suitable equation). We know that a particular scheme is determined by how the flux density, $\underline{H}=\underline{H}(U)$, is estimated at cell interfaces. This in turn may depend on how U is estimated at cell interfaces. In the FOU scheme the required U value at a cell interface was estimated by using the neighbouring upwind cell centre value of U. In effect we assumed that U was piecewise constant over cells. This gave potentially two U values at each cell interface and we simply chose the upwind U value at the interface as this takes in to account the direction of propagation of the solution. In reality U varies over the whole 2D computational domain so it seems reasonable to estimate interface U values by some sort of interpolation based on known neighbouring cell centre U values. The next sections present linear and quadratic interpolation on structured meshes. As we shall see linear interpolation requires two points and quadratic three points. It should be noted that these schemes are derived without any attempt at stability analysis so there is no guarantee that they will be useful; stable time steps, if any, should be determined by experiment. However, on a Cartesian mesh a FVS should reduce to the corresponding finite difference scheme which is amenable to a formal truncation error analysis and von Neumann stability analysis which was presented in the related textbook, 'Introductory Finite Difference Methods for PDEs'.

4.2 Linear Interpolation

In the following we estimate U at cell interface (i+1/2, j). Estimates for the other cell interfaces follow in an exactly similar manner. Although interface (i+1/2, j) is surrounded by several known cell centre U values it is reasonable to consider only the two neighbouring cells in the i direction as this is the direction in which fluxes are to be projected at this interface. Consequently we will drop the j subscripts. Figure 4.1 shows interface i+1/2 and its two neighbouring cells in the i direction.

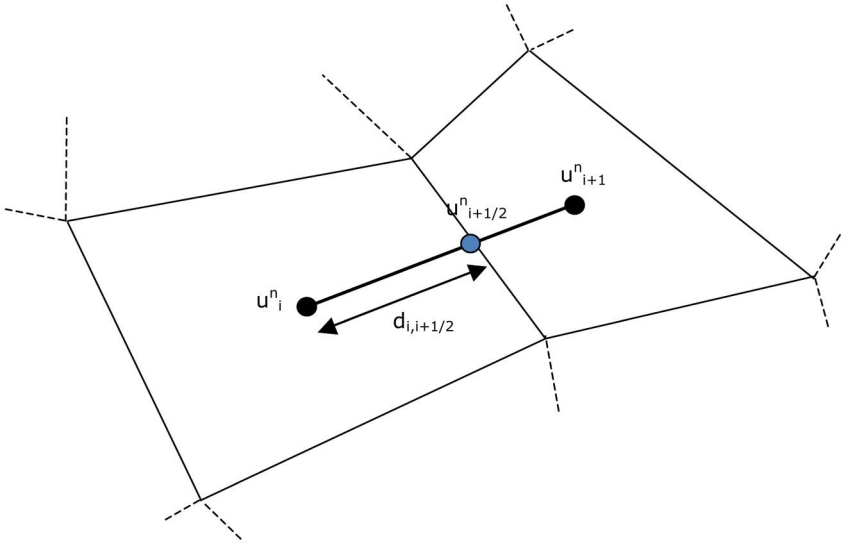

Figure 4.1: Interface i+1/2 and neighbouring cell centre values for linear interpolation

We assume that U varies linearly along the line joining the centres of cells i and i+1and hence we can estimate the interface U value by simple linear interpolation giving,

$$u^n_{i+1/2} = u^n_i + d_{i,i+1/2}\ g_{i,i+1} \tag{4.2a}$$

where, $d_{i,i+1/2}$ is the distance from the centre of cell i to interface i+1/2 measured along the line joining cell centres i and i+1 and $g_{i,i+1}$ is the gradient of U between cell centres i and i+1 which is given by,

$$g_{i,i+1} = \frac{u_{i+1} - u_i}{d_{i,i+1}} \tag{4.2b}$$

where $d_{i,i+1}$ is the distance between the centres of cells i and i+1.

Notes:

1. A FVS based on this procedure will require ghost values at the left and the right boundaries.
2. This procedure is not the only or best possibility for linear interpolation. Since fluxes are projected normal to cell interfaces it makes more sense to calculate the shortest distance from the centre of cell i to cell interface i+1/2 since the resulting direction is normal to the cell interface. The gradient given by (4.2b) can then be regarded as a vector and its component in the direction normal to interface i+1/2 calculated for use in (4.2a). For simplicity we won't do this.
3. A modified version of linear interpolation is used in modern high resolution schemes.

Equations (4.2b) and (4.2a) specify U and therefore $\underline{H}$ at cell interfaces and therefore (4.1) gives a new FVS. It is instructive to look at this scheme on a 1D Cartesian mesh.

4.2.1 Linear Interpolation FVS on a Cartesian Mesh

Restricting attention to a 1D uniform Cartesian mesh in the x direction and writing

$\underline{H} = F\,\underline{i}$, we have seen (Chapter 2) that (4.1) reduces to,

$$u_i^{n+1} = u_i^n - \Delta t \left(\frac{F_{i+1/2}^n - F_{i-1/2}^n}{\Delta x} \right) \tag{4.3}$$

For the linear advection equation, $F = v_x U$, hence (4.3) becomes,

$$u_i^{n+1} = u_i^n - \frac{v_x \Delta t}{\Delta x} \left(u_{i+1/2}^n - u_{i-1/2}^n \right) \tag{4.4}$$

We wish to estimate the two interface values in the bracket on the right hand side of (4.4) using linear interpolation via equations (4.2a,b). On our uniform 1D Cartesian mesh, $d_{i,i+1} = \Delta x$ and $d_{i,i+1/2} = \Delta x/2$ so (4.2b) gives,

$$g_{i,i+1} = \frac{u_{i+1} - u_i}{\Delta x} \tag{4.5}$$

and (4.2a) gives,

$$u_{i+1/2}^n = u_i^n + (\Delta x/2) \frac{u_{i+1} - u_i}{\Delta x} \tag{4.6a}$$

which simplifies to,

$$u^n_{i+1/2} = \frac{u_{i+1} + u_i}{2} \tag{4.6b}$$

Similarly,

$$u^n_{i-1/2} = \frac{u_i + u_{i-1}}{2} \tag{4.6c}$$

i.e. interface values of U are estimated by the average of the two neighbouring values.

Putting (4.6b,c) into the FVS (4.4) gives,

$$u^{n+1}_i = u^n_i - \frac{v_x \Delta t}{\Delta x}\left(\frac{u^n_{i+1} + u^n_i}{2} - \frac{u^n_i + u^n_{i-1}}{2}\right) \tag{4.7a}$$

which simplifies to,

$$u^{n+1}_i = u^n_i - v_x \Delta t\left(\frac{u^n_{i+1} - u^n_{i-1}}{2\Delta x}\right) \tag{4.7b}$$

The FVS (4.7b) is identical to the classical forward-in-time-centred-in-space (FTCS) finite difference scheme which is known to be unconditionally unstable! The finite difference FTCS scheme can be stabilised by replacing the first term on the right by the average of its two neighbouring values to give the Lax-Friedrichs scheme. The finite volume version of the Lax-Friedrichs scheme in 2D on a structured mesh is therefore,

$$u^{n+1}_{i,j} = \frac{u^n_{i-1,j} + u^n_{i+1,j} + u^n_{i,j-1} + u^n_{i,j+1}}{4} - \frac{\Delta t}{A_{i,j}} \sum_{\text{sides}} \underline{H}^n \cdot \underline{s} \tag{4.8}$$

where the sum is taken over the four sides of cell (i, j) and $\underline{H}$ at cell interfaces is estimated by linear interpolation of U as described.

The 1D Lax-Friedrichs FVS for the linear advection equation is coded in file FVlinearadvectionLAXF1D.

4.3 Quadratic Interpolation

The previous method of estimating interface fluxes assumed a linear distribution of U and fitted a unique straight line through two neighbouring cell centre U values which enabled the interface value between them to be estimated. This has an obvious generalisation: assume a quadratic distribution of U and fit a quadratic through three neighbouring U values. As before we consider only cells in the i direction. To estimate U at cell interface i+1/2 we use neighbouring cell centre values in cells i and i+1. To produce an interpolating quadratic we need a third U value and there are two candidates: namely the U values at the centres of cells i-1 and i+2. It seems reasonable to select the upwind U value since this is usually where information is flowing from. So if the flow is in the positive i direction the third U value is at the centre of cell i-1. It is then a simple matter to fit a quadratic and use it to estimate U at cell interface i+1/2.

Notes:

1. At the boundary of the computational domain a FVS based on this procedure requires two ghost values in the upwind direction and one ghost value in the downwind direction.
2. Using more points to estimate interface fluxes may lead to higher scheme accuracy although it can be shown that oscillations can occur in the solution.
3. As in the previous linear interpolation there is no guarantee that direct quadratic interpolation produces a stable scheme.
4. The above procedure assumes that the three cell centres lie on a straight line (so that the interpolating quadratic is a unique function of a single variable) but this is not true for a general 2D mesh so that this method has its limitations.
5. This procedure is the basis of the well known QUICK scheme.

4.3.1 Quadratic Interpolation FVS on a Cartesian Mesh

On a uniform 1D Cartesian mesh in the x direction it can be shown that the estimated value of $u_{i+1/2}$ when flow is in the positive i direction using quadratic interpolation is,

$$u_{i+1/2} = (-u_{i-1}+6u_i+3u_{i+1})/8 \tag{4.9}$$

Substitution in to (4.4) and simplifying gives the following quadratic FVS for the linear advection equation,

$$u_i^{n+1} = u_i^n - \frac{v_x \Delta t}{8\Delta x}\left(u_{i-2}^n - 7u_{i-1}^n + 3u_i^n + 3u_{i+1}^n\right) \tag{4.10}$$

This scheme is coded in file FVlinearadvectionQUAD1D.

4.4 Converting from Finite Difference to Finite Volume

We have seen that on a uniform Cartesian mesh a finite volume scheme (FVS) reduces to an equivalent finite difference scheme (FDS). A natural question to ask is, 'given a finite difference scheme can we infer an equivalent finite volume scheme?'. The following procedure generates a FVS which reduces back to the parent FDS on a uniform Cartesian mesh. Using this approach we may be able construct FV schemes directly without considering how to estimate interface fluxes.

The steps are:

Step 1. Starting in 1D in the x direction write the FDS with flux components.

Step 2. Write flux components as dot products of flux (density) vectors and side vectors.

Step 3. Re-index flux vectors to match side vector indices and introduce cell areas.

Step 4. Generalize to 2D (or 3D) in the obvious way.

(Step 5. Check that the scheme reduces back to the FDS on a uniform Cartesian mesh.)

In the following example we convert the FOU FDS for the linear advection equation to the equivalent FVS.

Step 1: In 1D (for $v_x > 0$) the FOU FDS is:

$$u_i^{n+1} = u_i^n - \frac{v_x \Delta t}{\Delta x}\left(u_i^n - u_{i-1}^n\right) \tag{4.11}$$

For the 1D linear advection equation the flux density vector is $\underline{H} = F\,\underline{i}$ where $F = U\,v_x$. Writing (4.11) using flux components gives,

$$\begin{aligned} u_i^{n+1} &= u_i^n - \frac{\Delta t}{\Delta x}\left(v_x u_i^n - v_x u_{i-1}^n\right) \\ &= u_i^n - \frac{\Delta t}{\Delta x}\left(F_i^n - F_{i-1}^n\right) \end{aligned} \tag{4.12a}$$

Step 2: Left and right side vectors for cell i are $\underline{s}_{i-1/2} = -\Delta y\,\underline{i}$ and $\underline{s}_{i+1/2} = \Delta y\,\underline{i}$ respectively so (4.12a) becomes,

$$\begin{aligned} u_i^{n+1} &= u_i^n - \frac{\Delta t}{\Delta x}\frac{1}{\Delta y}\left((F_i^n \underline{i})\cdot(\Delta y \underline{i}) + (F_{i-1}^n \underline{i})\cdot(-\Delta y \underline{i})\right) \\ &= u_i^n - \frac{\Delta t}{\Delta x \Delta y}\left(\underline{H}_i^n \cdot \underline{s}_{i+1/2} + \underline{H}_{i-1}^n \cdot \underline{s}_{i-1/2}\right) \end{aligned} \tag{4.12b}$$

Step 3: Write flux vectors at appropriate cell interfaces and replace $\Delta x\,\Delta y$ by cell area A_i to give,

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{A_i}\left(\underline{H}_{i+1/2}^n \cdot \underline{s}_{i+1/2} + \underline{H}_{i-1/2}^n \cdot \underline{s}_{i-1/2}\right) \tag{4.12c}$$

where the interface fluxes are given by their upwind values (this defines the FOU scheme).

Step 4: In 2D cells are indexed by two subscripts, $\underline{H}$ has $\underline{i}$ and $\underline{j}$ components and each cell has four interfaces (which, in the FOU scheme, are found from the upwind cell centre values), giving the 2D FOU FVS,

$$u_{i,j}^{n+1} = u_{i,j}^n - \frac{\Delta t}{A_{i,j}}\left(\underline{H}_{i+1/2,j}^n \cdot \underline{s}_{i+1/2,j} + \underline{H}_{i-1/2,j}^n \cdot \underline{s}_{i-1/2,j} + \underline{H}_{i,j+1/2}^n \cdot \underline{s}_{i,j+1/2} + \underline{H}_{i,j-1/2}^n \cdot \underline{s}_{i,j-1/2}\right) \tag{4.12d}$$

It is a simple matter to check that this reduces to the FOU FDS scheme on a uniform 2D Cartesian mesh.

Exercise 4

1. Show that (4.2a) is correct.

2. Show that (4.6b) follows from (4.6a).

3. Show that (4.7b) follows from (4.7a).

4. Write down the linear interpolation FVS for the 2D linear advection equation on a general 2D mesh and show that it becomes the FTCS scheme on a uniform 1D Cartesian mesh.

5. Write down the Lax-Friedrichs FVS on a general pseudo-1D mesh.

6. Derive (4.9).

7. Show that (4.10) follows from (4.9).

8. Figure 4.2 is a small (test) FV mesh for the 2D linear advection equation with flow speeds of 3m/s in both x and y directions. Initial U values are given at cell centres. Boundary conditions are zero gradient everywhere at all times.

a) Obtain the coordinates of the centre of cell (2, 1) using simple averaging of vertices.
b) Using linear interpolation estimate the initial values of U at the interfaces of cell (2,1).

9) Use a single iteration of the Lax-Friedrichs scheme to estimate the value of U in cell (2, 1) at time 0.2 seconds.

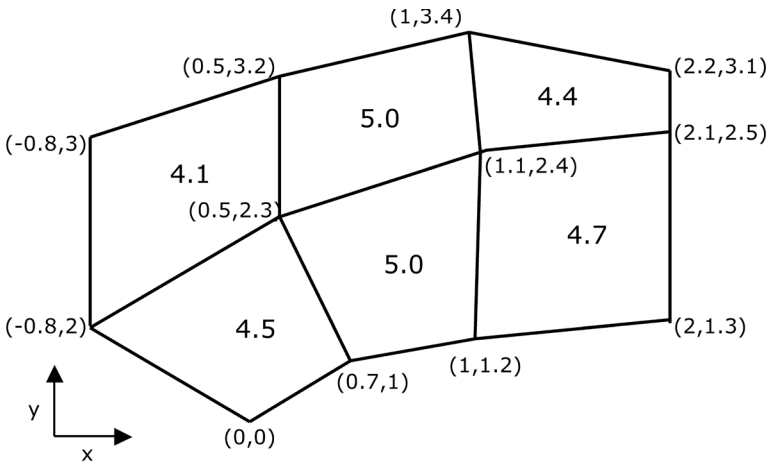

Figure 4.2: Test 2D mesh with initial cell centre u values.

10. 'FVlinearadvectionLAXF1D' is a code to implement the finite volume Lax-Friedrichs scheme to solve the 1D linear advection equation. This code is written in a structured way so that each part can be verified easily.

a) Verify the code.
b) Implement the code on a uniform 1D Cartesian mesh with a Gaussian initial profile based at the centre of the mesh. Use positive and negative flow speeds.
c) Find the maximum allowable time step.
d) Comment critically on your results in the light of the exact solution.
e) Run the code on a pseudo-1D uniform mesh parallel to the line y = x. Set $v_x = v_y = 2^{1/2}$. Plot your mesh. Comment on your results.

11) Adapt 'FVlinearadvectionLAXF1D' to make it the 1D FTCS scheme and run it to show that it is unconditionally unstable.

12) 'FVlinearadvectionQUAD1D' is a code to implement the finite volume quadratic interpolation scheme to solve the 1D linear advection equation. This code is written in a structured way so that each part can be verified easily.

a) Verify that the interpolation is correctly coded for a uniform Cartesian mesh.
b) Implement the code on a uniform Cartesian mesh with a Gaussian initial profile based at the centre of the mesh. Use both positive and negative flow speeds.
c) Determine whether or not the solver is stable and find the maximum allowable time step if possible.
d) Comment critically on your results in the light of the exact solution.

13) Adapt 'FVlinearadvectionFOU2D' to a 2D Lax-Friedrichs solver for the 2D linear advection equations. Verify your code and find the maximum allowable time step.

14) Verify that (4.12d) reduces to the 2D FOU FDS on a uniform Cartesian mesh.

15) Starting with the Lax-Friedrichs FDS for the 1D linear advection equation obtain the 1D and 2D Lax-Friedrichs FVS for the linear advection equation.

See the website for codes.

5 Systems of Equations

5.1 Introduction

In previous chapters we implemented the FVM for a *single* PDE (which was derived from a conservation law). Many phenomena of interest to engineers and scientists are described by *systems* of coupled PDEs (which often arise from several conservation laws). A famous example are the Navier-Stokes equations which, together with the Continuity Equation, form a system of coupled PDEs describing fluid flow. Solving these equations is beyond the scope of an introductory text so we apply the FVM to a simpler system of PDEs which are nonetheless very useful.

5.2 The Shallow Water Equations

After much simplification the Navier-Stokes and Continuity equations can be reduced to the Shallow Water Equations (SWE) which may be expressed in 1D or 2D. The SWE can be regarded as simplified model of water flow and are used by engineers to simulate many phenomena of practical interest including river flooding, tsunami propagation and dam break flows. One key simplification in deriving the SWE is that the flow velocity in the water column is depth-averaged so that the equations model situations where the water velocity does not vary much with depth. We present the SWE in both 1D and 2D since both systems of equations are used extensively by the engineering community.

5.2.1 2D SWE

The 2D SWE form a system of three coupled non-linear hyperbolic PDEs with independent variables t (time) and x, y (*horizontal* space). In differential form (where the partial derivatives are applied to each row) the 2D SWE can be written as the matrix PDE,

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = Q \tag{5.1}$$

where,

$$U = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix} = \begin{bmatrix} \phi \\ \phi v_x \\ \phi v_y \end{bmatrix}, \tag{5.2a}$$

$$F = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} = \begin{bmatrix} \phi v_x \\ \phi v_x^2 + \phi^2/2 \\ \phi v_x v_y \end{bmatrix}, \quad G = \begin{bmatrix} G_1 \\ G_2 \\ G_3 \end{bmatrix} = \begin{bmatrix} \phi v_y \\ \phi v_x v_y \\ \phi v_y^2 + \phi^2/2 \end{bmatrix} \tag{5.2b}$$

$$Q = \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \end{bmatrix} = \begin{bmatrix} 0 \\ g\ \phi\ b_x \\ g\ \phi\ b_y \end{bmatrix}. \tag{5.2c}$$

$\phi = \phi(t,x,y)$ is called the geopotential and $\phi = g\,h$ where $h = h(t,x,y)$ is the water depth and g is the acceleration due to gravity ($g = 9.81 m/s^2$). $v_x = v_x(t,x,y)$ and $v_y = v_y(t,x,y)$ are the water speeds in x and y directions respectively. U is the column matrix of dependent variables, F = F(U) and G = G(U) are column matrices of fluxes in the x and y directions respectively and Q is a column matrix of source terms which here only includes bathymetric terms where b_x and b_y are bed slopes in the x and y directions (measured positive *downwards*). Note that in some texts U, F, G and Q are referred to as vectors and (5.1) is a vector PDE.

Solving the 2D SWE gives the three components of U (effectively momentum in x and y directions and mass) from which the water depth and flow speeds can be found at times t and points (x, y).

5.2.2 1D SWE

The 1D SWE are an obvious reduction of the 2D SWE to 1D and are used to model thin straight channels. They form a system of two coupled non-linear hyperbolic PDEs with independent variables t (time) and x (*horizontal* space). In differential form (where the partial derivatives are applied to each row) the 1D SWE can be written as the matrix PDE,

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = Q \tag{5.3}$$

where,

$$U = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} \phi \\ \phi v_x \end{bmatrix}, \tag{5.4a}$$

$$F = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = \begin{bmatrix} \phi v_x \\ \phi v_x^2 + \phi^2 / 2 \end{bmatrix}, \tag{5.4b}$$

$$Q = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} = \begin{bmatrix} 0 \\ g \phi b_x \end{bmatrix}. \tag{5.4c}$$

Variables are defined as in the 2D SWE with the obvious reductions to 1D. Solving the 1D SWE gives the two components of U (effectively mass and momentum in x direction) from which the water depth and flow speed can be found at required times t and points x.

Except for very special situations the SWE do not have analytical solutions. We use the FVM to find approximate solutions. It should be noted that an in-depth treatment of the SWE requires the study of such concepts as supercritical and subcritical flow and Riemann invariants which we leave to a more advanced book to follow. The following in a purely mathematical treatment to show how the FVM applies to a system of PDEs.

5.3 General FVS for the SWE

As we have seen previously, the FVM applies to PDEs that can be written in finite volume form. We will focus on the 2D SWE since the FVM for the 1D SWE is essentially the same. We observe that (5.1) is very similar to (1.1a) so we expect that it will be possible to write (5.1) in finite volume form which we now do.

Let $\underline{H} = F\,\underline{i} + G\,\underline{j}$ and $\underline{v} = v_x\,\underline{i} + v_y\,\underline{j}$ then (5.1) becomes,

$$\frac{\partial U}{\partial t} + \nabla \cdot \underline{H} = Q \tag{5.5a}$$

where,

$$\underline{H} = \begin{bmatrix} \underline{H}_1 \\ \underline{H}_2 \\ \underline{H}_3 \end{bmatrix} = \begin{bmatrix} \phi \underline{v} \\ \phi v_x \underline{v} + 0.5\phi^2 \underline{i} \\ \phi v_y \underline{v} + 0.5\phi^2 \underline{j} \end{bmatrix} \tag{5.5b}$$

$\underline{H} = \underline{H}(U)$ is the flux density, $\underline{v}$ is the flow velocity and the differential operators are applied to each row of U and $\underline{H}$. (5.5a) represents the SWE written in finite volume form as required. The theory and operations of Chapters 1 and 2 are applied to each row of (5.5a) to give,

$$\frac{\partial U}{\partial t} = -\frac{1}{A}\oint_C \underline{H}\cdot\underline{n}\ ds + Q \tag{5.6}$$

where, as before, U and Q now represent averaged quantities over an arbitrary region of area A in the xy-plane.

Discretising each row of (5.6) in the same way as in Chapters 1 and 2 and rearranging gives,

$$U_k^{n+1} = U_k^n - \frac{\Delta t}{A_k}\left(\sum_{\text{sides}} \underline{H}^n\cdot\underline{s}\right) + \Delta t\ Q_k^n \tag{5.7}$$

Notes:

1. The matrix difference equation (5.7) may be regarded as a general *explicit* FVS for the system of equations in (5.1).
2. Derivation of (5.7) for the system (5.1) is the same as for a single equation written in finite volume form - we simply repeat the steps for each row using the *same* time step Δt.
3. Implementation of (5.7) is done row by row.
4. As in the single equation case, a particular FVS based on (5.7) is constructed by *estimating* the interface fluxes for each row of $\underline{H}$.
5. Since $\underline{H} = \underline{H}(U)$, interface flux estimation may be done by estimating U at cell interfaces or estimating $\underline{H}$ at cell interfaces directly, in either case using some form of extrapolation.

6. Instead of (as in the single equation case) using the corresponding lower case letters to indicate approximate values from the FVS we retain upper case letters for U and Q in (5.7) to emphasise that they are column matrices. This slight change in notation is not a problem if we adopt the convention that any upper case variable having subscripts and superscripts is an approximation to the corresponding exact solution (e.g. U_k^n is an approximation to the exact solution $U(n\Delta t, x_k, y_k)$ where (x_k, y_k) is the centre of cell k).
7. Since the derivation of a FVS for a system of equations that can be written in the form (5.5a) is essentially the same as for a single equation we should expect that any FVS for a single equation will be the same for such a system when we interpret the scheme row by row.

5.4 FVS for the 2D SWE on a Structured Mesh

In the following treatment we write (5.7) on a *structured* mesh whose cells are indexed by (i, j) with the subscript ½ to denote cell interfaces in the usual way. For simplicity we take Q = 0 and (5.7) becomes,

$$U_{i,j}^{n+1} = U_{i,j}^{n} - \frac{\Delta t}{A_{i\,i}}\left(\underline{H}_{i+1/2,j}^{n} \cdot \underline{S}_{i+1/2,j} + \underline{H}_{i,j+1/2}^{n} \cdot \underline{S}_{i,j+1/2} + \underline{H}_{i-1/2,j}^{n} \cdot \underline{S}_{i-1/2,j} + \underline{H}_{i,j-1/2}^{n} \cdot \underline{S}_{i,j-1/2}\right) \tag{5.8}$$

A particular FVS is determined from the estimates of the interface fluxes and it is assumed that the same estimation technique is used for each row of (5.8).

5.4.1 First Order Upwind (FOU) Scheme for the SWE

As previously stated the FOU scheme simply estimates interface fluxes based on the nearest upwind cell centre U values. Therefore at interface (i+1/2, j),

$$\underline{H}_{i+1/2,j}^{n} = \begin{cases} \underline{H}(U_{i,j}^{n}), & \underline{v}_{i,j}^{n} \cdot \underline{S}_{i+1/2,j} \geq 0, \\ \underline{H}(U_{i+1,j}^{n}), & \text{otherwise.} \end{cases} \tag{5.9a}$$

And similarly at interface (i, j+1/2),

$$\underline{H}_{i,j+1/2}^{n} = \begin{cases} \underline{H}(U_{i,j}^{n}), & \underline{v}_{i,j}^{n} \cdot \underline{S}_{i,j+1/2} \geq 0, \\ \underline{H}(U_{i,j+1}^{n}), & \text{otherwise.} \end{cases} \tag{5.9b}$$

Notes:

1. Unlike the linear advection equation, flow velocity in the SWE varies both spatially and temporally so at each time step the direction of flow at each cell interface must be found in order to determine the upwind direction. This is done by calculating the sign of the component of flow velocity normal to a cell side via the dot product of the flow velocity vector and the corresponding side vector.
2. U values from the upwind cell centres are used to determine the interface fluxes by first finding ϕ, v_x and v_y from the rows of U (e.g. $v_x=U_2/U_1$).
3. The same time step is used for each row of (5.8).

5.4.2 Lax-Friedrichs Scheme for the SWE

In Chapter 4 we met the Lax-Friedrichs scheme for the 2D linear advection equation on a structured mesh. It is,

$$u_{i,j}^{n+1} = \frac{u_{i-1,j}^n + u_{i+1,j}^n + u_{i,j-1}^n + u_{i,j+1}^n}{4} - \frac{\Delta t}{A_{i,j}} \sum_{\text{sides}} \underline{H}^n \cdot \underline{s} \tag{5.10}$$

where $\underline{H} = \underline{v}\, u$ and interface fluxes are estimated by linear interpolation. Since the SWE and the linear advection equation have the same finite volume form we expect their equivalent FVS to have the same form. Accordingly the Lax-Friedrichs scheme for the SWE (with Q = 0) is,

$$U_{i,j}^{n+1} = \frac{U_{i-1,j}^n + U_{i+1,j}^n + U_{i,j-1}^n + U_{i,j+1}^n}{4} - \frac{\Delta t}{A_{i,j}} \sum_{\text{sides}} \underline{H}^n \cdot \underline{s} \tag{5.11}$$

where U and $\underline{H}$ are given by (5.2a) and (5.5b) respectively and interface fluxes are estimated by linear interpolation as described in Chapter 4.

Notes:

1. As mentioned previously, a FVS is derived from the finite volume form which is independent of the particular PDE or system of PDEs. Hence it is no surprise that a FVS for a single PDE has the same form as its equivalent for a system of PDEs as is the case with the FOU and Lax-Friedrichs schemes.
2. The previous note implies that any FV code to solve a single PDE should be convertible easily to a system of PDEs although there is no guarantee that it will work!
3. (5.11) is an explicit scheme so the time step will be constrained by stability considerations which we now address.

5.5 Heuristic Time Step for a 2D SWE FVS

Although equivalent FVS for the linear advection equation and SWE look the same, determination of a stable time step for SWE FVS is a little more complicated. This is because information propagates at different speeds for each equation in the system and the speeds of propagation vary in time and space. The interested reader should refer to the companion textbook 'Introductory Finite Difference Methods for PDEs' for a simple analysis of propagation speeds for the 1D SWE.

In the following we adopt the same heuristic approach that we used to determine the time step for the linear advection equation (see Chapter 2). This gave rise to the following formula for the maximum stable time step,

$$\Delta t \le \min_{i,j} \left\{ \frac{A_{i,j}}{\max\left\{ \left| \underline{v} \cdot \underline{s}_{i+1/2,j} \right|, \left| \underline{v} \cdot \underline{s}_{i,j+1/2} \right| \right\}} \right\} \tag{5.12}$$

For the SWE this becomes,

$$\Delta t \le \min_{i,j} \left\{ \frac{A_{i,j}}{\max\left\{ \left| \underline{v}^n_{i,j} \cdot \underline{s}_{i+1/2,j} \right| + \sqrt{\phi^n_{i,j}} \left| \underline{s}_{i+1/2,j} \right|, \left| \underline{v}^n_{i,j} \cdot \underline{s}_{i,j+1/2} \right| + \sqrt{\phi^n_{i,j}} \left| \underline{s}_{i,j+1/2} \right| \right\}} \right\} \tag{5.13}$$

Notes:

1. Comparison of (5.12) and (5.13) indicates that the maximum propagation speed of information in the 2D SWE in cell (i, j) at time level n is $\left| \underline{v}^n_{i,j} \right| + \sqrt{\phi^n_{i,j}}$ (there are actually 3 speeds, one for each equation).
2. Since $\underline{v}$ and ϕ vary in time Δt must be recalculated before each iteration of a FVS using (5.13).
3. The heuristic analysis behind (5.13) takes no account of the actual FVS so must be regarded as a rough guide for determining the maximum stable time step. (5.13) can be tuned to a particular FVS by multiplying its right hand side by a positive constant which can be determined by numerical experiment.

Exercise 5

1. Explain why, in the SWE, water depth and velocity do not depend on the vertical variable z.

2. Show that the 1D SWE follow from the 2D SWE.

3. Derive (5.5a,b) from (5.1).

4. For the 2D SWE, $U = [18.4, 4.1, 2.2]^T$ at a particular point in time and space. Calculate the water depth and flow speeds in the x and y directions at this point. Also calculate the fluxes $\underline{H}$.

5. Express (5.8) on a regular Cartesian mesh.

6. Assuming positive flow speeds in x and y directions, write down the FOU scheme for the 2D SWE on a regular Cartesian mesh.

7. Repeat Q6 for the Lax-Friedrichs scheme.

8. Write down the heuristic time step inequality for the 1D SWE on a regular Cartesian mesh.

Appendix A Review of Vectors

A.1 Introduction

The Finite Volume Method (FVM) depends on being able to convert surface integrals to line integrals when working in 2D (or volume integrals into surface integrals in 3D) and to calculate flows normal to cell edges in 2D (or surfaces in 3D) whilst retaining the standard Cartesian coordinate system. These operations are simply and elegantly described by vectors which will now be reviewed with a focus on 2D although results generalise easily to 3D. It is assumed that the reader has encountered vectors previously.

A.2 Review of Vectors

A.2.1 Basics

In 2D any vector $\underline{s}$ may be written *uniquely* as,

$$\underline{s} = a\,\underline{i} + b\,\underline{j} \tag{A.1}$$

where $\underline{i}$ and $\underline{j}$ are the usual Cartesian basis vectors and a and b are called the *components* of $\underline{s}$ (in the $\underline{i}$ and $\underline{j}$ directions).

The length (modulus) of $\underline{s}$ is a *scalar* written, $|\underline{s}|$ and,

$$|\underline{s}| = \sqrt{a^2 + b^2} \tag{A.2}$$

A.2.2 Unit Vectors

A vector whose length is 1 is called a *unit vector*. $\underline{i}$ and $\underline{j}$ are unit vectors. Every non-zero vector, $\underline{s}$, has a *unique* unit vector, $\hat{\underline{s}}$ pointing in the same direction, this is,

$$\hat{\underline{s}} = \frac{\underline{s}}{|\underline{s}|} \tag{A.3a}$$

(A.3a) implies that every non-zero vector can be expressed as the product of its length and its unique unit vector, i.e.

$$\underline{s} = |\underline{s}|\ \hat{\underline{s}} \tag{A.3b}$$

A.2.3 Dot Product

The dot (or scalar) product of two vectors, $\underline{v}$ and $\underline{w}$ is a scalar quantity expressed and defined by,

$$\underline{v} \cdot \underline{w} = |\underline{v}|\ |\underline{w}|\ \cos(\theta) \tag{A.4}$$

where θ is the angle between $\underline{v}$ and $\underline{w}$.

The dot product is a very useful operation for the following reasons:

DP1) Two non-zero vectors $\underline{v}$ and $\underline{w}$ are *normal* to each other (i.e. at right angles) if and only if their dot product is zero.

DP2) The component of $\underline{v}$ in the *direction* given by the unit vector $\hat{\underline{s}}$ is

$$\underline{v} \cdot \hat{\underline{s}} \tag{A.5}$$

In the FVM this idea is used to find flux components normal to cell sides.

The dot product may be calculated easily when vectors are expressed with respect to the standard Cartesian basis. Let, $\underline{v} = a\,\underline{i} + b\,\underline{j}$ and $\underline{w} = c\,\underline{i} + d\,\underline{j}$,

then,

$$\underline{v} \cdot \underline{w} = ac + bd \tag{A.6}$$

A.2.4 Vector Between 2 Points

The vector *starting* at point P(a, b) and *ending* at point Q(c, d) is,

$$\underline{PQ} = (c - a)\,\underline{i} + (d - b)\,\underline{j} \tag{A.7}$$

This is illustrated in Figure A.1. (A.7) is a consequence of the well known triangle law of addition for vectors.

Figure A.1: Vector between two points.

In the FVM this idea is used for representing cell sides as vectors.

A.2.5 Cross Product

Although we will be working in 2D we will make use of the *cross product* of 2 vectors which is an operation in 3D. Let, $\underline{v} = a\,\underline{i} + b\,\underline{j} + c\,\underline{k}$ and $\underline{w} = d\,\underline{i} + e\,\underline{j} + f\,\underline{k}$ (where $\underline{k}$ is the usual unit vector pointing in the z direction). The cross product of two vectors, $\underline{v}$ and $\underline{w}$ is vector quantity which may be calculated from a formal determinant by,

$$\underline{v} \times \underline{w} = \begin{vmatrix} \underline{i} & \underline{j} & \underline{k} \\ a & b & c \\ d & e & f \end{vmatrix} \tag{A.8}$$

The vector product is *normal* to both $\underline{v}$ and $\underline{w}$ and $\underline{v}$, $\underline{w}$ and $\underline{v}\times\underline{w}$ form a right handed system.

Note that we may take the cross product of 2, 2D vectors by regarding them as being in 3D with zero $\underline{k}$ component. In the FVM the cross product is used to find cell areas.

A.3 Side Vectors

The FVM requires the calculation of the total flux (flow) *out* of cells so at every point on a cell side we must find the unique *outward pointing* unit normal vector, $\underline{\hat{n}}$. From (A.5) the component of the flux vector, $\underline{H}$, flowing out of a cell at any point on its side is $\underline{H} \cdot \underline{\hat{n}}$. In 2D, cell sides are *straight lines* so for each side there is a *single* outward pointing normal vector. In the FVM, $\underline{H}$ is taken to be constant on a cell side so the *total* flux flowing out of a cell through a single side of length L is,

$$\underline{H} \cdot \underline{s} \tag{A.9a}$$

where,

$$\underline{s} = L\,\underline{\hat{n}} \tag{A.9b}$$

$\underline{s}$ is called a *side vector*. Figure A.2 illustrates a side vector for a cell side PQ. Let $\underline{PQ} = a\underline{i} + b\underline{j}$. By considering $\underline{PQ}$ to be a vector in 3D with zero $\underline{k}$ component,

$$\underline{s} = \underline{PQ} \times \underline{k} = \begin{vmatrix} \underline{i} & \underline{j} & \underline{k} \\ a & b & 0 \\ 0 & 0 & 1 \end{vmatrix} = b\,\underline{i} - a\,\underline{j} \tag{A.10}$$

By the properties of the cross product this formula ensures that $\underline{s}$ always points to the *right* relative to the direction of $\underline{PQ}$ which ensures that, as the perimeter of a cell is traversed in an anti-clockwise direction, $\underline{s}$ always points *outwards* as required.

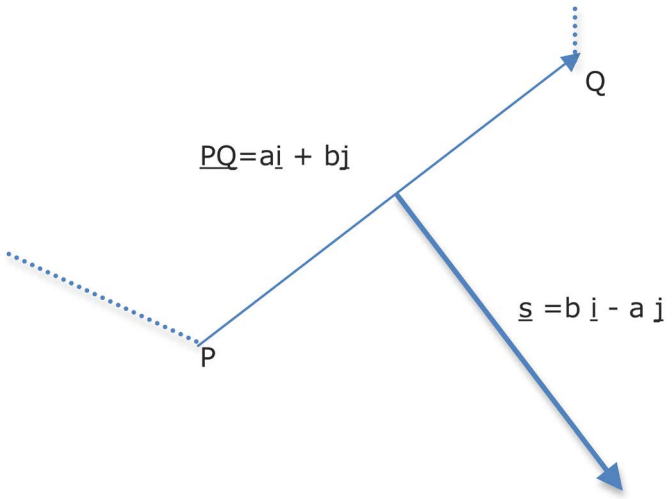

Figure A.2: Cell side and side vector $\underline{s}$.

Exercise A

1. Vectors in 2D are given by, $\underline{x} = 4\,\underline{i} + 3\,\underline{j}$, $\underline{y} = 2\,\underline{i} - 3\,\underline{j}$. Find,

a) $|\underline{x}|$
b) the unique unit vector in the direction of $\underline{x}$
c) the component of $\underline{y}$ in the direction of $\underline{x}$
d) $\underline{x}.\underline{y}$
e) two unit vectors normal to $\underline{y}$
f) the vector parallel to $\underline{x}$ having twice its length.

2. Prove that for any non-zero 2D vectors $\underline{x}$ and $\underline{y}$ and any scalar a,

a) $a\,(\underline{x} \cdot \underline{y}) = (a\underline{x}) \cdot \underline{y} = \underline{x} \cdot (a\underline{y}) = a\,(\underline{y} \cdot \underline{x})$
b) $\underline{x}$ is normal to $\underline{y}$ if and only if $\underline{x} \cdot \underline{y} = 0$
c) $\underline{x}/|\underline{x}|$ is the unique unit vector in the direction of $\underline{x}$
d) result (A.7)

3. Find a unit vector parallel to the line from A(3, 4) to B(1, 2) and pointing from A to B (i.e. vector $\underline{AB}$).

4. Show that (A.10) does give a normal vector of the correct length.

5. a) A computational cell has vertices (0,0), (2,0), (3, 3) and (1,2) and the sides are traversed in an anticlockwise direction. Draw the cell and find all side vectors. Draw the side vectors as per Figure A.2 to verify that they point out of the cell.
   b) Given that the flux vector is $\underline{H} = 6\,\underline{i} + 3\,\underline{j}$ on each cell side find the total flux out of the cell.

6. A computational cell has vertices (1, 0), (0, 1), (-1, 0) and (-1, -1). The flux vector, $\underline{H}$, is assumed to be constant on each of the 4 cell sides. Let $\underline{H}_k$ be the flux on the cell side in the k^{th} quadrant. Find the total flux out of the cell when, $\underline{H}_1 = 2\,\underline{i} + 4\,\underline{j}$, $\underline{H}_2 = -3\,\underline{i} + 4\,\underline{j}$, $\underline{H}_3 = -\,\underline{i} - 3\,\underline{j}$, $\underline{H}_4 = 2\,\underline{i} - 5\,\underline{j}$.

7. Using the vectors of Q1 find $|\underline{x} \times \underline{y}|$.

8. Verify by a couple of examples that the area of a 2D quadrilateral is half the magnitude of the cross product of its diagonal vectors (where the diagonal vectors are regarded as being in 3D with a zero $\underline{k}$ component).

9. Derive a vector based formula for the shortest distance from point (x, y) to the line through points P(a, b) and Q(c, d).

Appendix B Review of Vector Calculus

B.1 Introduction

We review vector calculus, line integrals and double integrals and present two key results which underpin the Finite Volume Method (FVM) in 2D. This Appendix assumes the reader is familiar with vectors (see Appendix A).

B.2 Vector Calculus

We present the basics of vector calculus in 2D but definitions and results extend easily to 3D. It should be noted that the various operations have physical meanings described in detail in any textbook on vector calculus.

B.2.1 Vector Fields

A vector field is simply a function whose values at the independent variables are vectors. In 2D, a general vector field, $\underline{H}$, has the form,

$$\underline{H} = f(x, y)\,\underline{i} + g(x, y)\,\underline{j} \tag{B.1}$$

In the FVM the flux (density) function is a vector field. The component of $\underline{H}$ in the direction of the unit vector $\underline{n}$ is $\underline{H}.\underline{n}$. Vector fields can be used to describe the rates of change of a function in the coordinate directions as described in the next section.

B.2.2 The Gradient Operator ∇

It is often necessary to calculate the rates of change (i.e. partial derivatives) of a function in each of the coordinate directions and store the results in a vector. This operation is neatly described by the gradient operator, ∇ (pronounced 'grad') which is defined in 2D by,

$$\nabla = \underline{i}\,\frac{\partial}{\partial x} + \underline{j}\,\frac{\partial}{\partial y} \tag{B.2}$$

Given a suitably differentiable function, f = f(x, y), we can apply the gradient operator to it to obtain a vector field, $\underline{H}$, whose components are the rates of change of f in the coordinate directions i.e.

$$\underline{H} = \nabla f = \underline{i}\,\frac{\partial f}{\partial x} + \underline{j}\,\frac{\partial f}{\partial y} \tag{B.3}$$

f is said to be a *potential function* for the vector field $\underline{H}$. A vector field for which there exists a potential function is said to be *conservative*.

B.2.3 The Divergence Operator div ($\nabla\cdot$)

The divergence of a vector field $\underline{H}$ is a scalar valued function written div($\underline{H}$) or often $\nabla\cdot\ \underline{H}$. Let $\underline{H}$ be defined as in (B.1) then,

$$\begin{aligned}\nabla\cdot \underline{H} &= \left(\underline{i}\ \frac{\partial}{\partial x} + \underline{j}\ \frac{\partial}{\partial y}\right)\ .\ \left(\ \underline{i}\ f(x,y) + \underline{j}\ g(x,y)\ \right)\\ &= \frac{\partial f}{\partial x} + \frac{\partial g}{\partial y}\end{aligned} \tag{B.4}$$

The divergence measures the net flow at a point. If the divergence is zero there is no net flow (a characteristic of a conserved quantity).

B.3 Line Integrals and Double Integrals

Line integrals and double integrals play a fundamental role in the FVM in 2D. In this section we review the basic concepts and obtain a key result for the FVM.

B.3.1 Line Integrals

A line integral is a generalization of the familiar 1D integral,

$$\int_{x=a}^{x=b} f(x)\ \ dx \tag{B.5a}$$

dx is the infinitesimal distance along the *line on the x-axis* from x=a to x=b. The integral is defined as the limit of a sum of terms (for more details consult any textbook on integral calculus).

Three important properties are:

I1) If k is constant, $\int_{x=a}^{x=b} k\ \ f(x)\ \ dx = k\ \int_{x=a}^{x=b} f(x)\ \ dx$

I2) If a < c < b, $\int_{x=a}^{x=b} f(x)\ \ dx = \int_{x=a}^{x=c} f(x)\ \ dx + \int_{x=c}^{x=b} f(x)\ \ dx$

I3) $\int_{x=a}^{x=b} 1\ \ dx$ = length of the line from x=a to x=b.

In a line integral f(x) is replaced by f(x, y), the line on the x-axis is replaced by a *curve C in the xy-plane* and dx is replaced by ds which is the infinitesimal distance along the *curve* C. The line integral is written,

$$\int_C f(x,y)\ \ ds \tag{B.5b}$$

The line integral is defined as the limit of a sum of terms (for more details consult any textbook on integral calculus). There are three important properties of line integrals analogous to those of ordinary integrals:

LI1) If k is constant, $\int_C k\ f(x,y)\ ds = k \int_C f(x,y)\ ds$

LI2) If C is made up of non-overlapping curves C_1 and C_2 then,

$$\int_C f(x,y)\ ds = \int_{C_1} f(x,y)\ ds + \int_{C_2} f(x,y)\ ds$$

LI3) $\int_C 1\ ds$ = length of C.

B.3.2 Double Integrals

A double integral is an extension of 1D integration to 2D. In the 1D case, (B.5a), we have integration of f(x) over a line on the x-axis of length (b-a) where dx is an infinitesimal element of the line. In double integral we have integration of f(x, y) over a 2D region R in the xy-plane of area A where dR is an infinitesimal element of the area. The double integral of f(x,y) over the 2D region R in the x-y plane is written,

$$\iint_R f(x,y)\ dR \tag{B.6}$$

The double integral is defined as the limit of a (double) sum of terms (for more details consult any textbook on integral calculus).

B.3.2.1 Definition of Mean Value

The mean value, $\bar{f}$, of f(x, y) over a region R in the xy-plane of area A is defined to be,

$$\bar{f} = \frac{1}{A}\iint_R f(x,y)\ \ dR \tag{B.7}$$

B.3.2.2 FVM: First Key Result

Let f=U(t, x, y). f is a function of x and y (for a fixed t) so by (B.7),

$$\bar{U} = \frac{1}{A}\iint_R U\ \ dR \tag{B.8a}$$

$\bar{U}$, is the mean value of U over R and depends on t, hence differentiating both sides of (B.8a) partially with respect to t gives,

$$\frac{\partial \bar{U}}{\partial t} = \frac{\partial}{\partial t}\left(\frac{1}{A}\iint_R U\ \ dR\right) \tag{B.8b}$$

Since differentiation is a linear operation and the double integral is the limit of sums of terms, the partial derivative operator can be taken inside the double integral to give,

$$\begin{aligned}\frac{\partial \bar{U}}{\partial t} &= \left(\frac{1}{A}\iint_R \frac{\partial U}{\partial t}\ \ dR\right)\\ &= \overline{\frac{\partial U}{\partial t}}\end{aligned} \tag{B.8c}$$

This shows that 'the derivative of the mean is the mean of the derivative' and is one of two key results for the FVM of Chapter 1. The second key result follows from a famous theorem.

B.4 Green's Theorem

Let R be a simply connected region in the xy-plane enclosed by a simple closed curve C and let P(x, y) $\underline{i}$ + Q(x, y) $\underline{j}$ be a vector field where P and Q are suitably differentiable, then, traversing C anti-clockwise,

$$\iint_R \frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\ \ dR \ = \ \oint_C P\ \ dx \ + \ Q\ \ dy \tag{B.9}$$

Green's Theorem, (B.9), relates a double integral over the region to a line integral around its perimeter (the Divergence Theorem is the 3D extension of Green's Theorem and is used for the FVM in 3D). (B.9) is the usual form of Green's Theorem but we need to modify it to give the second key result needed for the FVM.

B.4.1 FVM: Second Key Result

In (B.9), let Q(x, y) = f(x, y), P(x, y) = -g(x, y), giving,

$$\iint_R \frac{\partial f}{\partial x} - \frac{\partial(-g)}{\partial y}\ dR = \oint_C (-g)\ dx + f\ dy \tag{B.10a}$$

$$\therefore \iint_R \frac{\partial f}{\partial x} + \frac{\partial g}{\partial y}\ dR = \oint_C -g\ dx + f\ dy \tag{B.10b}$$

writing integrands as dot products (B.10b) becomes,

$$\iint_R (\underline{i}\frac{\partial}{\partial x} + \underline{j}\frac{\partial}{\partial y}) \cdot (f\ \underline{i} + g\ \underline{j})\ dR = \oint_C (-g\ \underline{i} + f\ \underline{j}) \cdot (dx\ \underline{i} + dy\ \underline{j}) \tag{B.10c}$$

Let $\underline{H} = f\,\underline{i} + g\,\underline{j}$, and $\underline{ds} = dx\,\underline{i} + dy\,\underline{j}$. $\underline{H}$ is a vector field and $\underline{ds}$ is the infinitesimal vector of length ds *tangent* to C at the point (x, y) and pointing in the anti-clockwise direction. (B.10c) becomes,

$$\iint_R \nabla \cdot \underline{H}\ dR = \oint_C (-g\ \underline{i} + f\ \underline{j}) \cdot \underline{ds} \tag{B.10d}$$

writing $\underline{ds}$ in component form (B.10d) becomes,

$$\iint_R \nabla \cdot \underline{H}\ dR = \oint_C (-g\ \underline{i} + f\ \underline{j}) \cdot (dx\ \underline{i} + dy\ \underline{j}) \tag{B.10e}$$

using properties of dot products (B.10e) can be rewritten as,

$$\iint_R \nabla \cdot \underline{H}\ dR = \oint_C (f\ \underline{i} + g\ \underline{j}) \cdot (dy\ \underline{i} - dx\ \underline{j}) \tag{B.10f}$$

From Appendix A we see that (dy $\underline{i}$ - dx $\underline{j}$) is the unique *normal* vector to $\underline{ds}$ which points to the right (of the direction defined by $\underline{ds}$) and whose length is ds. Since ds is the tangent vector to C and C is traversed in the anti-clockwise direction (dy $\underline{i}$ - dx $\underline{j}$) is the *outward pointing normal* vector to C. Letting $\underline{n}$ denote the unique *unit* vector in the direction of (dy $\underline{i}$ - dx $\underline{j}$) we may write, (dy $\underline{i}$ - dx $\underline{j}$) = $\underline{n}$ ds and so (B.10f) becomes,

$$\iint_R \nabla \cdot \underline{H}\ dR = \oint_C \underline{H} \cdot\ \underline{n}\ ds \tag{B.10g}$$

This is the second key result for the FVM of Chapter 1.

B.5 Approximation of the Line Integral

The FVM in 2D requires the *approximation* of a line integral around the perimeter, C, of each mesh cell in the computational domain.

We have to approximate,

$$\oint_C \underline{H} \cdot \underline{n} \; ds \tag{B.11a}$$

For our purposes each cell is a quadrilateral (since we are using a structured mesh) so has 4 sides, C_1, C_2, C_3 and C_4. By LI2), (B.11a) becomes,

$$\oint_C \underline{H} \cdot \underline{n} \; ds = \oint_{C_1} \underline{H} \cdot \underline{n} \; ds + \oint_{C_2} \underline{H} \cdot \underline{n} \; ds + \oint_{C_3} \underline{H} \cdot \underline{n} \; ds + \oint_{C_4} \underline{H} \cdot \underline{n} \; ds \tag{B.11b}$$

$\underline{H}$ is *approximated* by a constant, $\underline{H}_i$, on side C_i. Since C_i is a straight line it has the same outward pointing unit normal vector, $\underline{n}_i$, at any point on it. Therefore $\underline{H}_i \cdot \underline{n}_i$ is constant on C_i. Hence, by LI1),

$$\oint_{C_i} \underline{H}_i \cdot \underline{n}_i \; ds \approx \left(\underline{H}_i \cdot \underline{n}_i\right) \oint_{C_i} 1 \; ds \tag{B.11c}$$

By LI3), the right hand line integral in (B.11c) is the length, L_i, of the line C_i. Hence,

$$\begin{aligned} \oint_{C_i} \underline{H}_i \cdot \underline{n}_i \; ds &\approx \left(\underline{H}_i \cdot \underline{n}_i\right) L_i \\ &= \underline{H}_i \cdot \left(L_i \underline{n}_i\right) \\ &= \underline{H}_i \cdot \underline{s}_i \end{aligned} \tag{B.11d}$$

where $\underline{s}_i$ is the outward pointing normal vector to side C_i whose length is the length of C_i. $\underline{s}_i$ is called a *side vector*.

Putting these results together the line integral in (B.11a) is approximated by,

$$\oint_C \underline{H} \cdot \underline{n} \; ds \approx \underline{H}_1 \cdot \underline{s}_1 + \underline{H}_2 \cdot \underline{s}_2 + \underline{H}_3 \cdot \underline{s}_3 + \underline{H}_4 \cdot \underline{s}_4 \tag{B.12}$$

Exercise B

1. Find ∇f ('grad f') when a) $f(x, y) = 3\sin(x)y + x^4\cos(5y)$.

2. Find the component of the vector field $\underline{H} = xy\,\underline{i} + y^2\sin(x)\,\underline{j}$ in the direction of the vector $\underline{v} = 3\,\underline{i} + 4\,\underline{j}$.

3. Find div($\underline{H}$) where $\underline{H}$ is defined in Q2.

4. Show that Laplace's equation, $\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = 0$, can be written in vector form as, $\nabla \cdot (\nabla U) = 0$ (this is sometimes expressed as $\nabla^2 U = 0$, 'del squared U equals zero').

5. Prove I3).

6. Verify LI3) when C is a line parallel to the x-axis.

7. Using the standard result that,

$$\oint_C f(x, y) \; ds = \int_{t=a}^{t=b} f(x(y), y(t)) \; \sqrt{x'(t)^2 + y'(t)^2} \; dt$$

   where C is the curve parameterized by (x(t), y(t)) from t=a to t=b, show that LI3) holds for any line in the xy-plane.

8. f(x, y)=2 at all points in the xy-plane. Find the mean value of f over the unit circle.

9. Verify that 'the derivative of the mean is the mean of the derivative' in 1D. i.e. show that

$$\frac{\partial \bar{U}}{\partial t} = \frac{1}{(b-a)} \int_a^b \frac{\partial U}{\partial t} \, dx$$

by choosing any function $U=U(t, x)=t^2x^3 + t + x$.

Repeat with another function of your choice.

10. a) Show that (B.10c) follows from (B.10b)
    b) Show that (B.10d) follows from (B.10c)
    c) Show that (B.10f) follows from (B.10d)
    d) Show that dy $\underline{i}$ - dx $\underline{j}$ is normal to $\underline{ds}$ with length $|\underline{ds}|$ and hence write dy $\underline{i}$ - dx $\underline{j}$ as the product of a scalar and a unit vector.

11. R is a unit square in the xy-plane centred on (0, 0) with perimeter C. $\underline{H}$ is a 2D vector field and $\underline{H} = 3\underline{i} + 6\underline{j}$ on C.
    a) Express $\iint_R \nabla \cdot \underline{H} \, dR$ as a single line integral,
    b) Write this line integral as the sum of 4 separate line integrals,
    c) Evaluate each line integral and hence find $\iint_R \nabla \cdot \underline{H} \, dR$.

12. C is the closed curve defined by the perimeter of the quadrilateral, R, in the xy-plane whose vertices are (2, 2), (4, 2), (5, 6), (3, 3).
    $\underline{H} = -\underline{i} + 3\underline{j}$ on C.
    a) Sketch C and calculate the outward pointing unit normals for each side of R.
    b) Evaluate $\oint_C \underline{H} \cdot \underline{n} \, ds$,
    c) Show that, $\oint_C \underline{H} \cdot \underline{n} \, ds = \sum_{i=1}^{4} \underline{H} \cdot \underline{s}_i$, where the sum is taken over each side of R and $\underline{s}_i$ is the outward pointing normal vector for side i whose length is the length of side i.
    d) Given that R is a general quadrilateral and $\underline{H} = \underline{H}_i$ = constant on side i, show that the result in c) is true for this general case.

13. Find all side vectors for the cell whose vertices are (0, 0), (1, 0), (1, 1), (-1, 2).

Appendix C The Anatomy of a Finite Volume Code

C.1 Introduction

Coding up a finite volume solver is quite complicated so a structured approach, where the code is broken down in to a number of smaller units, is a good strategy. This approach encourages the coder to think about the essential elements of the program and how they fit together. A structured approach allows each unit to be tested easily and units can be re-used in other codes. In the following example a finite volume FOU scheme for the 2D linear advection equation is coded in Matlab using subfunctions to break up the code into independent units. This code, which is named, 'FVlinearadvectionFOU2D.m', may be downloaded from the website and was used as a basis for the more complicated shallow water solvers. In the following listing numbers have been appended to the start of each line the code. These line numbers are not part of the Matlab code and are given so that specific lines can be referred to easily in the commentary in section C.4. Note that in Matlab the % symbol is used to prefix comments.

It should be noted that the presented code has been written for ease of understanding, not for computational efficiency or elegance. Higher compute speed could be achieved by making more use of Matlab's 'whole array' operations which are much faster than an equivalent for loop.

C.2 Code Structure

Before listing the code it is useful to outline its structure. This is essentially,

1. Briefly describe the code and variables.
2. Input parameter values.
3. Read in the mesh.
4. Compute cell areas and side vectors.
5. Initialise variables on the mesh.
6. Insert ghost cells.
7. Start the time marching loop.
8. Insert ghost values.
9. Calculate the time step.
10. Calculate interface fluxes.
11. Implement solver and update solution.
12. Return to step 7) until the run time is achieved.
13. Output results.

C.3 Code Listing

```
function FVlinearadvectionFOU2D
% File name: FVlinearadvectionFOU2D.m
% Author: Clive Mingham
% Date: 24 Feb 2011
% Description: Solves the 2D PDE, dU/dt + div(H) = 0 using the FOU finite volume scheme.
% This equation corresponds to the finite volume form:
% doubleintegral(dU/dt dA) lineintegral(H.n ds) = 0.
%
% In this case the equation is the linear advection equation so: flux density: H = vU
% velocity: v = vx i + vy j = <vx,vy> where vx and vy are constant.
%
% The program is written in a structured way using subfunctions so that it
% can easily be modified for other equations, schemes and meshes.
% Subfunctions: freadmesh, fcellarea, fsidevectors, finitialu, fIflux, finsertghostcells,
% fremoveghostcells, fcalcdt, fdrawmesh, fplotresults, fdisplay.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Verification: verified on a uniform Cartesian mesh.
% Comments: Although program appears to work the numerical scheme is highly dissipative.
%
clc; clf; clear all;
runtime=1.0;   % target runtime [s]
t=0.0;         % time [s]
tlevel=0;      % time level
```

```
vx=10;          % velocity component in x direction [m/s]
vy=10;          % velocity component in y direction [m/s]
v=[vx,vy];      % velocity vector <vx,vy>
[x,y,xcen,ycen,solid]=freadmesh; % gets coordinates of the 2D computational mesh
                                 % cell centres and solid flags (excluding ghost cells).
disp('read in mesh')
[m,n]=size(x);
NI=m-1; % number of computational cells in i direction.
NJ=n-1; % number of computational cells in j direction.
A=fcellarea(x,y); % compute and store cell areas in an NIxNJ array
disp('calculated cell areas')
S=fsidevectors(x,y); % compute and store cell side vectors in an
                     % (NI+2)x(NJ+2)x4x2 array which contains ghost cells.
disp('calculated cell side vectors')
u=finitialu(xcen,ycen,0,v); % put initial cell centre values of u in NIxNJ array
uinitial=u; % store for plotting initial conditions
disp('inserted initial conditions')
% Append extra cells around arrays to store any ghost values.
u=finsertghostcells(u); % u is now (NI+2)x(NJ+2)
unext=zeros(size(u)); % (NI+2)x(NJ+2) array for u values at next time level
A=finsertghostcells(A); % A is now (NI+2)x(NJ+2)
%
disp('time marching starts')
while(t<runtime)
u=fbcs(u); % Implement boundary conditions using ghost cells.
% u is (NI+2)x(NJ+2) and each computational cell is indexed i=2 to NI+1, j=2 to NJ+1.
dt=fcalcdt(A,S,v); % Finds stable time step for each iteration.
t=t+dt % update time
tlevel=tlevel+1;
for j=2:NJ+1
  for i=2:NI+1
     IH=fIflux(v,u,i,j); % gets interface fluxes for cell (i,j)
     IH1=[IH(1,1),IH(1,2)]; % interface flux vector for side 1
     IH2=[IH(2,1),IH(2,2)]; % interface flux vector for side 2
     IH3=[IH(3,1),IH(3,2)]; % interface flux vector for side 3
     IH4=[IH(4,1),IH(4,2)]; % interface flux vector for side 4
     %
     s1=[S(i,j,1,1),S(i,j,1,2)]; % side vector for side 1
     s2=[S(i,j,2,1),S(i,j,2,2)]; % side vector for side 2
     s3=[S(i,j,3,1),S(i,j,3,2)]; % side vector for side 3
     s4=[S(i,j,4,1),S(i,j,4,2)]; % side vector for side 4
```

```
65.         % FV scheme
66.         % compute total flux out of cell (i,j)
67.         totalfluxout=dot(IH1,s1)+dot(IH2,s2)+dot(IH3,s3)+dot(IH4,s4);
68.         unext(i,j)=u(i,j)-(dt/A(i,j))*totalfluxout;
69.     end % of i loop
70.     end % of j loop
71.     u(2:NI+1,2:NJ+1)=unext(2:NI+1,2:NJ+1); % update u values
72.     %
73.     end % of while loop
74.     disp('time marching ends')
75.     uexact=finitialu(xcen,ycen,t,v); % exact solution at time t
76.     u=u(2:NI+1,2:NJ+1); % extract final computational values
77.     fdrawmesh(x,y,solid) % draws mesh if fplotresults is commented out
78.     % fdisplay(u); % print results as a matrix for a small mesh if needed
79.     fplotresults(xcen,ycen,uinitial,u,uexact) % plot results: comment in or out
80.     disp('finished main program, see figures')
81.     end % FVlinearadvectionFOU2D
82.     %%%%%%%%%%% subfunctions %%%%%%%%%%%%%%%%%%
83.     function [x,y,xcen,ycen,solid]=freadmesh
84.     % The mesh is structured and has NI cells in the i direction and
85.     % NJ cells in the j direction (so each cell has 4 sides).
86.     % The x and y coordinates of the lower left hand corner of cell (i,j) are
```

```
87.   % held in arrays x and y respectively which are both (NI+1)x(NJ+1). In
88.   % this way the 4 vertices of cell (i,j) are (x(i),y(j)), (x(i+1),y(j)),
89.   % (x(i+1),y(j+1)) and (x(i),y(j+1)). xcen and ycen are NI by NJ arrays
90.   % which hold cell centres. solid is an NI by NJ array which
91.   % flags solid cells. If cell (i,j) is solid then solid(i,j)=1 otherwise
92.   % solid(i,j)=0.
93.   %
94.   NI=61;
95.   NJ=41;
96.   dx=2;
97.   dy=4;
98.   x=zeros(NI+1,NJ+1); % allocate correct sized array
99.   y=zeros(NI+1,NJ+1); % allocate correct sized array
100.  xcen=zeros(NI,NJ); % allocate correct sized array
101.  ycen=zeros(NI,NJ); % allocate correct sized array
102.  solid=zeros(NI,NJ); % allocate correct sized array
103.  % create cell vertices
104.  for i=1:NI+1
105.    for j=1:NJ+1
106.      x(i,j)=(i-1)*dx;
107.      y(i,j)=(j-1)*dy;
108.    end % of j loop
109.  end % of i loop
110.  nonCartesian=1; % 1 for non-Cartesian mesh
111.  if (nonCartesian==1)
112.    disp('mesh is non-Cartesian')
113.    % Add 'random' numbers to mesh coordinates to produce a non-Cartesian mesh
114.    % ensure that the abs of these numbers are less than min(dx/2,dy/2).
115.    % Create non-Cartesian mesh
116.    for i=1:NI+1
117.     for j=1:NJ+1
118.       x(i,j)=x(i,j)+(dx/6)*cos(1.727*(i+j));
119.      y(i,j)=y(i,j)+(dy/5)*sin(2.46723*i*j);
120.    end % of j loop
121.   end % of i loop
122.  else
123.    disp('mesh is Cartesian')
124.  end % end of if statement create mesh
125.  %
126.  % find cell centres by averaging coordinates of vertices
127.  for i=1:NI
128.    for j=1:NJ
```

```
        xcen(i,j)=(x(i,j)+x(i+1,j)+x(i+1,j+1)+x(i,j+1))/4;
        ycen(i,j)=(y(i,j)+y(i+1,j)+y(i+1,j+1)+y(i,j+1))/4;
    end % of j loop
end % of i loop
end % freadmesh
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function A=fcellarea(x,y)
% Verification: verified.
% cell area = half the modulus of cross product of cell diagonal vectors.
% These vectors are 2D so append 0 k component to make 3D for cross product
[m,n]=size(x);
NI=m-1; NJ=n-1;
A=zeros(NI,NJ);
for i=1:NI
    for j=1:NJ
        d1=[x(i+1,j+1)-x(i,j),y(i+1,j+1)-y(i,j),0];
        d2=[x(i,j+1)-x(i+1,j),y(i,j+1)-y(i+1,j),0];
        d=cross(d1,d2);
       A(i,j)=0.5*sqrt(dot(d,d));
    end % of j loop
end % of i loop
end % fcellarea
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function S=fsidevectors(x,y)
% Verification: Verified.
% Finds the 4 side (S) vectors for each cell. Cell sides are numbered 1 to 4 anti-clockwise.
% For cell (i,j) side 1 is the vector s1 from (x(i,j), y(i,j)) to (x(i+1,j),y(i+1,j)).
% if s1=<a,b> then the corresponding side vector S1=<b,-a>.
% The side vector array has ghost cells around it so that side vector
% indices (i+1,j+1) refer to cell (i,j).
% Note that vectors are 1 by 2 arrays so care must be taken because side vectors are stored in
% higher dimensional arrays.
%
[m,n]=size(x);
NI=m-1; NJ=n-1;
S=zeros(NI+2,NJ+2,4,2);
for i=1:NI
    for j=1:NJ
        s1=[x(i+1,j)-x(i,j),y(i+1,j)-y(i,j)];
        S(i+1,j+1,1,1)=s1(2);
        S(i+1,j+1,1,2)=-s1(1);
        %
```

```
            s2=[x(i+1,j+1)-x(i+1,j),y(i+1,j+1)-y(i+1,j)];
            S(i+1,j+1,2,1)=s2(2);
            S(i+1,j+1,2,2)=-s2(1);
            %
            s3=[x(i,j+1)-x(i+1,j+1),y(i,j+1)-y(i+1,j+1)];
            S(i+1,j+1,3,1)=s3(2);
            S(i+1,j+1,3,2)=-s3(1);
            %
            s4=[x(i,j)-x(i,j+1),y(i,j)-y(i,j+1)];
            S(i+1,j+1,4,1)=s4(2);
            S(i+1,j+1,4,2)=-s4(1);
        end % of j loop
    end % of i loop
    end % fsidevectors
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function u=finitialu(xcen,ycen,t,v)
    % Inserts exact u values (and initial conditions for t=0)
    [NI,NJ]=size(xcen);
    u=zeros(NI,NJ); % create correct sized array (without ghost values)
    vx=v(1);
    vy=v(2);
    %
```

```
% Initial 2D Gaussian function based on the centre (xc,yc) of the
% computational domain.
xmax=max(max(xcen)); % max x value in mesh
ymax=max(max(ycen)); % max y value in mesh
xmin=min(min(xcen)); % min x value in mesh
ymin=min(min(ycen)); % min y value in mesh
xc=(xmax+xmin)/2; % approx x coord of mesh centre
yc=(ymax+ymin)/2; % approx y coord of mesh centre
%
cutoff=min(xmax-xc,ymax-yc)/2; % cut off radius for Gaussian
for i=1:NI
 for j=1:NJ
  x=xcen(i,j)-vx*t;
   y=ycen(i,j)-vy*t;
    d=sqrt((x-xc)^2+(y-yc)^2); % distance from centre
     if (d<cutoff)
       u(i,j)=exp(-0.01*d^2);
    else
       u(i,j)=0;
    end
  end % of j loop
end % of i loop
end % finitialu
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function IH=fIflux(v,u,i,j)
% Calculates the 4 interface fluxes on the 4 sides of cell (i,j). Cell sides are numbered 1 to 4
% anti-clockwise. For cell (i,j) side 1 is the vector s1 from (x(i,j), y(i,j)) and to
% (x(i+1,j),y(i+1,j)).
% Interface fluxes are denoted IH1, IH2, IH3 and IH4 and are 2D vectors. Flux H depends on
% U, i.e. H = H(U). For the 2D linear advection equation, H(U) = <vx U, vy U>.
% There is considerable choice for interface flux estimation: different choices give different
% FV schemes.
% The following scheme simply takes:
% IH1=H(u(i,j-1)) = v u(i,j-1)
% IH2=H(u(i,j)) = v u(i,j)
% IH3=H(u(i,j)) = v u(i,j)
% IH4=H(u(i-1,j)) = v u(i-1,j)
% This is an upwind scheme for vx > 0 and vy >0 and corresponds to the FOU finite
% difference scheme on a Cartesian mesh.
%
IH=zeros(4,2); % array to store all 4 interface flux vectors for a cell
vx=v(1);
```

```
vy=v(2);
%
IH(1,1)=vx*u(i,j-1); % component of IH1 in the x direction
IH(1,2)=vy*u(i,j-1); % component of IH1 in the y direction
IH(2,1)=vx*u(i,j); % component of IH2 in the x direction
IH(2,2)=vy*u(i,j); % component of IH2 in the y direction
IH(3,1)=vx*u(i,j); % component of IH3 in the x direction
IH(3,2)=vy*u(i,j); % component of IH3 in the y direction
IH(4,1)=vx*u(i-1,j); % component of IH4 in the x direction
IH(4,2)=vy*u(i-1,j); % component of IH4 in the y direction
end % fIflux
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function outarray=finsertghostcells(inarray)
% Assuming that ghost cells are needed around the whole domain an mxn array is embedded
% into an (m+2)x(n+2) array of zeros. Hence computational indices go from i=2 to m+1 and
% j=2 to n+1.
[m,n]=size(inarray);
dummy=zeros(m+2,n+2);
dummy(2:m+1,2:n+1)=inarray;
outarray=dummy;
end % finsertghostcells
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function outarray=fremoveghostcells(inarray)
% Removes ghost cells so that a mxn array becomes (m-2)x(n-2)
% [m,n]=size(inarray);
% outarray=inarray(2:m-1,2:n-1);
% end % fremoveghostcells
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function u=fbcs(u)
% Implements boundary conditions using ghost cells index by i=1, i=NI+2, j=1, j=NJ+2.
[m,n]=size(u);
NI=m-2;
NJ=n-2;
%
% Dirichlet: u=0 everywhere.
% Don't need to do anything as u values in ghost cells are already zero.
end % fbcs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function dt=fcalcdt(A,S,v)
% Verification:
% Finds allowable time step dt using heuristic formula. 0<F<1 is a tunable safety factor.
[m,n]=size(A); % A and S now enlarged to include ghost cells
```

```
NI=m-2; NJ=n-2;
dt=9999999; % large value of dt to start
F=0.4;
for i=2:NI+1
 for j=2:NJ+1
   % put side vectors into 1x2 arrays
   s2=[S(i,j,2,1),S(i,j,2,2)];
   s3=[S(i,j,3,1),S(i,j,3,2)];
   %
   v2=abs(dot(v,s2)); % side 2
   v3=abs(dot(v,s3)); % side 3
   dum=A(i,j)/max(v2,v3);
   % take smallest value of dt
   if (dum<dt)
       dt=dum;
   end
  end % j loop
end % i loop
dt=F*dt;
end % fcalcdt
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function fdrawmesh(x,y,solid)
```

```
% Description: Draws a structured 2D finite volume mesh and fills in any solid cells.
% Date structures:
% The mesh has NI cells in the i direction and NJ cells in the j direction. The x and y
% coordinates of the lower left hand corner of cell (i,j) are held in arrays x and y respectively
% which are both NI+1 by NJ+1. In this way the 4 vertices of cell (i,j) are (x(i),y(j)),
% (x(i+1),y(j)),
% (x(i+1),y(j+1)) and (x(i),y(j+1)). solid is an NI by NJ array which flags solid cells. If cell
% (i,j) is solid then solid(i,j)=1 otherwise solid(i,j)=0.
%
[m,n]=size(x);
NI=m-1; % number of cells in i direction
NJ=n-1; % number of cells in j direction
%
% Plot the mesh
hold on % keeps all plots on the same axes
% draw lines in the i direction
for i=1:NI+1
    plot(x(i,:),y(i,:))
end
% draw lines in the j direction
for j=1:NJ+1
    plot(x(:,j),y(:,j))
end
title('computational mesh')
xlabel('x')
ylabel('y')
% Fill in solid cells
for i=1:NI
  for j=1:NJ
    if (solid(i,j)==1)
      solidx=[x(i,j),x(i+1,j),x(i+1,j+1),x(i,j+1),x(i,j)];
      solidy=[y(i,j),y(i+1,j),y(i+1,j+1),y(i,j+1),y(i,j)];
      fill(solidx,solidy,'k')
    end
  end % of j loop
end % of i loop
end % fdrawmesh
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function fplotresults(x,y,uinitial,u,uexact)
 % Plots results on a structured mesh - even works for non-Cartesian mesh.
%
subplot(3,1,1), contour(x,y,uinitial)
```

```
341. title('Contour plot of initial conditions')
342. xlabel('x [m]')
343. ylabel('y [m]')
344. %
345. subplot(3,1,2), contour(x,y,u)
346. title('Contour plot of numerical solution')
347. xlabel('x [m]')
348. ylabel('y [m]')
349.  %
350. subplot(3,1,3), contour(x,y,uexact)
351. title('Contour plot of exact solution')
352. xlabel('x [m]')
353. ylabel('y [m]')
354. %
355. end % fplotresults
356. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
357. function fdisplay(in)
358. % displays matrices 'correctly' - WORKS
359. % Using index i for x which are columns and j for y which are rows in reverse order so this
360. % routine displays them in this fashion
361. [NX,NY]=size(in);
362. temp=transpose(in);
363. out=temp(NY:-1:1,:);
364. disp('printing as a matrix')
365. out
366. end % fdisplay
```

C.4 Commentary

Lines 1 to 15 describe the program, author, date etc. and it is good practice to always include such details at the start of any program. Note that the program is written as a Matlab dummy function (i.e. it does not return anything) to permit the inclusion of subfunctions within the same file. The file name (which is descriptive) must be the function name. The drawback of a dummy function is that variables are not available in the Command Window, a feature useful for debugging. Another possibility to get around this problem is to write the subfunctions as separate m-files and call them from a master script m-file. In our convention subfunction names and descriptive and start with 'f'.

Line 17 tells the reader that the program has been verified which is useful although the user should always verify programs themselves, particularly when using a commercial package. Line 18 is a comment on the numerical scheme. In this case, although the scheme has been coded correctly it not very accurate.

Lines 21 to 26 define some of the variables (with units) and assign values. Line 27 calls a subfunction to generate mesh data. Line 29 is a useful print statement to say where the code is up to. Print statement are useful when debugging a program and checking values as the computation proceeds.

Lines 30 to 32 extract the size of the computational domain from the mesh data. Line 33 calls a subfunction to calculate cell areas. This is an example of a subfunction which can be reused in another program. Line 35 calls a subfunction to calculate and side vectors. Side vectors are constant so it makes sense to calculate them once and store them unless storage space is limited.

Line 38 calls a subfunction to input initial conditions and line 39 copies these values to a fixed array in case they are needed for comparison purposes.

Line 42 calls a subfunction to append extra ghost cells around the initial data array. The number of ghost cells depends on the scheme. In this case a single row of ghost cells is appended around the initial array which, because Matlab indices start at 1, means that the first computational cell has index (2, 2). Line 43 creates an array of the correct size which will hold updated solution values at the next time level. Line 44 calls the same subfunction as in line 42 to insert ghost cells around the cell area array.

Line 47 start the time marching loop which runs until the specified run time is exceeded (which means that the specified run time will not be achieved exactly but this can be fixed easily by a conditional statement). Line 48 calls a subfunction to input values into the ghost cells on the basis of the boundary conditions in the problem.

Line 50 calls a subroutine to calculate the time step. In this problem, because propagation velocity is constant it is only necessary to find the time step once before time marching begins but for generality we put this routine in the time marching loop. Line 51 updates the current simulation time.

Lines 53 and 54 loop through the computational cells in the 2D mesh – note the shifted values of start and end indices due to the insertion of ghost cells into the mesh.

Line 55 calls a subfunction to calculate (the four) interface fluxes computational cell (i, j). This defines the FVS.

Lines 56 to 59 write the interface fluxes as separate 2D vectors and lines 61 to 64 do the same for the side vectors.

Line 67 computes the total flux out of a cell and line 68 is the FVS which finds the solution at the next time step from its current value.

Lines 69 and 70 end the spatial loop and line 71 overwrites the current solution with the updated solution prior to the next time step iteration. Line 73 is the end of the time marching loop. In large codes, in addition to indenting loops, it is a good idea to put a comment the end of a loop to identify it.

Line 75 calls a subfunction to generate the exact solution. For more complicated PDEs no exact solutions will be available but here we use the exact solution to observe the characteristics of the numerical scheme. Line 76 extracts the results without ghost values. In general a matrix of results is of little value and an appropriate graph is more useful.

Line 77 calls a function to draw the mesh. This could be useful for checking the mesh but in our case the figure will be overwritten. Line 78 calls a subfunction to display results in the Command Window in the correct way. Our indexing is not the same as matrix indexing so matrices have to be flipped around. This subfunction is commented out in the program but is useful for displaying results from a small computation when verifying the code by comparison to a pen and paper calculation. Line 79 produces appropriate plots of results which in this case are contour plots of initial conditions and numerical and exact solutions. Line 81 is the end of the main program.

Lines 83 to 133 form the subfunction to read in the mesh. More generally this function would be replace by a statement reading the mesh data from a file. An option to flag cells as solid is given but not used. For illustrative purposes a non-Cartesian mesh can be created by perturbing a Cartesian mesh. Subfunctions are separated by lines of comment statements for ease of viewing.

Lines 135 to 150 form the subfunction to calculate cell areas for general four sided cells in 2D and lines 152 to 184 form the subfunction to calculate side vectors for such cells.

Lines 186 to 215 form a subfunction to generate initial conditions (and exact solution). In general initial conditions would be read in from a file and there would not be an exact solution.

Lines 217 to 245 form a subfunction to calculate interface fluxes for a cell. In this case interface fluxes are based on the nearest upwind cell centre values which is the FOU scheme.

Lines 257 to 261 form a subroutine to remove ghost cells. This is not used.

Lines 263 to 270 form a subroutine to insert boundary values in to ghost cells. In this case all boundary values are zero. These Dirichlet conditions are the default setting.

Lines 273 to 296 form a subfunction to calculate the stable time step based on the heuristic method and a safety factor which can be adjusted based on numerical experiments.

Lines 298 to 335 form a subfunction to draw the mesh and mark any solid cells (there aren't any in our case). This subfunction could be useful to check the mesh and the shape of any solid regions.

Lines 337 to 355 form a subroutine to plot the results. In this case contour plots are used and plots work for structured meshes.

Lines 357 to 366 form a subroutine to display matrices of results in the correct way.

Exercise C

1. Download FVlinearadvectionFOU2D.m from the website.
2. Run the code on a uniform Cartesian mesh using positive and negative velocity components and satisfy yourself that the solver gives the correct qualitative behaviour for the solution.

Repeat Q2 on a structured non-Cartesian mesh.

By repeatedly running the code determine the maximum safety factor for a stable time step.

Using a small number of cells verify fcellarea and fsidevectors for a non-Cartesian mesh (you will need to compare you the outputs from these subroutines to pen and paper calculations).

Change subroutine fplotresults to give surface plots.

Modify the time marching loop so that the program terminates at the exact runtime specified by the user.

Modify the boundary condition subfunction fbcs so that periodic boundary conditions are imposed everwhere. Test by running the program so that the concentration profile leaves one side of the computational domain and appears on the opposite side.

Modify the program so that results are animated.