

Introductory Finite Difference Methods for PDEs

Professor D. M. Causon; Professor C. G. Mingham



Download free books at

bookboon.com

Professor D. M. Causon & Professor C. G. Mingham

Introductory Finite Difference Methods for PDEs

Introductory Finite Difference Methods for PDEs

© 2010 Professor D. M. Causon, Professor C. G. Mingham & Ventus Publishing ApS

ISBN 978-87-7681-642-1

Contents

Preface	9
1. Introduction	10
1.1 Partial Differential Equations	10
1.2 Solution to a Partial Differential Equation	10
1.3 PDE Models	11
1.4 Classification of PDEs	11
1.5 Discrete Notation	15
1.6 Checking Results	15
1.7 Exercise 1	16
2. Fundamentals	17
2.1 Taylor's Theorem	17
2.2 Taylor's Theorem Applied to the Finite Difference Method (FDM)	17
2.3 Simple Finite Difference Approximation to a Derivative	18
2.4 Example: Simple Finite Difference Approximations to a Derivative	18
2.5 Constructing a Finite Difference Toolkit	20
2.6 Simple Example of a Finite Difference Scheme	24
2.7 Pen and Paper Calculation (very important)	28
2.8 Exercise 2a	32
2.9 Exercise 2b	33

Please click the advert



Dear highly educated engineering and finance students,

if you are driven, ambitious, open-minded and focused - we have a challenge for you. Actually, the greatest challenge in the world. Curious? Visit dongenergy.com/job

Best wishes

DONG Energy

dongenergy.com/job




3.	Elliptic Equations	34
3.1	Introduction	34
3.2	Finite Difference Method for Laplace's Equation	34
3.3	Setting up the Equations	37
3.4	Grid Convergence	38
3.5	Direct Solution Method	38
3.6	Exercise 3a	41
3.7	Iterative Solution Methods	42
3.8	Jacobi Iteration	43
3.9	Gauss-Seidel Iteration	45
3.10	Exercise 3b	47
3.11	Successive Over Relaxation (SoR) Method	47
3.12	Line SoR	49
3.13	Exercise 3c	51
4.	Hyperbolic Equations	52
4.1	Introduction	52
4.2	1D Linear Advection Equation	53
4.3	Results for the Simple Linear Advection Scheme	55
4.4	Scheme Design	60
4.5	Multi-Level Scheme Design	67
4.6	Exercise 4a	69
4.7	Implicit Schemes	70
4.8	Exercise 4b	76

Please click the advert



May we offer you one
of the world's greatest
challenges?
In all humility.

We are looking for highly
educated engineers and
finance students.

Join us at dongenergy.com/job

dongenergy.com/job




5.	Parabolic Equations: the Advection-Diffusion Equation	77
5.1	Introduction	77
5.2	Pure Diffusion	78
5.3	Advection-Diffusion Equation	81
5.4	Exercise 5b	83
6.	Extension to Multi-dimensions and Operator Splitting	84
6.1	Introduction	84
6.2	2D Scheme Design (unsplit)	84
6.3	Operator Splitting (Approximate Factorisation)	92
7.	Systems of Equations	105
7.1	Introduction	105
7.2	The Shallow Water Equations	105
7.3	Solving the Shallow Water Equations	106
7.4	Example Scheme to Solve the SWE	109
7.5	Exercise 7	111
	Appendix A: Definition and Properties of Order	112
A.1	Definition of $O(h)$	112
A.2	The Meaning of $O(h)$	113
A.3	Properties of $O(h)$	113
A.4	Explanation of the Properties of $O(h)$	114
A.5	Exercise A	114

Please click the advert



Ambitious and curious?

We are looking for young and highly educated engineers and finance students.

Join us at dongenergy.com/job

dongenergy.com/job





	Appendix B: Boundary Conditions	115
B.1	Introduction	115
B.2	Boundary Conditions	116
B.3	Specifying Ghost and Boundary Values	118
B.4	Common Boundary Conditions	120
B.5	Exercise B	121
	Appendix C: Consistency, Convergence and Stability	123
C.1	Introduction	123
C.2	Convergence	124
C.3	Consistency and Scheme Order	124
C.4	Stability	126
C.5	Exercise C	133
	Appendix D: Convergence Analysis for Iterative Methods	135
D.1	Introduction	135
D.2	Jacobi Iteration	136
D.3	Gauss-Seidel Iteration	137
D.4	SoR Iterative Scheme	139
D.5	Theory for Dominant Eigenvalues	139
D.6	Rates of Convergence of Iterative Schemes	142
D.7	Exercise D	143

Please click the advert


**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

dongenergy.com/job

Everybody is talking...

...about future energy supply. We are not.
Stop talking and make a career
moving energy forward.

Ambitious engineers and
finance students go to
dongenergy.com/job



Professor D.M. Causon and Professor C.G. Mingham

Department of Computing and Mathematics, Manchester Metropolitan University, UK

To our parents and to Mags

Preface

The following chapters contain core material supported by pen and paper exercises together with computer-based exercises where appropriate. In addition there are web links to:

- worked solutions,
- computer codes,
- audio-visual presentations,
- case studies,
- further reading.

Codes are written using Scilab (a Matlab clone, downloadable for free from <http://www.scilab.org/>) and also Matlab.

The emphasis of this book is on the practical: students are encouraged to experiment with different input parameters and investigate outputs in the computer-based exercises. Theory is reduced to a necessary minimum and provided in appendices. Web links are found on the following web page:

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

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

1. Introduction

This book provides an introduction to the finite difference method (FDM) for solving partial differential equations (PDEs). In addition to specific FDM details, general concepts such as stability, boundary conditions, verification, validation and grid independence are presented which are important for anyone wishing to solve PDEs by using other numerical methods and/or commercial software packages. Material is presented in order of increasing complexity and supplementary theory is included in appendices.

1.1 Partial Differential Equations

The following equation is an example of a PDE:

$$a(t, x, y) U_t + b(t, x, y) U_x + c(t, x, y) U_{yy} = f(t, x, y) \quad (1.1)$$

where,

- t, x, y are the *independent* variables (often time and space)
- a, b, c and f are known functions of the independent variables,
- U is the *dependent* variable and is an unknown function of the independent variables.
- partial derivatives are denoted by subscripts: $U_t = \frac{\partial U}{\partial t}$, $U_x = \frac{\partial U}{\partial x}$, $U_{yy} = \frac{\partial^2 U}{\partial y^2}$ etc.

The *order* of a PDE is the order of its highest derivative. A PDE is *linear* if U and all its partial derivatives occur to the first power only and there are no products involving more than one of these terms. (1.1) is second order and linear. The *dimension* of a PDE is the number of independent *spatial* variables it contains. (1.1) is 2D if x and y are spatial variables.

1.2 Solution to a Partial Differential Equation

Solving a PDE means finding the unknown function U . An *analytical* (i.e. exact) solution of a PDE is a function that satisfies the PDE and also satisfies any *boundary* and/or *initial conditions* given with the PDE (more about these later). Most PDEs of interest do not have analytical solutions so a *numerical* procedure must be used to find an *approximate* solution. The approximation is made at discrete values of the independent variables and the approximation *scheme* is implemented via a computer program. The FDM replaces all partial derivatives and other terms in the PDE by *approximations*. After some manipulation, a finite difference scheme (FDS) is created from which the approximate solution is obtained. The FDM depends fundamentally on Taylor's beautiful theorem (circa 1712!) which is stated in the next chapter.

1.3 PDE Models

PDEs describe many of the fundamental natural laws (e.g. conservation of mass) so describe a wide range of physical phenomena. Examples include Laplace's equation for steady state heat conduction, the advection-diffusion equation for pollutant transport, Maxwell's equations for electromagnetic waves, the Navier-Stokes equations for fluid flow and many, many more. The authors' main interest is in solving PDEs for fluid flow problems and details, including pictures and animations, can be found at:

<http://www.docm.mmu.ac.uk/cmmfa/>

1.4 Classification of PDEs

Second order linear PDEs can be formally classified into 3 generic types: elliptic, parabolic and hyperbolic. The simplest examples are:

a) Elliptic: e.g. $U_{xx} + U_{yy} = f(x, y)$.

This is Poisson's equation or Laplace's equation (when $f(x, y) = 0$) which may be used to model the steady state temperature distribution in a plate or incompressible potential flow. Notice there is no time derivative.

b) Parabolic: e.g. $U_t = kU_{xx}$.

This is the 1D diffusion equation and can be used to model the time-dependent temperature distribution along a heated 1D bar.

c) Hyperbolic: e.g. $U_{tt} = c^2 U_{xx}$.

This is the wave equation and may be used to model a vibrating guitar string or 1D supersonic flow.

d) $U_t = -cU_x$.

This first order PDE is called the advection equation. Solutions of d) also satisfy c).

e) $U_t + cU_x = kU_{xx}$.

This is the advection-diffusion equation and may be used to model transport of a pollutant in a river. The coefficients k , c in the above PDEs quantify material properties that relate to the problem being solved e.g. k could be the coefficient of thermal conductivity in the case of a heated bar, or 1D diffusion coefficient in the case of pollutant transport; c is a wave speed, usually, in fluid flow, the speed of sound.

1.4.1 Initial and Boundary Conditions

PDEs require proper initial conditions (ICs) and boundary conditions (BCs) in order to define what is known as a well-posed problem. If too many conditions are specified then there will be no solution; if too few conditions are specified the solution will not be unique. If the ICs/BCs are specified in the wrong place or at the wrong time then the solution will not depend smoothly on the ICs/BCs and small errors in the ICs/BCs will bring about large changes in the solution. This is referred to as an ill-posed problem. The PDEs encountered in practice are often non-linear and multi-dimensional and cannot be reduced to the simple so-called canonical forms of a) - e). However, we need to understand the properties of the solution to these simple model PDEs before attempting to solve more complicated PDEs.

A second order elliptic PDE such as a) requires a boundary condition on U at each point on the boundary. Thus, these are called Boundary Value (BV) problems. The BC may be a value of U on the boundary or the value of its derivative (see Appendix B). Linear parabolic equations such as b) require ICs at the initial start time (usually $t=0$) and one BC at each end-point of the spatial domain (e.g. at the ends of the heated bar). Technically linear hyperbolic equations such as d) require ICs and as many BCs as there are inward-pointing characteristics (this is an advanced topic which we will not cover) which depend on the sign of wave speed c , thus:

If $c > 0$, we need ICs: $U(0, x) = f(x)$ and BCs: $U(t, 0) = g(t)$;

If $c < 0$, we need ICs: $U(0, x) = f(x)$ but no BCs.

These are called Initial Boundary Value Problems (IBV) problems.

1.4.2 Domain of Dependence

The differences between the types of PDEs can be illustrated by sketching their respective domains of dependence. So for example, in the hyperbolic case d), point $P(x_0, t_0)$ in Figure 1.1 can only be influenced by points lying within the region bounded by the two characteristics $x+ct = \text{const}$ and $x-ct = \text{const}$ and $t < t_0$. This region is called the *domain of dependence*. In turn, point P can influence points at later times lying within its *zone of influence*. In the parabolic case, shown in Figure 1.2 information travels downstream (or forward in time) only and so the domain of dependence of point $P(x_0, t_0)$ in this case is the region $t < t_0$ and the zone of influence is all points for which $t > t_0$.

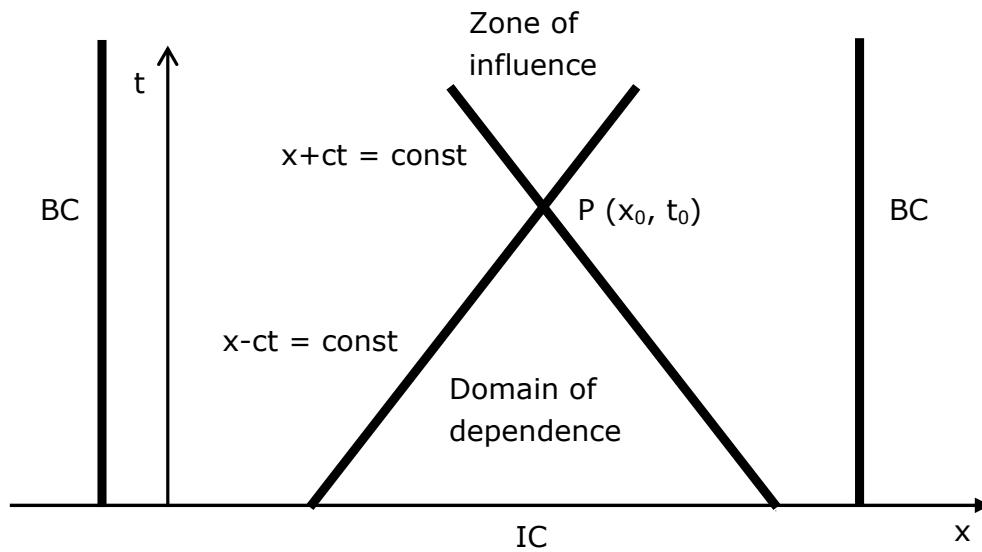


Figure 1.1 Domain of dependence: hyperbolic case.

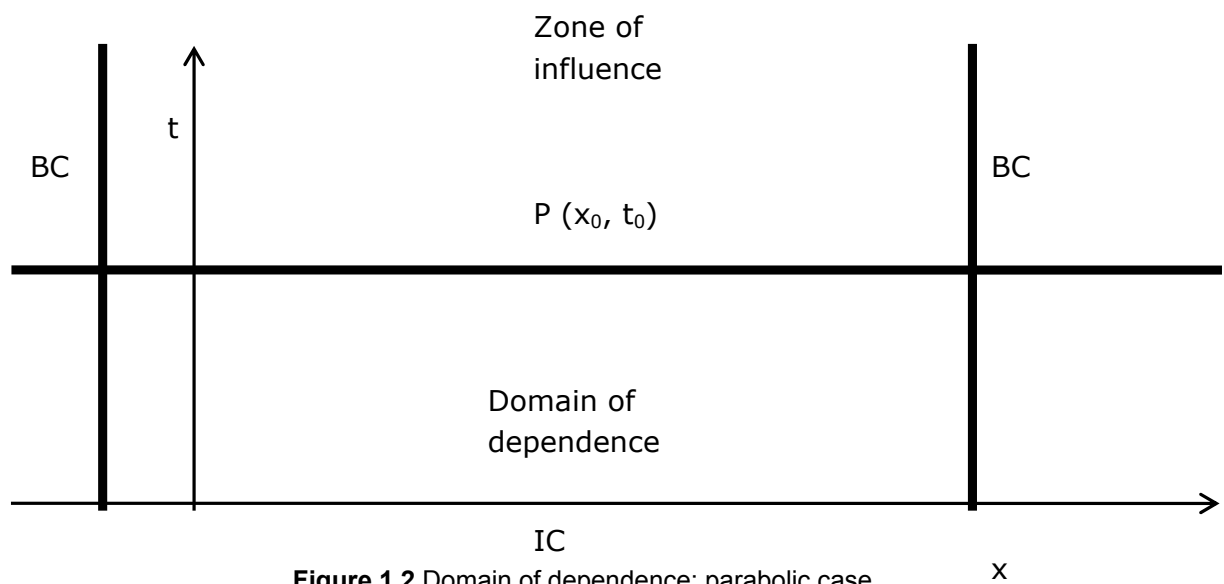


Figure 1.2 Domain of dependence: parabolic case.

In the elliptic case, corresponding to subsonic flow (Figure 1.3), information travels in all directions at infinite speed so the solution at point $P(x_0, t_0)$ influences all points within the domain and vice versa.

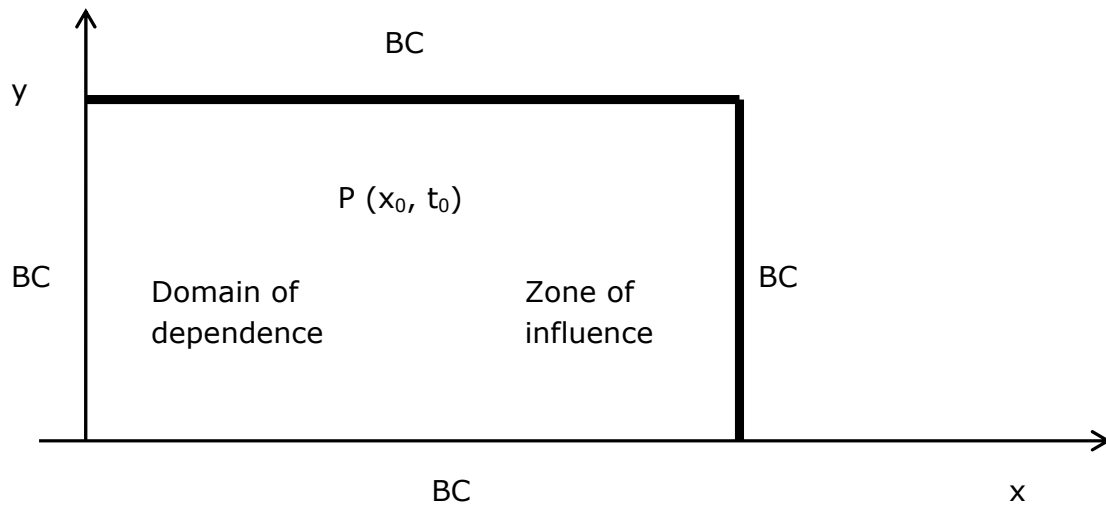
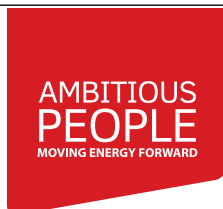


Figure 1.3 Domain of dependence: elliptic case.

Notice in this case that the whole region bounded by the BCs is both a domain of dependence and zone of influence.

Please click the advert



dongenergy.com/job

Dear highly educated engineering and finance students,

if you are driven, ambitious, open-minded and focused - we have a challenge for you. Actually, the greatest challenge in the world. Curious? Visit dongenergy.com/job

Best wishes

DONG Energy



DONG
energy

The type of PDE fundamentally influences the choice of solution strategy. Time dependent hyperbolic problems and parabolic problems illustrated by Figures 1.1 and 1.2 are solved numerically by time-marching methods which involves, as its name suggests, obtaining the numerical solution at a later time from that at an earlier time starting from given ICs.

Elliptic problems, as illustrated in Figure 1.3 are solved numerically by so-called relaxation methods.

1.5 Discrete Notation

We will use upper case U to denote the analytic (exact) solution of the PDE and lower case u to denote the numerical (approximate) solution. Subscripts will denote discrete points in space and superscripts discrete levels in time. e.g. $u_{i,j}^n$ denotes the numerical solution at grid point (i, j) in a 2D region at time level n .

1.6 Checking Results

Before applying a numerical scheme to real life situations modelled by PDEs there are two important steps that should always be undertaken.

1.6.1 Verification

The computer program implementing the scheme must be *verified*. This is a check to see if the program is doing what it is supposed to do. Comparing results from pen and paper calculations at a small number of points to equivalent computer output is a way to (partially) verify a program. Give or take a small amount of rounding error the numbers should be the same. Another way to verify the program is to find an exact solution to the PDE for a simpler problem (if one exists) and compare numerical and exact results. Complete program verification involves testing that all branches, program elements and statements are executed and produce the expected outcomes. For large programs there exist software verification programs to facilitate the verification process. For a commercial solver it may not be possible to completely verify the program if the source code is unavailable.

1.6.2 Validation

Validation is really a check on whether the PDE is a good model for the real problem being studied. Validation means comparing numerical results with results from similar *physical* problems. Physical results may come from measurements from real life or from small-scale laboratory experiments. Either way, due to measurement errors, scaling problems and the inevitable failure of the PDEs to capture all the underlying physics, agreement between numerical and physical results will not be perfect and the user will have to decide what is ‘close enough’.

1.7 Exercise 1

1. Assuming that t is time and x and y are spatial variables give the dimensions of the PDEs in a) to e) of Section 1.4.
2. Classify the following PDEs:

a) $U_{tt} = 2 U_{xx}$, b) $U_{xx} + U_{yy} = 0$, c) $U_t - U_{xx} = 0$.

Please click the advert



dongenergy.com/job

May we offer you one
of the world's greatest
challenges?
In all humility.

We are looking for highly
educated engineers and
finance students.

Join us at dongenergy.com/job





2. Fundamentals

The finite difference method (FDM) works by replacing the region over which the independent variables in the PDE are defined by a finite *grid* (also called a *mesh*) of points at which the dependent variable is approximated. The partial derivatives in the PDE at each grid point are approximated from neighbouring values by using Taylor's theorem.

2.1 Taylor's Theorem

Let $U(x)$ have n continuous derivatives over the interval (a, b) . Then for $a < x_0, x_0+h < b$,

$$U(x_0+h) = U(x_0) + hU_x(x_0) + h^2 \frac{U_{xx}(x_0)}{2!} + \dots + h^{n-1} \frac{U_{(n-1)}(x_0)}{(n-1)!} + O(h^n), \quad (2.1)$$

where,

- $U_x = \frac{dU}{dx}, U_{xx} = \frac{d^2U}{dx^2}, \dots, U_{(n-1)} = \frac{d^{n-1}U}{dx^{n-1}}.$
- $U_x(x_0)$ is the derivative of U with respect to x *evaluated* at $x = x_0$.
- $O(h^n)$ is an unknown error term defined in Appendix A.

The usual interpretation of Taylor's theorem says that if we know the value of U and the values of its derivatives at point x_0 then we can write down the equation (2.1) for its value at the (nearby) point x_0+h . This expression contains an unknown quantity which is written in as $O(h^n)$ and pronounced 'order h to the n '. If we discard the term $O(h^n)$ in (2.1) (i.e. *truncate* the right hand side of (2.1)) we get an *approximation* to $U(x_0+h)$. The error in this approximation is $O(h^n)$.

2.2 Taylor's Theorem Applied to the Finite Difference Method (FDM)

In the FDM we know the U values at the grid points and we want to replace partial derivatives in the PDE we are solving by approximations at these grid points. We do this by interpreting (2.1) in another way. In the FDM both x_0 and x_0+h are grid points and $U(x_0)$ and $U(x_0+h)$ are *known*. This allows us to rearrange equation (2.1) to get so-called Finite Difference (FD) approximations to derivatives which have $O(h^n)$ errors. Appendix A explains the meaning of $O(h^n)$ notation.

2.3 Simple Finite Difference Approximation to a Derivative

Truncating (2.1) after the first derivative term gives,

$$U(x_0+h)=U(x_0)+hU_x(x_0)+O(h^2) \quad (2.2)$$

Rearranging (2.2) gives,

$$\begin{aligned} U_x(x_0) &= \frac{U(x_0+h)-U(x_0)}{h} + \frac{O(h^2)}{h} \\ &= \frac{U(x_0+h)-U(x_0)}{h} + O(h) \quad (\text{by A.3.3}) \end{aligned}$$

Neglecting the $O(h)$ term gives,

$$U_x(x_0) \approx \frac{U(x_0+h)-U(x_0)}{h} \quad (2.3)$$

(2.3) is called a *first order* FD approximation to $U_x(x_0)$ since the approximation error = $O(h)$ which depends on the *first* power of h . This approximation is called a *forward* FD approximation since we start at x_0 and step forwards to the point x_0+h . h is called the *step size* ($h > 0$).

2.4 Example: Simple Finite Difference Approximations to a Derivative

This simple example shows that our forward difference approximation works and has the stated order of accuracy. We choose a simple function for U . Let $U(x) = x^2$. We will find the first order forward FD approximation to $U_x(3)$ using step size $h = 0.1$. From (2.3) the general first order forward FD approximation formula is,

$$U_x(x_0) \approx \frac{U(x_0+h)-U(x_0)}{h} \quad (2.4)$$

Substituting for U gives,

$$U_x(x_0) \approx \frac{(x_0+h)^2 - x_0^2}{h}$$

Replacing x_0 by 3 and h by 0.1 gives,

$$U_x(3) \approx \frac{(3+0.1)^2 - 3^2}{0.1} = 6.1$$

The exact answer from basic Calculus is clearly $U_x(3) = 6$ so the error in the approximation is $6.1 - 6 = 0.1$. Repeating the problem with $h = 0.05$ (i.e. half the step size) gives,

$$U_x(3) \approx \frac{(3+0.05)^2 - 3^2}{0.05} = 6.05$$

The error is $6.05 - 6 = 0.05$. The approximation formula (2.4) is first order so the errors should be proportional to h which is seen to be the case: halving the step size results in a halving of the error.

Please click the advert

**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

dongenergy.com/job

Ambitious and curious?

We are looking for young and highly educated engineers and finance students.

Join us at dongenergy.com/job

2.5 Constructing a Finite Difference Toolkit

We now construct common FD approximations to common partial derivatives. For simplicity we suppose that U is a function of only two variables, t and x . We will approximate the partial derivatives of U with respect to x . As t is held constant U is effectively a function of the single variable x so we can use Taylor's formula (2.1) where the ordinary derivative terms are now partial derivatives and the arguments are (t, x) instead of x . Finally we will replace the step size h by Δx (to indicate a change in x) so that (2.1) becomes,

$$U(t, x_0 + \Delta x) = U(t, x_0) + \Delta x U_x(t, x_0) + \frac{\Delta x^2}{2!} U_{xx}(t, x_0) + \dots + \frac{\Delta x^{n-1}}{(n-1)!} U_{(n-1)}(t, x_0) + O(\Delta x^n) \quad (2.5a)$$

Truncating (2.5a) to $O(\Delta x^2)$ gives,

$$U(t, x_0 + \Delta x) = U(t, x_0) + \Delta x U_x(t, x_0) + O(\Delta x^2) \quad (2.5b)$$

Now we derive some FD approximations to partial derivatives. Rearranging (2.5b) gives,

$$\begin{aligned} U_x(t, x_0) &= \frac{U(t, x_0 + \Delta x) - U(t, x_0)}{\Delta x} - \frac{O(\Delta x^2)}{\Delta x} \\ \therefore U_x(t, x_0) &= \frac{U(t, x_0 + \Delta x) - U(t, x_0)}{\Delta x} - O(\Delta x) \end{aligned} \quad (2.6a)$$

Equation (2.6a) holds at any point (t, x_0) . In numerical schemes for solving PDEs we are restricted to a *grid* of discrete x values, x_1, x_2, \dots, x_N , and discrete t levels $0 = t_0, t_1, \dots$. We will assume a constant grid spacing, Δx , in x , so that $x_{i+1} = x_i + \Delta x$. Evaluating Equation (2.6a) for a point, (t_n, x_i) , on the grid gives,

$$U_x(t_n, x_i) = \frac{U(t_n, x_{i+1}) - U(t_n, x_i)}{\Delta x} - O(\Delta x) \quad (2.6b)$$

We will use the common subscript/superscript notation,

$$U_i^n = U(t_n, x_i) \quad (2.6c)$$

so that dropping the $O(\Delta x)$ error term, (2.6b) becomes,

$$U_x(t_n, x_i) \approx \frac{U_{i+1}^n - U_i^n}{\Delta x} \quad (2.6d)$$

(2.6d) is the first order forward difference approximation to $U_x(t_n, x_i)$ that we derived previously in approximation (2.4). We now derive another FD approximation to $U_x(t_n, x_i)$. Replacing Δx by $-\Delta x$ in (2.5b) gives,

$$U(t, x_o - \Delta x) = U(t, x_o) - \Delta x U_x(t, x_o) + O(\Delta x^2) \quad (2.7a)$$

Evaluating (2.7a) at (t_n, x_i) and rearranging as previously gives,

$$U_x(t_n, x_i) \approx \frac{U_i^n - U_{i-1}^n}{\Delta x} \quad (2.7b)$$

(2.7b) is the *first order backward difference* approximation to $U_x(t_n, x_i)$.

Our first two FD approximations are first order in x but we can increase the order (and so make the approximation more accurate) by taking more terms in the Taylor series as follows. Truncating (2.5a) to $O(\Delta x^3)$, then replacing Δx by $-\Delta x$ and subtracting this new expression from (2.5a) and evaluating at (t_n, x_i) gives, after some algebra,

$$U_x(t_n, x_i) \approx \frac{U_{i+1}^n - U_{i-1}^n}{2\Delta x} \quad (2.8)$$

(2.8) is called the *second order central difference* FD approximation to $U_x(t_n, x_i)$.

We could construct even higher order FD approximations to U_x by taking even more terms in the Taylor series but we will stop at second order approximations to first order derivatives.

Many PDEs of interest contain second order (and higher) partial derivatives so we need to derive approximations to them. We will restrict our attention to second order *unmixed* partial derivatives i.e. U_{xx} . Truncating (2.5a) to $O(\Delta x^4)$ gives,

$$U(t, x_0 + \Delta x) = U(t, x_0) + \Delta x U_x(t, x_0) + \frac{\Delta x^2}{2!} U_{xx}(t, x_0) + \frac{\Delta x^3}{3!} U_{xxx}(t, x_0) + O(\Delta x^4) \quad (2.9a)$$

Replacing Δx by $-\Delta x$ in (2.9a) gives,

$$U(t, x_0 - \Delta x) = U(t, x_0) - \Delta x U_x(t, x_0) + \frac{\Delta x^2}{2!} U_{xx}(t, x_0) - \frac{\Delta x^3}{3!} U_{xxx}(t, x_0) + O(\Delta x^4) \quad (2.9b)$$

Adding (2.9a) and (2.9b) gives,

$$U(t, x_0 + \Delta x) + U(t, x_0 - \Delta x) = 2U(t, x_0) + \Delta x^2 U_{xx}(t, x_0) + O(\Delta x^4) \quad (2.10a)$$

Evaluating (2.10a) at (t_n, x_i) and using our discrete notation gives,

$$U_{i+1}^n + U_{i-1}^n = 2U_i^n + \Delta x^2 U_{xx}(t_n, x_i) + O(\Delta x^4) \quad (2.10b)$$

Please click the advert


dongenergy.com/job

**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

Everybody is talking...

...about future energy supply. We are not.
Stop talking and make a career
moving energy forward.

Ambitious engineers and
finance students go to
dongenergy.com/job



Rearranging (2.10b) and dropping the $O(\Delta x^2)$ error term gives,

$$U_{xx}(t_n, x_i) \approx \frac{U_{i+1}^n - 2U_i^n + U_{i-1}^n}{\Delta x^2} \quad (2.11)$$

(2.11) is the *second order symmetric difference* FD approximation to $U_{xx}(t_n, x_i)$. These results are put into Table 2.1 to form a FD approximation toolkit. FD approximations to partial derivatives with respect to t are derived in a similar manner and are included in Table 2.1.

partial derivative	finite difference approximation	type	order
$\frac{\partial U}{\partial x} = U_x$	$\frac{U_{i+1}^n - U_i^n}{\Delta x}$	forward	first in x
$\frac{\partial U}{\partial x} = U_x$	$\frac{U_i^n - U_{i-1}^n}{\Delta x}$	backward	first in x
$\frac{\partial U}{\partial x} = U_x$	$\frac{U_{i+1}^n - U_{i-1}^n}{2\Delta x}$	central	second in x
$\frac{\partial^2 U}{\partial x^2} = U_{xx}$	$\frac{U_{i+1}^n - 2U_i^n + U_{i-1}^n}{\Delta x^2}$	symmetric	second in x
$\frac{\partial U}{\partial t} = U_t$	$\frac{U_i^{n+1} - U_i^n}{\Delta t}$	forward	first in t
$\frac{\partial U}{\partial t} = U_t$	$\frac{U_i^n - U_i^{n-1}}{\Delta t}$	backward	first in t
$\frac{\partial U}{\partial t} = U_t$	$\frac{U_i^{n+1} - U_i^{n-1}}{2\Delta t}$	central	second in t
$\frac{\partial^2 U}{\partial t^2} = U_{tt}$	$\frac{U_i^{n+1} - 2U_i^n + U_i^{n-1}}{\Delta t^2}$	symmetric	second in t

Table 2.1 Finite Difference Toolkit for Partial Derivatives

The above FD toolkit can be used to create a finite difference scheme (FDS) to obtain the approximate solution of a large number of PDEs simply by replacing each partial derivative by an appropriate FD approximation.

2.6 Simple Example of a Finite Difference Scheme

We construct a simple FDS to find an approximate solution of a simple PDE. This PDE will be studied in more detail in Chapter 4. For now it suffices to generate a simple FDS to provide motivation for further study. The 1D linear advection equation is,

$$U_t + v U_x = 0, \quad (2.12a)$$

where the independent variables are t (time) and x (space). x is restricted to the finite interval $[p, q]$ which is called the *computational domain*. v is a constant and the dependent variable, $U = U(t, x)$. In addition to the PDE, we need *initial conditions* for U . Let the initial conditions be,

$$U(0, x) = f(x), \quad p \leq x \leq q. \quad (2.12b)$$

i.e. the initial value of U is given for every x value in the computational domain by a *known* function $f(x)$.

A *solution* to (2.12a, 2.12b) is a function $U = U(t, x)$ which satisfies the PDE (2.12a) at all points x in the computational domain and all times t and the *initial conditions* (2.12b). $U(t, x)$, the exact solution of (2.12a,b), is defined at an *infinite* number of values of the independent variables t and x . We will create a FDS to approximate U at a *finite* set of values of the independent variables. The approximate values of U on this finite set will be denoted by u . We proceed in stages.

2.6.1 Step 1: Spatial Discretisation

The computational domain (Figure 2.1) contains an infinite number of x values so first we must replace them by a finite set. This process is called spatial discretisation.

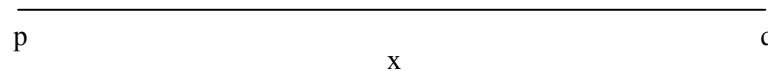


Figure 2.1: 1D computational domain.

For simplicity the computational domain is replaced by a grid of N *equally spaced* grid points. Starting with the first grid point at $x = p$ and ending with the last grid point at $x = q$, the constant grid spacing, Δx , is,

$$\Delta x = \frac{(q-p)}{(N-1)} \quad (2.13a)$$

The values of x in the discretised computational domain are indexed by subscripts to give,

$$x_1 = p, x_2 = p + \Delta x, \dots, x_i = p + (i-1) \Delta x, \dots, x_N = p + (N-1) \Delta x = q. \quad (2.13b)$$

Since the grid spacing is constant,


$$x_{i+1} = x_i + \Delta x \quad (2.13c)$$

The discretised computational domain is shown in Figure 2.2:



Figure 2.2 Discretised computational domain.

Please click the advert



dongenergy.com/job


Dear highly educated engineering and finance students,

if you are driven, ambitious, open-minded and focused - we have a challenge for you. Actually, the greatest challenge in the world. Curious? Visit dongenergy.com/job

Best wishes

DONG Energy





Fixing t at $t = t_n$ we approximate the spatial partial derivative, U_x , in (2.12a) at each point (t_n, x_i) using the forward difference formula from the toolkit in Table 2.1 to give,

$$U_x(t_n, x_i) \approx \frac{U_{i+1}^n - U_i^n}{\Delta x} \quad (2.14)$$

Replacing U_x in (2.1a) by its approximation (2.14) gives,

$$U_t + v \frac{U_{i+1}^n - U_i^n}{\Delta x} = 0 \quad (2.15)$$

(2.15) is said to be in *semi-discrete* form since only the spatial derivative has been discretised.

Note: The *grid* is also called the *mesh* and the operation of discretising the computational domain is called *gridding* or *meshing*.

2.6.2 Step 2: Time Discretisation

Fixing x at $x = x_i$ we approximate the temporal partial derivative, U_t , in (2.12a) at each point (t_n, x_i) using the first order forward difference formula from the toolkit in Table 2.1 (where Δt is the spacing between time levels) to give,

$$U_t \approx \frac{U_i^{n+1} - U_i^n}{\Delta t} \quad (2.16)$$

On substituting (2.16) for U_t , 2.15 becomes,

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} + v \frac{U_{i+1}^n - U_i^n}{\Delta x} = 0 \quad (2.17a)$$

which rearranges to give,

$$U_i^{n+1} = U_i^n - \frac{v\Delta t}{\Delta x} (U_{i+1}^n - U_i^n) \quad (2.17b)$$

Equation (2.17b) is an example of a FDS to approximate the solution of the PDE (2.12a). (2.17b) is a so-called time-marching scheme which enables U values at time level $n+1$ to be approximated from U values at the previous time level n . Since all U values are only known exactly at the initial time level (2.17b) is rewritten as,

$$u_i^{n+1} = u_i^n - \frac{v\Delta t}{\Delta x} (u_{i+1}^n - u_i^n) \quad (2.18)$$

where u_i^n is an approximation to $U_i^n = U(t_n, x_i)$.

Notes:

1. (2.18) holds for each grid point x_i , $i = 1, 2, \dots, N$. $u(t_n, x_i)$ is a numerical approximation to $U(t_n, x_i)$, the *exact* solution of the PDE (2.1a).
2. $u(0, x_i) = U(0, x_i)$ but this will not be true in general for later times.
3. u values on the right hand side of (2.18) are all at time t_n whereas on the left hand side u values are all at the next time level $t_n + \Delta t = t_{n+1}$.
4. (2.18) is an example of a **time-marching** scheme in that (known) data for each grid point at time t_n is used to find data at each grid point at the future time $t_n + \Delta t$. This is called an *iteration* of the scheme. After an iteration of the scheme all u values at each grid point are known at time $t_n + \Delta t$. These new values can be used as known data for another iteration of the scheme to give data for each grid point at the next time level. This process can be repeated until the required future time is attained. Iterations of (2.18) are performed by a computer program.
5. The errors in approximating the spatial and temporal derivatives which are used to get (2.18) are $O(\Delta x)$ and $O(\Delta t)$ respectively and so (2.18) is said to be (formally) first order in space (x) and first order in time (t).
6. The grid spacing, Δx , was determined by *choosing* the number of grid points, N . A larger N gives a smaller Δx and a (hopefully) more accurate solution as spatial derivatives are more accurately approximated. However as N increases compute time increases so there is a trade off between accuracy and speed.

7. The time step, Δt , is for the moment, chosen arbitrarily. However a smaller time step will mean that more iterations of (2.18) are needed to reach a stated future time which will obviously increase the compute time. In addition, since the result of each iteration is an approximation to the required solution, more iterations could cause the build up of more error. We will see later (Chapter 4 and Appendix C) that there is often a limit to the maximum size of a time step.
8. (2.18) is said to be an *explicit* method since the value of u at the next time level is given by an explicit formula for each grid point.
9. Later we will see that (2.18) doesn't work for $v > 0$! There is more to FDS than meets the eye!

2.7 Pen and Paper Calculation (very important)

In practice numerical schemes are implemented by writing then running a computer program. Before doing this it is extremely useful to work through a pen and paper calculation for two reasons:

1. To check understanding of the scheme.
2. To be able to check results from the computer program against pen and paper results (verification).

Please click the advert



May we offer you one of the world's greatest challenges?
In all humility.

We are looking for highly educated engineers and finance students.

Join us at dongenergy.com/job

dongenergy.com/job




Download free ebooks at bookboon.com

Since we are doing a pen and paper calculation we will only use a small number of grid points. In a computer implementation of the calculation we would use many more grid points (hundreds, thousands, perhaps millions) for accuracy. We now set up a simple problem.

Let $p=0$ and $q=100$, $v=0.5$ and let the initial conditions be,

$$U(0, x) = \begin{cases} e^{-0.01(x-45)^2}, & 20 \leq x \leq 70, \\ 0, & \text{elsewhere} \end{cases} \quad (2.19)$$

Note that there is nothing special or realistic about these initial conditions.

Let the (small) number of grid points be $N = 11$. Then by (2.13a),

$$\Delta x = \frac{(100-0)}{(11-1)} = 10.$$

The subscripts for the grid values go from 1 to 11 and are entered into the first row of Table 2.2. The actual x values of the corresponding grid points are 0, 10, 20, 30, ..., 90, 100 and are entered into the second row of Table 2.2. It remains to choose the time step Δt . Quite arbitrarily let $\Delta t = 3$. Then (2.18) becomes,

$$u_i^{n+1} = u_i^n - 0.15(u_{i+1}^n - u_i^n) \quad (2.20)$$

(2.20) is a FDS for calculating the solution to our problem at the next time level using data at the current time level. We start at time $t_0=0$, i.e. $n = 0$, hence (2.20) becomes,

$$u_i^1 = u_i^0 - 0.15(u_{i+1}^0 - u_i^0) \quad (2.21)$$

Time level zero corresponds to the initial conditions. The initial u values are needed at the computational grid points. i.e. we need to know u_i^0 for $i = 1, 2, \dots, 11$. In general u_i^n is only an approximation to the exact solution $U(t_n, x_i)$ but at time $t_0=0$, $u_i^0 = U(0, x_i)$.

Using the initial conditions we get,

$$u_i^0 = U(0, x_i) = \begin{cases} e^{-0.01(x_i-45)^2}, & 20 \leq x \leq 70, \\ 0, & \text{elsewhere} \end{cases} \quad (2.22)$$

Evaluating (2.22) at a few grid points gives,

$$u_1^0 = U(0, x_1) = U(0, 0) = 0, \quad u_5^0 = U(0, x_5) = U(0, 40) = e^{-0.01(40-45)^2} = 0.7788, \text{ etc.}$$

These *initial values*, u_i^0 , are entered into the third row of Table 2.2.

i	1	2	3	4	5	6	7	8	9	10	11	12
x_i	0	10	20	30	40	50	60	70	80	90	100	110
u_i^0	0	0	0.0019	0.1054	0.7788				0	0	0	0
u_i^1	0	-0.00029	-0.01363								0	
u_i^2												

Table 2.2 Implementation of Finite difference Scheme (2.20)

Having set up the initial data we use (2.21) to find u_i^1 i.e. the u values at the next time level at each grid point. These new values are entered into the fourth row of Table 2.2. The first few values are found from the following:

Putting $i = 1$ into (2.21) gives:

$$u_1^1 = u_1^0 - 0.15(u_2^0 - u_1^0) = 0 - 0.15(0 - 0) = 0$$

Putting $i = 2$ into (2.21) gives:

$$u_2^1 = u_2^0 - 0.15(u_3^0 - u_2^0) = 0 - 0.15(0.0019 - 0) = -0.00029$$

Putting $i = 3$ into (2.21) gives:

$$u_3^1 = u_3^0 - 0.15(u_4^0 - u_3^0) = 0.0019 - 0.15(0.1054 - 0.0019) = -0.01363$$

etc.

However there is a problem! We are using *forward* differences for the spatial derivatives, i.e. to approximate the spatial derivative at a grid point we need the function value at the next grid point. This is OK for *interior* grid points but when we come to the last point on the right hand *boundary* of the computational domain (point index $i = 11$ corresponding to $q = 100$) we need data at point index $i = 12$ which we *DO NOT HAVE!* In this case we have to invent a fictitious point with an associated function value. These points are called *ghost points* and the function values, *ghost values*. How we invent ghost values is based on the boundary conditions that define the particular problem we are solving. This topic is discussed in more detail in Appendix B. In our case we will assume that ghost point is at $x_{12} = 110$ and that the u value at this point takes the same value as its value at the nearest interior point (i.e. x_{11}) *at all times*.

$$\text{i.e. } u_{12}^n = u_{11}^n, n = 0, 1, 2, \dots \quad (2.23)$$

Please click the advert



Ambitious and curious?

We are looking for young and highly educated engineers and finance students.

Join us at dongenergy.com/job

dongenergy.com/job




Download free ebooks at bookboon.com

Now we can use our numerical scheme to calculate u_{11}^1 .

Putting $i = 11$ into (2.21) gives:

$$u_{11}^1 = u_{11}^0 - 0.15(u_{12}^0 - u_{11}^0) = 0 - 0.15(0 - 0) = 0$$

This completes the *first iteration* of the FDS (2.21) and row 4 of Table 2.2 and we have found the approximate solution at each grid point at

$$t = t_1 = \Delta t = 3.$$

Once all the values of u_i^1 have been calculated the *same* procedure can be used to find u_i^2 by a *second iteration*: the time indices in equation (2.21) are increased by 1 to give,

$$u_i^2 = u_i^1 - 0.15(u_{i+1}^1 - u_i^1) \quad (2.24)$$

and we repeat the previous process to fill in row 5 of Table 2.2 which is the approximate solution at each grid point at $t = t_2 = 2\Delta t = 6$.

By successive iteration of (2.20), the solution can be found at each grid point at future time levels. At each iteration we use the known data at a particular time level to obtain the unknown data at the next time level. Of course this iterative procedure can be automated. A computer program for this scheme is given on the website.


2.8 Exercise 2a

1. $U(x) = x^2$. Find the first order *backward* difference approximations to $U_x(3)$ using: a) $h = 0.1$, b) $h = 0.05$, c) $h = 0.025$.
2. Repeat Q1a), b), c) using the central FD approximation and show that it is second order accurate.
3. Following the text, derive the central FD approximation to a first order spatial derivative.
4. Following the text, derive the symmetric FD approximation to a second order spatial derivative.
5. Using exactly similar working for the spatial approximations derive all the time derivative approximation in Table 2.1.
6. Complete Table 2.2 by carrying out pen and paper calculations (tedious but very important for checking your numerical algorithm).

2.9 Exercise 2b

1. From the website download a computer program to implement (2.20) for the given problem.
2. Read each line of the program and make sure you understand it. This program will be used as the basis for other programs later on.
3. Verify the program by comparison to Table 2.2.

Please click the advert



The advertisement is enclosed in a rectangular frame. On the left side, outside the frame, is the vertical text "Please click the advert". Inside the frame, the top left corner features a red square logo with the text "AMBITIOUS PEOPLE" in white, with "MOVING ENERGY FORWARD" in smaller white text below it. The top right corner contains the URL "dongenergy.com/job" in a small, grey font. A large, light-grey speech bubble with a drop shadow is centered in the frame. Inside the bubble, the text "Everybody is talking..." is written in a large, black, sans-serif font. Below this, in a smaller black font, is the text "...about future energy supply. We are not. Stop talking and make a career moving energy forward." At the bottom of the bubble, in an even smaller grey font, is the text "Ambitious engineers and finance students go to dongenergy.com/job". In the bottom right corner of the frame, the "DONG energy" logo is displayed, with "DONG" in a large, bold, red font and "energy" in a smaller, black font below it.

AMBITIOUS
PEOPLE
MOVING ENERGY FORWARD

dongenergy.com/job

Everybody
is talking...

...about future energy supply. We are not.
Stop talking and make a career
moving energy forward.

Ambitious engineers and
finance students go to
dongenergy.com/job

DONG
energy

3. Elliptic Equations

3.1 Introduction

Elliptic PDEs form a class of PDEs that may be used to model *steady state* problems (i.e. the dependent variable remains constant over time). Solutions of elliptic PDEs are over closed regions on which boundary values are given in some way. These boundary values determine the solution of the PDE in the interior of the region. The two most widely used elliptic PDEs are Laplace's equation and Poisson's equation.

In 2D, Laplace's equation is:

$$U_{xx} + U_{yy} = 0 \quad (3.1)$$

Laplace's equation may be used to model a wide range of phenomena including steady state groundwater flow and temperature distribution over a region. Additionally Laplace's equation can describe 'potential flow' which can be used in a simplified description of water flow amongst other things.

In 2D, Poisson's equation is:

$$U_{xx} + U_{yy} = f(x, y). \quad (3.2)$$

Poisson's equation may also be used to model a wide range of phenomena including gravitational fields, stress patterns and simplified viscous flow.

The above PDEs can only be solved analytically for simple situations so we need to use numerical methods to obtain approximate solutions for cases of practical interest. In the following we focus on Laplace's equation since it is simpler than Poisson's equation and the techniques carry over easily. For simplicity the computational domain will be rectangular.

3.2 Finite Difference Method for Laplace's Equation

The computational domain is discretised using constant grid spacings of Δx and Δy in the x and y directions respectively. Grid points are indexed by (i, j) in the usual way and the approximate value of U at grid point (i, j) is denoted by $u_{i,j}$. Figure 3.1 shows a rectangular grid with M and N grid points in the x and y directions respectively. u is *known* ($=U$) at the boundary grid points. It is required to find u at the interior grid points.

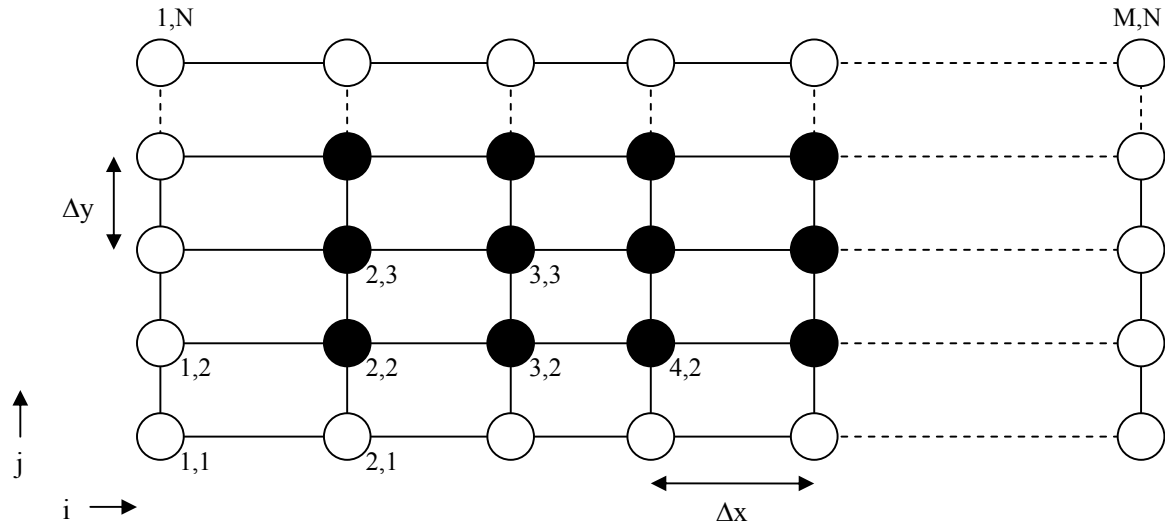


Figure 3.1: Computational grid showing interior grid points (black) and boundary grid points (white)

Each partial derivative in Equation (3.1) is replaced by a symmetric FD approximation from our tool kit (Table 2.1) to give,

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} = 0 \quad (3.3a)$$

Letting $b = \Delta x / \Delta y$, (3.3a) can be rewritten to give,

$$u_{i,j} = \frac{u_{i+1,j} + u_{i-1,j} + b^2 u_{i,j+1} + b^2 u_{i,j-1}}{2(1+b^2)} \quad (3.3b)$$

Equation (3.3b) shows that $u_{i,j}$ depends on its 4 surrounding values. This is called a 5-point stencil. Sometimes ‘compass notation’ is used and (3.3b) becomes,


$$u_o = \frac{u_E + u_W + b^2 u_N + b^2 u_S}{2(1+b^2)} \quad (3.3c)$$

where o denotes the current grid point and subscripts N, S, E and W denote its north, south, east and west neighbours respectively.

Notes:

1. Using the indexing system in Figure 3.1 the *unknown* value of u nearest to the bottom left hand corner of the computational domain is $u_{2,2}$ and the *unknown* value nearest to the top right hand corner of the computational domain is $u_{M-1,N-1}$.
2. In an $M \times N$ grid there will be $(M-2) \times (N-2)$ unknown interior values of $u_{i,j}$ which may be a very large number.
3. Assuming that the boundary values of u are known then (3.3b) gives a system of $(M-2) \times (N-2)$ linear equations for $u_{i,j}$ in $(M-2) \times (N-2)$ unknowns.

Please click the advert





Dear highly educated engineering and finance students,

if you are driven, ambitious, open-minded and focused - we have a challenge for you. Actually, the greatest challenge in the world. Curious? Visit dongenergy.com/job

Best wishes

DONG Energy

dongenergy.com/job

3.3 Setting up the Equations

There are two basic methods of solving for $u_{i,j}$. Both methods set up a system of linear equations as follows. Letting $c = 1/(2(1+b^2))$,

$d = b^2/(2(1+b^2))$, rearranging Equation (3.3b) gives,

$$u_{i,j} = cu_{i-1,j} + du_{i,j-1} + du_{i,j+1} + cu_{i+1,j} \quad (3.4)$$

Evaluating Equation (3.4) at successive grid points starting at 2,2 and sweeping along the *rows first* gives,

$$\begin{array}{rclcl} u_{2,2} & = & cu_{1,2} & + du_{2,1} & + du_{2,3} & + cu_{3,2} \\ u_{3,2} & = & cu_{2,2} & + du_{3,1} & + du_{3,3} & + cu_{4,2} \\ \vdots & & \vdots & \vdots & \vdots & \vdots \\ u_{M-1,2} & = & cu_{M-2,2} & + du_{M-1,1} & + du_{M-1,3} & + cu_{M,2} \\ u_{2,3} & = & cu_{1,3} & + du_{2,2} & + du_{2,4} & + cu_{3,3} \\ \vdots & & \vdots & \vdots & \vdots & \vdots \\ u_{M-1,N-1} & = & cu_{M-2,N-1} & + du_{M-1,N-2} & + du_{M-1,N} & + cu_{M,N-1} \end{array} \quad (3.5)$$

The boundary values $u_{1,2}$, $u_{2,1}$, $u_{M,2}$, $u_{1,3}$, etc. are *known*. In each equation the known u values are moved to the left hand side and the unknown u values moved to the right hand side (and positioned to preserve the ordering). For example in the first equation $u_{1,2}$ and $u_{2,1}$ are known (being boundary values) so the equation becomes,

$$-cu_{1,2} - du_{2,1} = -u_{2,2} + du_{2,3} + cu_{3,2}$$

The result is that Equations (3.5) can be written as a single matrix equation,

$$\underline{d} = A \underline{u} \quad (3.6a)$$

where,

\underline{d} is an $(M-2)(N-2)$ by 1 matrix of known constants,

A is a $(M-2)(N-2)$ by $(M-2)(N-2)$ matrix of known coefficients and

\underline{u} is a $(M-2)(N-2)$ by 1 matrix of unknowns and

$$\underline{u} = (u_{2,2} \ u_{3,2} \ \dots \ u_{(M-1),2} \ u_{2,3} \ u_{3,3} \ \dots \ u_{(M-1),3} \ \dots \dots \ u_{(M-1),(N-1)})^T.$$

The solution to (3.6a) may be written symbolically as,

$$\underline{u} = A^{-1} \underline{d} \quad (3.6b)$$

As A may be very large we must study efficient ways of finding \underline{u} .

3.4 Grid Convergence

Before looking at solution methods we make the following important point *which applies to all grid-based numerical schemes*. Clearly the accuracy of the numerical results depends on the size of the computational grid. Ideally we would like a *grid converged solution* i.e. a solution that does not change significantly when more grid points are used. Grid converged solutions can be studied formally by grid convergence indices but for us it is enough to compare numerical solutions with more and more grid points until there is no significant difference. A good strategy is to implement a scheme using a small number of grid points (for verification of the program) then successively double the number of grid points until numerical results don't change perceptibly. In this way we can be confident that inaccuracies in the numerical solution are not caused by the grid. We present other questions about the accuracy of schemes in Appendix C that is best read later.

3.5 Direct Solution Method

One way to achieve a solution of (3.6a) is by using standard Gaussian elimination. This is a so-called direct method.

Note:

We state again that in our convention the grid is traversed from left to right starting at the bottom, i.e. we start at position (2, 2) go along the whole row, repeat for the next row etc. and eventually end at position (M-1, N-1).

The direct method is illustrated by an example on a small grid (much too small for a real computation but useful for verification of a program). Consider the following 5 by 4 grid of u values where $\Delta x = \Delta y$,

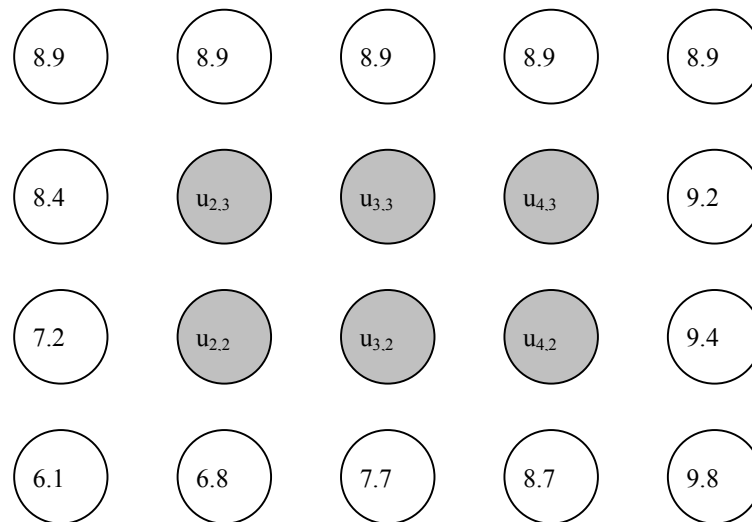


Figure 3.2 Boundary (white) and interior (shaded) u values on a 5×4 grid.

Please click the advert



May we offer you one
of the world's greatest
challenges?
In all humility.

We are looking for highly
educated engineers and
finance students.

Join us at dongenergy.com/job

dongenergy.com/job






The u values at the boundary grid points are known, e.g. $u_{1,1} = 6.1$. We want to find the unknown values of u at the interior grid points, i.e. $u_{2,2}$, $u_{3,2}$, $u_{4,2}$, $u_{2,3}$, $u_{3,3}$, $u_{4,3}$. Since $\Delta x = \Delta y$, Equation (3.4) simplifies to,

$$u_{i,j} = (u_{i-1,j} + u_{i,j-1} + u_{i,j+1} + u_{i+1,j}) / 4 \quad (3.7a)$$

In compass notation this is written,

$$u_o = (u_w + u_s + u_N + u_E) / 4 \quad (3.7b)$$

Starting on the second row, evaluation of (3.7a) at grid point (2, 2) gives,

$$u_{2,2} = (u_{1,2} + u_{2,1} + u_{2,3} + u_{3,2}) / 4 = (7.2 + 6.8 + u_{2,3} + u_{3,2}) / 4 \quad (3.8a)$$

Evaluation of (3.7a) at grid point (3, 2) gives,

$$u_{3,2} = (u_{2,2} + u_{3,1} + u_{3,3} + u_{4,2}) / 4 = (u_{2,2} + 7.7 + u_{3,3} + u_{4,2}) / 4 \quad (3.8b)$$

Evaluation at grid point (4, 2) gives,

$$u_{4,2} = (u_{3,2} + u_{4,1} + u_{4,3} + u_{5,2}) / 4 = (u_{3,2} + 8.7 + u_{4,3} + 9.4) / 4 \quad (3.8c)$$

Moving to the next (third) row, evaluation of (3.7a) at grid point (2, 3) gives,

$$u_{2,3} = (u_{1,3} + u_{2,2} + u_{2,4} + u_{3,3}) / 4 = (3.4 + u_{2,2} + 8.9 + u_{3,3}) / 4 \quad (3.8d)$$

Evaluation of (3.7a) at grid point (3, 3) gives,

$$u_{3,3} = (u_{2,3} + u_{3,2} + u_{3,4} + u_{4,3}) / 4 = (u_{2,3} + u_{3,2} + 8.9 + u_{4,3}) / 4 \quad (3.8e)$$

Finally evaluation of (3.7a) at grid point (4, 3) gives (fill in this for yourself), (3.8f)

These equations can be written out in standard form as 6 simultaneous linear equations in which the order of the unknown variables is as above i.e. $u_{2,2}$, $u_{3,2}$, $u_{4,2}$, $u_{2,3}$, $u_{3,3}$, $u_{4,3}$.

Rewriting Equation (3.8a) in standard form gives,

$$-14/4 = -1 u_{2,2} + (1/4) u_{3,2} + 0 u_{4,2} + (1/4) u_{2,3} + 0 u_{3,3} + 0 u_{4,3}$$

Rewriting Equation (3.8b) in standard form gives,

$$-7.7/4 = (1/4) u_{2,2} - 1 u_{3,2} + (1/4) u_{4,2} + 0 u_{2,3} + (1/4) u_{3,3} + 0 u_{4,3}$$

Equations (3.8c-f) are rewritten similarly. The 6 simultaneous linear equations are then written as the matrix equation,

$$\begin{pmatrix} -14/4 \\ -7.7/4 \\ \vdots \end{pmatrix} = \begin{pmatrix} -1 & 1/4 & 0 & 1/4 & 0 & 0 \\ 1/4 & -1 & 1/4 & 0 & 1/4 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} u_{2,2} \\ u_{3,2} \\ u_{4,2} \\ u_{2,3} \\ u_{3,3} \\ u_{4,3} \end{pmatrix} \quad (3.9)$$

which can be solved by standard Gaussian elimination. For large systems this isn't very efficient so we will study efficient iterative methods next.

3.6 Exercise 3a

1. Finish labelling all the grid points in Figure 3.1.
2. Check that (3.3a) is correct by referring back to the FD toolkit in Table 2.1.
3. Derive (3.3b) from (3.3a).
4. Given that $\Delta x = 4$ and $\Delta y = 2$, write down the expression for $u_{3,4}$ using both subscript and compass notation.
5. Write down Equation (3.8f) in the space provided in the notes.
6. a) Write down Equations (3.8c-f) in the standard way.
b) Hence complete the matrix equation (3.9).
7. Solve (3.9) by Gaussian elimination and check your solution by back substitution.

3.7 Iterative Solution Methods

Equation (3.9) can be expressed as,

$$A \underline{u} = \underline{b} \quad (3.10)$$

For practical problems A is likely to be a large matrix which makes the direct solution of (3.10) computationally inefficient. More efficient methods use iterative approaches where an initial estimate for \underline{u} is updated to form a better estimate. The process is repeated until the distance between successive estimates is less than some pre-defined tolerance (assuming that the iterative process converges to the solution). Since the output from each iteration is a vector of \underline{u} values we define what is meant by distance between vectors as follows.

Please click the advert

dongenergy.com/job

**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

Ambitious and curious?

We are looking for young and highly educated engineers and finance students.

Join us at dongenergy.com/job

3.7.1 Distance between Vectors

Let $\underline{x} = (x_1, x_2, \dots, x_N)$ and $\underline{y} = (y_1, y_2, \dots, y_N)$ be two vectors in \mathbb{R}^N . The distance between \underline{x} and \underline{y} is denoted by $d(\underline{x}, \underline{y})$ and defined by,

$$d(\underline{x}, \underline{y}) = \max_i (|x_i - y_i|) = \|\underline{x} - \underline{y}\|_\infty \quad (3.11)$$

(this is sometimes called the ‘infinity norm’).

$$\text{e.g. } \underline{x} = (1, 3, 5), \underline{y} = (4, 5, 3), \|\underline{x} - \underline{y}\|_\infty = \max(|1-4|, |3-5|, |5-3|) = 3.$$

In the following we illustrate three variants of the iterative approach with respect to the test problem of Figure 3.2 where $\Delta x = \Delta y$.

When $\Delta x = \Delta y$, our 5-point formula is,

$$u_{i,j} = (u_{i-1,j} + u_{i,j-1} + u_{i,j+1} + u_{i+1,j}) / 4 \quad (3.12)$$

3.8 Jacobi Iteration

We introduce the *iteration index* as a superscript, m , and write (3.12) as the Jacobi formula (also called the point-Jacobi formula),

$$u_{i,j}^{m+1} = (u_{i-1,j}^m + u_{i,j-1}^m + u_{i,j+1}^m + u_{i+1,j}^m) / 4 \quad (3.13)$$

(Note that this formula assumes $\Delta x = \Delta y$).

For each interior grid point (i, j) , $u_{i,j}$ at the next iteration $(m+1)$ is found from (3.13). Once an iteration has been completed for all interior grid points we compute the distance between vectors \underline{u}^{m+1} and \underline{u}^m . If,

$$\|\underline{u}^{m+1} - \underline{u}^m\|_\infty < \text{tol}, \quad (3.14)$$

where tol is a pre-defined tolerance, the iterations terminate and the solution to (3.10) is \underline{u}^{m+1} otherwise the iterations continue.

To start the iteration (at index 0) values must be given for the unknown interior values of $u_{i,j}$. These values can be set to zero or interpolated from the known boundary values. The following example of Jacobi iteration applies to the previous ‘test problem’ with all interior starting values set to zero and a tolerance of 0.5×10^{-3} .

3.8.1 First Iteration

Equation (3.13) with $m=0$ gives,

$$u_{i,j}^1 = (u_{i-1,j}^0 + u_{i,j-1}^0 + u_{i,j+1}^0 + u_{i+1,j}^0) / 4$$

Evaluating at each interior grid point gives,

$$u_{2,2}^1 = (u_{1,2}^0 + u_{2,1}^0 + u_{2,3}^0 + u_{3,2}^0) / 4 = (7.2 + 6.8 + 0 + 0) / 4 = 3.500$$

$$u_{3,2}^1 = (u_{2,2}^0 + u_{3,1}^0 + u_{3,3}^0 + u_{4,2}^0) / 4 = (0 + 7.7 + 0 + 0) / 4 = 1.925, \text{ similarly}$$

$$u_{4,2}^1 = 4.525, u_{2,3}^1 = 4.325, u_{3,3}^1 = 2.225, u_{4,3}^1 = 4.525. \quad (3.15)$$

Having completed the first iteration we test for convergence. i.e. we measure how close \underline{u}^0 is to \underline{u}^1 .

$$\underline{u}^0 = (0, 0, 0, 0, 0, 0), \underline{u}^1 = (3.5, 1.925, 4.525, 4.325, 2.225, 4.525),$$

therefore,

$$\|\underline{u}^1 - \underline{u}^0\|_{\infty} = \max(|3.5 - 0|, |1.925 - 0|, |4.525 - 0|, |4.325 - 0|, |2.225 - 0|, |4.525 - 0|)$$

This is greater than the specified tolerance of 0.5×10^{-3} so the iteration is repeated to find \underline{u}^2 etc. Eventually (we hope!) after p iterations,

$$\|\underline{u}^p - \underline{u}^{p-1}\|_{\infty} < 0.5 \times 10^{-3}$$

and the iterative procedure terminates. The solution is \underline{u}^p .

A program to implement Jacobi iteration is available from the website.

3.9 Gauss-Seidel Iteration

This is a potentially more efficient version of Jacobi iteration. We note that in Equation (3.13) some of the updated $u_{i,j}$ values are *already* available for use in the iteration formula. In our way of traversing the grid when we reach grid position (i, j) we have *already* updated $u_{i-1,j}$ and $u_{i,j-1}$ therefore we can use these values in the 5-point formula which becomes,

$$u_{i,j}^{m+1} = (u_{i-1,j}^{m+1} + u_{i,j-1}^{m+1} + u_{i,j+1}^m + u_{i+1,j}^m) / 4 \quad (3.16)$$

(Note that this formula assumes that $\Delta x = \Delta y$).

This is called the Gauss-Seidel formula (also called point-Gauss-Seidel) and the implementation is the same as for Jacobi. As an example of Gauss-Seidel iteration we repeat the previous problem using starting u values of zero (and where we don't yet have an updated u value we use its current value).


dongenergy.com/job

**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

Everybody is talking...

...about future energy supply. We are not.
Stop talking and make a career
moving energy forward.

Ambitious engineers and
finance students go to
dongenergy.com/job



3.9.1 First Iteration

Equation (3.16) with $m=0$ gives,

$$u_{i,j}^1 = (u_{i-1,j}^1 + u_{i,j-1}^1 + u_{i,j+1}^0 + u_{i+1,j}^0) / 4$$

Evaluating at each interior grid point gives,

$$u_{2,2}^1 = (u_{1,2}^1 + u_{2,1}^1 + u_{2,3}^0 + u_{3,2}^0) / 4.$$

Note that we don't have $u_{1,2}^1, u_{2,1}^1$ yet so we use $u_{1,2}^0, u_{2,1}^0$ to give,

$$= (u_{1,2}^0 + u_{2,1}^0 + u_{2,3}^0 + u_{3,2}^0) / 4 = (7.2 + 6.8 + 0 + 0) / 4 = 3.500.$$

$$u_{3,2}^1 = (u_{2,2}^1 + u_{3,1}^1 + u_{3,3}^0 + u_{4,2}^0) / 4$$

Note that we now have $u_{2,2}^1$ but not $u_{3,1}^1$, so we use $u_{3,1}^0$ to give,

$$= (u_{2,2}^1 + u_{3,1}^0 + u_{3,3}^0 + u_{4,2}^0) / 4 = (3.5 + 7.7 + 0 + 0) / 4 = 2.8$$

$$u_{4,2}^1 = (u_{3,2}^1 + u_{4,1}^1 + u_{4,3}^0 + u_{5,2}^0) / 4$$

Note that we now have $u_{3,2}^1$ but not $u_{4,1}^1$, so we use $u_{4,1}^0$ to give,

$$= (u_{3,2}^1 + u_{4,1}^0 + u_{4,3}^0 + u_{5,2}^0) / 4 = (2.8 + 8.7 + 0 + 9.4) / 4 = 5.225$$

(3.17)

The remaining values are calculated similarly.

Notes:

1. We are assuming that the iterative procedures converge – this needs further study.
2. For simplicity we have taken $\Delta x = \Delta y$. It is a simple matter to generalise the iterative formulae for the case of different grid spacing in the x and y directions (start from Equation (3.4)).

3. It is quite tricky to write a general program for an M by N mesh. The difficult part is to map the $(M-2) \times (N-2)$ array of interior grid values to a system of $(M-2)(N-2)$ linear equations in $(M-2)(N-2)$ unknowns.

3.10 Exercise 3b

1. Check the direct solution to (3.9) by using Scilab (or Matlab).
2. Find $\|\underline{x} - \underline{y}\|_\infty$ for $\underline{x} = (1, 3, 5, 8, 4, -2, 4, 0, 2)$, $\underline{y} = (2, -4, 4, 1, 4, 2, -3, 0, 2)$.
3. Using pen and paper perform a second iteration for each unknown for the Jacobi test problem (equations (3.15)).
4. Write a program to implement Jacobi iteration for the test problem above and verify by comparison to your pen and paper calculations (and by comparison to the direct solution). Comparison programs can be found from the website.
- 5a. Using pen and paper complete the first iteration of the Gauss-Seidel calculations for the test problem.
- 5b. Perform a second iteration.
- 6a. Write a program to implement Gauss-Seidel iteration for the test problem using zero starting values and verify by comparison to 5a,b.
- 6b. For a given tolerance compare the number of iterations taken using Gauss-Seidel and Jacobi methods for the test problem.
7. Write a Jacobi iterative solver for a general rectangular grid and validate it on the test problem (a program is available on the website for comparison).
8. Repeat Q7. using a Gauss-Seidel iterative solver (a program is available on the website for comparison).

3.11 Successive Over Relaxation (SoR) Method

The idea behind this method is that in an iterative formula the point value at the new iteration depends on the old point value plus some error (residual) at that point.

$$\text{i.e.} \quad u_{i,j}^{m+1} = u_{i,j}^m + R_{i,j}^m \quad (3.18)$$

$R_{i,j}^m$ is the difference between successive iterates of $u_{i,j}$ and is called the *residual*. It might be possible to speed up the convergence of the iterative scheme by weighting the residual on the right hand side of (3.18) appropriately. We write,

$$u_{i,j}^{m+1} = u_{i,j}^m + w R_{i,j}^m \quad (3.19)$$


w is called a *relaxation* parameter. For $0 < w < 1$ (3.19) corresponds to under-relaxation and for $1 < w < 2$ (3.19) corresponds to over-relaxation. This idea is applied to improve the point-Gauss-Seidel method. The point-Gauss-Seidel iteration formula (for $\Delta x = \Delta y$) is,

$$u_{i,j}^{m+1} = (u_{i-1,j}^{m+1} + u_{i,j-1}^{m+1} + u_{i,j+1}^m + u_{i+1,j}^m) / 4 \quad (3.20)$$

which can be re-written as,

$$u_{i,j}^{m+1} = u_{i,j}^m + (u_{i-1,j}^{m+1} + u_{i,j-1}^{m+1} - 4u_{i,j}^m + u_{i,j+1}^m + u_{i+1,j}^m) / 4 \quad (3.20a)$$

Please click the advert




dongenergy.com/job


Dear highly educated engineering and finance students,

if you are driven, ambitious, open-minded and focused - we have a challenge for you. Actually, the greatest challenge in the world. Curious? Visit dongenergy.com/job

Best wishes

DONG Energy





which is the of the same form as (3.18) with,

$$R_{i,j}^m = (u_{i-1,j}^{m+1} + u_{i,j-1}^{m+1} - 4u_{i,j}^m + u_{i,j+1}^m + u_{i+1,j}^m) / 4 \quad (3.20b)$$

Hence we may convert this to,

$$u_{i,j}^{m+1} = (1 - w)u_{i,j}^m + w(u_{i-1,j}^{m+1} + u_{i,j-1}^{m+1} + u_{i,j+1}^m + u_{i+1,j}^m) / 4 \quad (3.21)$$

which is called the (point) SoR method. This method is implemented in the same way as the previous iterative methods. If $w = 1$ (3.21) reduces to the Gauss-Seidel method. The question is what is the best choice of w for fastest convergence? This is a difficult question to answer in general and we must use numerical experiments to find an approximate best value. For the interested reader some convergence analysis for the three iterative methods is given in Appendix D.

3.12 Line SoR

As we shall see next, the use of a classical tridiagonal matrix method can greatly improve the efficiency of the previous SoR method. This is because we can update the solution along a whole line of grid points at once instead of simply point by point, hence the name *line* SoR. The line of grid points will normally be a whole row or column in the grid. We start from the finite difference toolkit approximation to Laplace's equation in compass notation (3.3c)

$$u_O^{m+1} = \frac{u_E^m + u_W^{m+1} + b^2(u_N^m + u_S^{m+1})}{2(1+b^2)} \quad (3.22)$$

where we have introduced the iteration index m and assumed the usual ordering working through the grid points left to right and bottom to top. (3.22) is the Gauss-Seidel method (3.16) in compass notation on a rectangular mesh with $b = \Delta x / \Delta y$. The corresponding point SoR method is,

$$u_O^{m+1} = (1 - w)u_O^m + \frac{w}{2(1+b^2)} [u_E^m + u_W^{m+1} + b^2(u_N^m + u_S^{m+1})] \quad (3.23)$$

which is the compass notation form of (3.21) on a rectangular mesh. We proceed by moving the East and West terms onto the left hand side, with the East term written at the $m+1$ level, i.e. (3.23) becomes,

$$2(1+b^2)u_O^{m+1} - wu_E^{m+1} - wu_W^{m+1} = b_O \quad (3.24)$$

Where

$$b_o = 2(1 + b^2)(1 - w)u_o^m + wb^2(u_N^m + u_s^{m+1}) \quad (3.25)$$

and noting that the data u_s^{m+1} on the right hand side is known if we assume a bottom to top ordering for the calculation. (3.24) is now in tridiagonal form. The corresponding matrix equations for the unknown data u at the interior grid points ($i = 2, M-1$) in a single row j are:

$$\begin{bmatrix} 2(1+b^2) & -w & & & \\ -w & 2(1+b^2) & -w & & 0 \\ & \cdot & \cdot & \cdot & \\ & 0 & & -w & 2(1+b^2) & -w \\ & & & -w & 2(1+b^2) \end{bmatrix} \begin{bmatrix} u_{2,j}^{m+1} \\ u_{3,j}^{m+1} \\ \cdot \\ \cdot \\ u_{M-1,j}^m \end{bmatrix} = \begin{bmatrix} b_{2,j} \\ b_{3,j} \\ \cdot \\ \cdot \\ b_{M-1,j} \end{bmatrix} \quad (3.26)$$

where we see that a tridiagonal matrix is one in which the entries in the main diagonal and in the diagonals above and below it are in general non-zero, with the remaining entries zero. A tridiagonal system like (3.26) can be solved very efficiently using the Thomas algorithm (see website) for the $m+1$ iterated values of u along one complete line or row of the mesh at once. This is the *line SoR* method, or *SoR by lines*. For convenience, the matrix equations (3.26) assume zero end-point boundary values. Dirichlet or von Neumann boundary conditions (see Appendix B) involve only marginal changes to the entries in the tridiagonal matrix and the right hand side vector b . (3.26) also assumes that the nodes in each row j are numbered left to right from $i=1$ to M where at the end points ($i=1$ and $i=M$) boundary conditions are applied. Instead of visiting each node in the mesh at each iteration, we solve a complete row at a time by solving (3.26), working up the rows bottom to top to complete one iteration cycle. The procedure is illustrated in Figure 3.3. The row sweeps are continued until the solution values converge to the required accuracy. In order to maintain diagonal dominance of the equations (3.26), and retain computational efficiency, we must ensure that $w \leq 1+b^2$ (on a square mesh $w \leq 2$).

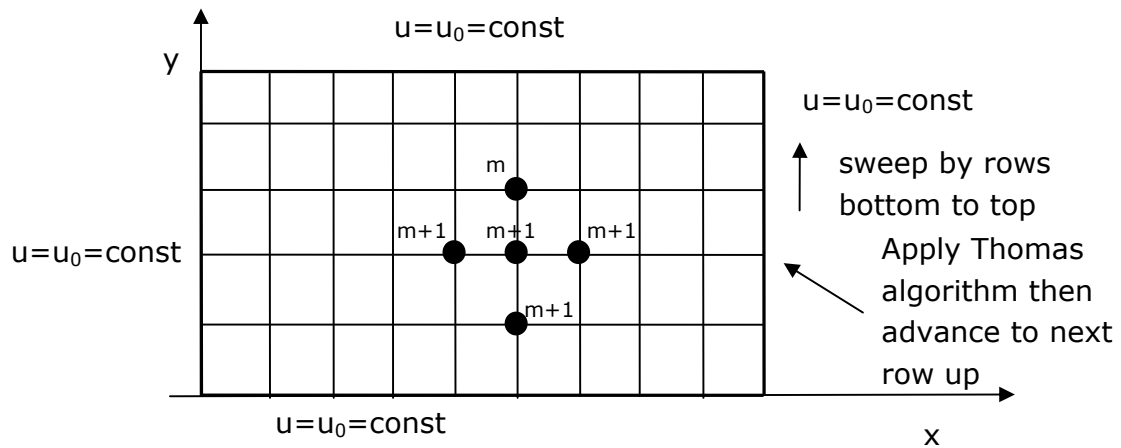


Figure 3.3 SoR by Lines

The faster convergence of line SoR compared to standard (point) SoR is due to the greater influence of the boundary values that affect all nodes at each sweep. It can be shown that the optimum value of the relaxation parameter w is given by the smaller root of,

$$t^2 w^2 - 16w + 16 = 0 \quad (3.27)$$

where $t = \cos(\pi/(M-1)) + \cos(\pi/(N-1))$ and M and N are the number of grid points in the x and y directions respectively.

3.13 Exercise 3c

1. Write a general program to implement SoR and validate it on the test problem. Experiment with tolerances and choice of relaxation parameter.
2. Repeat Q1 using line SoR.

4. Hyperbolic Equations

4.1 Introduction

Hyperbolic equations describe many time-dependent (transient) phenomena (e.g. fluid flow) and are characterised by a speed of propagation of information (often called ‘wave speed’). Consequently future solution values can only be affected by past values in a limited neighbourhood. Hyperbolic equations may also admit discontinuities in the solution variables requiring advanced numerical treatment. A proper mathematical treatment of hyperbolic equations involves the study of *characteristics* that is beyond the scope of this book. Instead we use experimental results to illustrate key concepts. Attention is focused on the 1D linear advection equation that, although not strictly hyperbolic according to the classical definition, contains the essential elements of hyperbolic PDEs and, as we will see later, is a component of the shallow water equations that are truly hyperbolic. We will look at numerical results from the simple FD scheme in Chapter 2 to solve the 1D linear advection equation and this will lead to a series of basic questions about numerical schemes in general.

Please click the advert



May we offer you one of the world's greatest challenges?
In all humility.

We are looking for highly educated engineers and finance students.

Join us at dongenergy.com/job

dongenergy.com/job




4.2 1D Linear Advection Equation

The linear advection equation (also known as the transport equation) may be used in a model of various phenomena like the movement of pollutant in a river or the movement of an air/water interface inside a sophisticated numerical method (e.g. Volume of Fluid method). We look at the simplest version of this equation. In one spatial dimension the linear advection equation is:

$$U_t + v U_x = 0 \quad (4.1a)$$

Here the independent variables are t (time) and x (space). The dependent variable U is a function of t and x .

A PDE on its own does not give a complete model of a particular problem. In addition to the PDE, we need to give the correct initial and/or boundary conditions for the dependent variable for the particular problem. Let the initial condition for U in (4.1a) be,

$$U(0, x) = f(x) \quad (4.1b)$$

i.e. the initial value of U is given over the spatial domain by a *known* function $f(x)$. For the moment we will assume that the spatial domain is infinite so that we can ignore any boundary conditions.

4.2.1 An Interpretation of the Linear Advection Equation

If we interpret (4.1a, 4.1b) as a (partial) model of the transport of a soluble pollutant by a 1D river then $U(t, x)$ is pollutant *concentration* at time t and position x along the river and v is the (constant) velocity of the river. Equation (4.1b) gives the *initial* pollutant concentration at each point along the river. For a fixed value of t the graph of U against x is called the *concentration profile*. Suitable units measure t in seconds, x in metres and U in kg/m. A dimensional analysis of (4.1a) shows that v is measured in m/s which is correct for velocity.

A *solution* to (4.1a, 4.1b) is a function $U = U(t, x)$ which satisfies (4.1a) and the *initial conditions* (4.1b) at all points (x, t) .

4.2.2 Exact Solution of the Linear Advection Equation

It can be shown that the exact solution to (4.1a, 4.1b) is,

$$U(t, x) = f(x - vt) \quad (4.1c)$$

This means that $U(t, x)$ is just the initial concentration profile, $f(x)$, translated vt metres along the x axis. For $v > 0$, the translation is to the right and for $v < 0$, the translation is to the left. In either case the pollution moves *downstream* at the speed of the river. The following diagrams illustrate this using an arbitrary made up initial concentration profile.

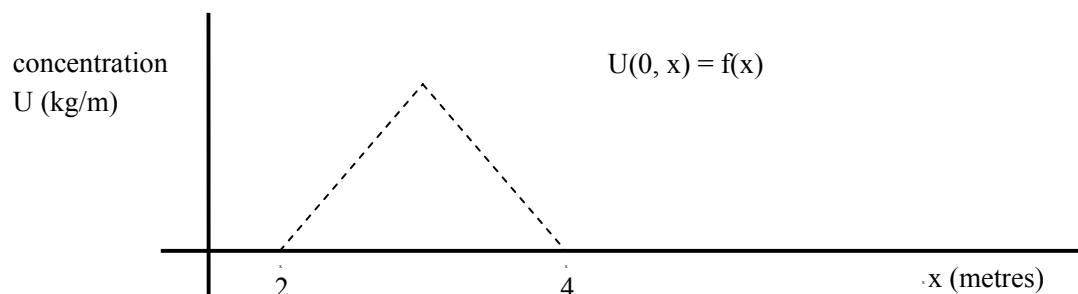


Figure 4.1a Initial concentration profile.

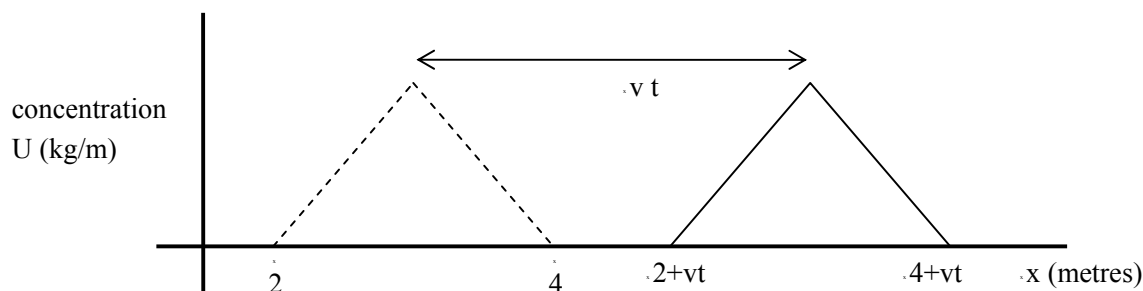


Figure 4.1b Concentration profile after time t (solid) compared to initial profile (dashed), $v > 0$.

Figure 4.1a shows a triangular initial concentration profile along the river. Concentration is only non-zero in the interval $2 < x < 4$. At time t the concentration profile is the *same shape* as the initial concentration profile but has been translated vt units *downstream*. Physically equation (4.1a) models pollutant transport in the absence of *diffusion* so the pollutant is just carried along at the speed, v , of the river. This model is unrealistic but is useful for learning purposes. It is important to note that the model is NOT a model of the flow in the river which flows with constant velocity v (we will look at models of water flow later as they are more complicated).

The linear advection equation is a good PDE to study because it has an exact solution and we can assess the performance of FD schemes by comparison to this.

4.3 Results for the Simple Linear Advection Scheme

Graphical output from the simple FD scheme of Chapter 2 is given for a range of parameters. In all cases the same initial condition profile given by Equation (2.19) was used and the computational domain, $[0, 100]$, was discretised by $N = 101$ equally spaced points. There is a brief discussion of the output in each case which will motivate the analysis later in the chapter.

4.3.1 Test Case 1

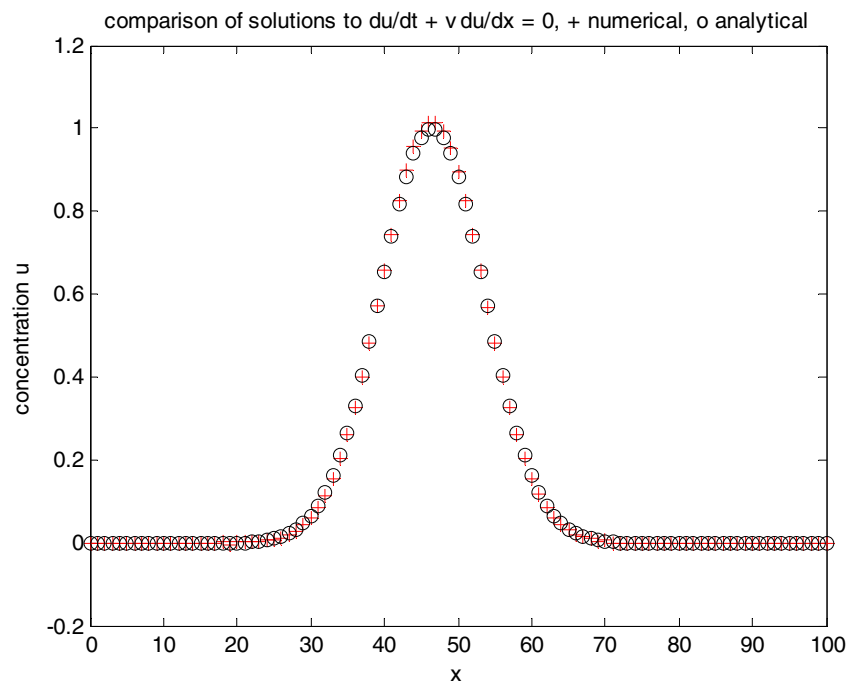


Figure 4.2a Comparison of numerical (+) and exact solutions (o) to the 1D linear advection equation using first order forward differences in both space and time using $v = 0.5$, $\Delta t = 0.3$, 10 time steps.

The simulation was run for $0.3 \times 10 = 3$ seconds. The initial condition profile has moved about 1.5 metres to the right. This is what we would expect since $v = 0.5 \text{ m/s}$ is positive and distance equals speed multiplied by time. There is ‘good’ agreement between numerical and exact solutions.

4.3.2 Test Case 2

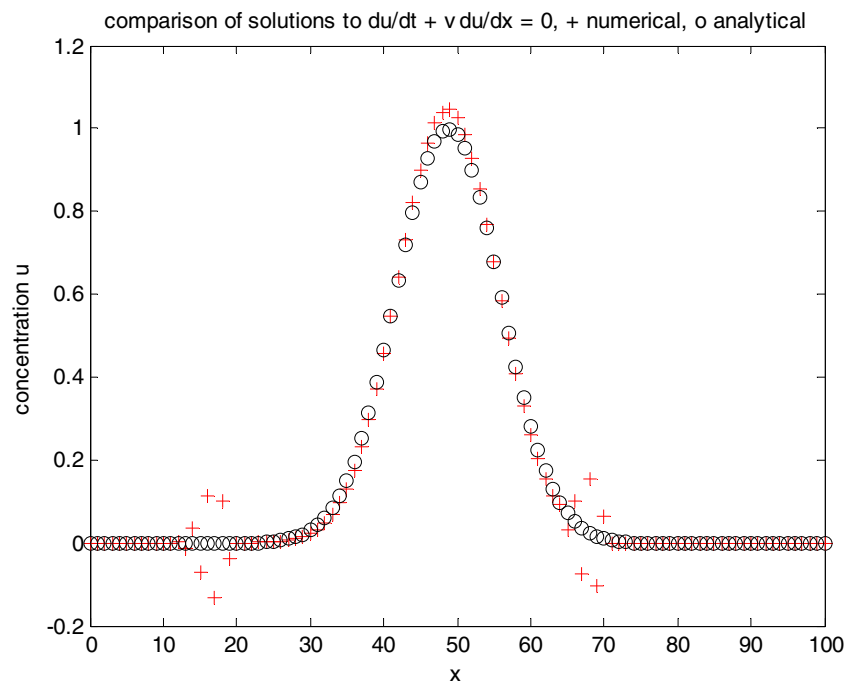
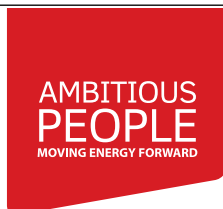


Figure 4.2b Comparison of numerical (+) and exact solutions (o) to the 1D linear advection equation using first order forward differences in both space and time using $v = 0.5$, $\Delta t = 0.3$, 25 time steps.

Please click the advert



Ambitious and curious?

We are looking for young and highly educated engineers and finance students.

Join us at dongenergy.com/job



dongenergy.com/job

DONG
energy

Download free ebooks at bookboon.com

The simulation has been run for $0.3 \times 25 = 7.5$ seconds. The numerical concentration peak has moved to the right place but is higher than the exact solution. More worryingly there is some noticeable divergence from the exact solution in the numerical solution around $x = 15$ and $x = 67$. Something is going wrong!

4.3.3 Test Case 3

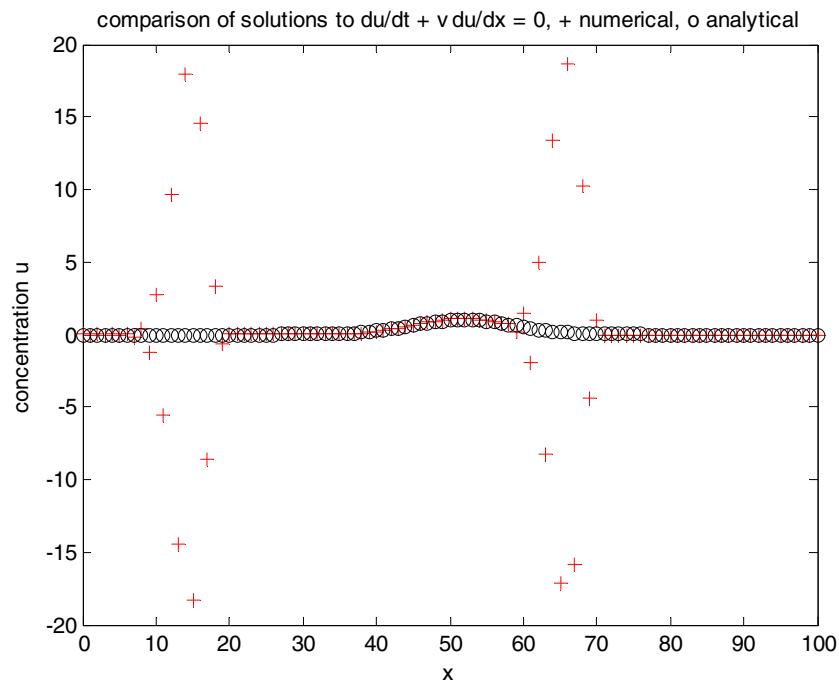


Figure 4.2c Comparison of numerical (+) and exact solutions (o) to the 1D linear advection equation using first order forward differences in both space and time using $v = 0.5$, $\Delta t = 0.3$, 44 time steps.

The simulation has been run for $0.3 \times 45 = 13.5$ seconds and numerical results have gone haywire. Note that the vertical scale has changed and the numerical results have ‘blown up’. Something is terribly wrong with this scheme!

4.3.4 Test Case 4

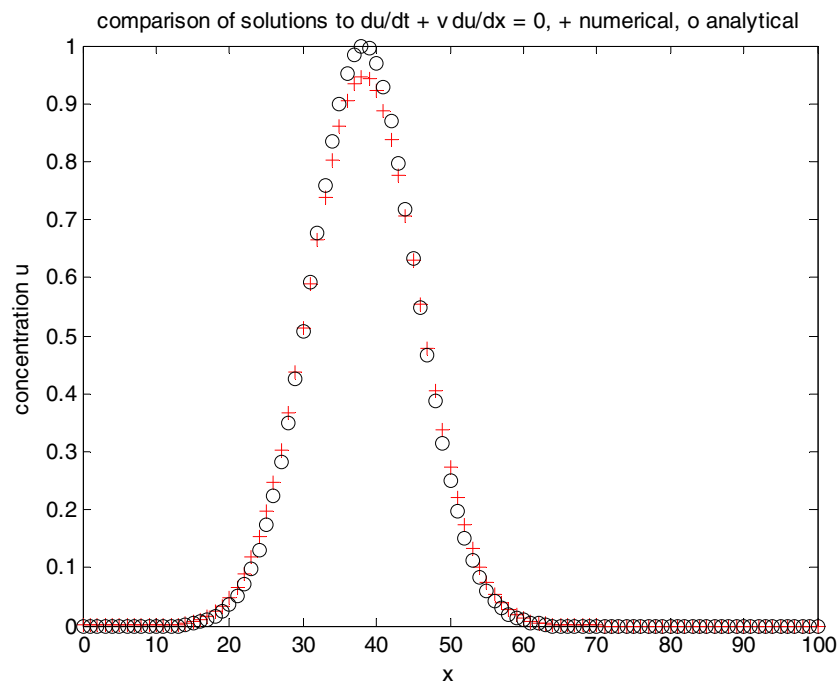


Figure 4.2d Comparison of numerical (+) and exact solutions (o) to the 1D linear advection equation using first order forward differences in both space and time using $v = -0.5$, $\Delta t = 0.3$, 44 time steps.

The simulation has again been run for 13.5 seconds but the sign of the velocity, v , has been changed. As expected the concentration profile moves to the left. Results look reasonable this time so there is a lack of symmetry with Test Case 3. Obviously the behaviour of the numerical scheme is influenced by the sign of v .

4.3.5 Test Case 5

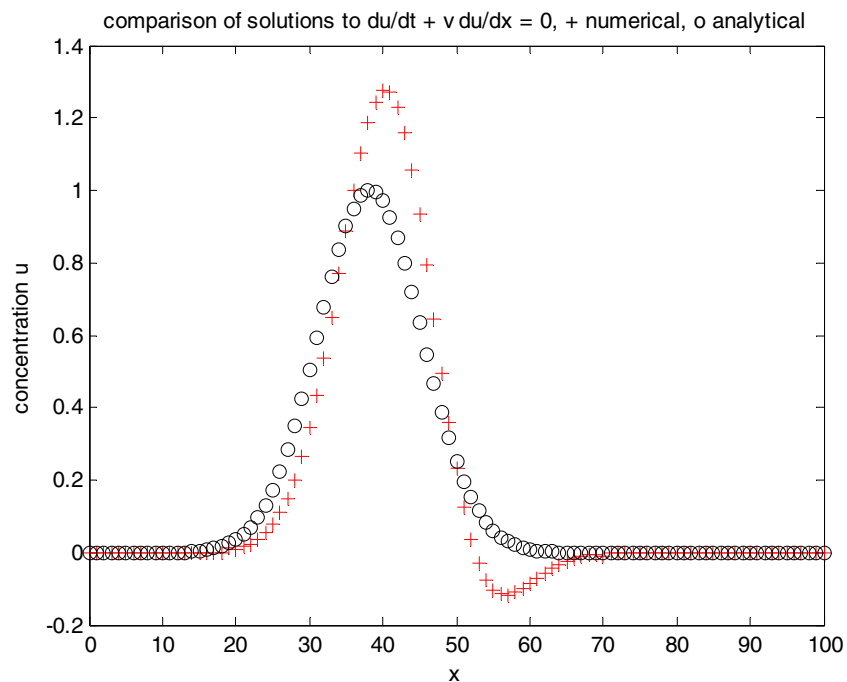


Figure 4.2e Comparison of numerical (+) and exact solutions (o) to the 1D linear advection equation using first order forward differences in both space and time using $v = -0.5$, $\Delta t = 13.5$, 1 time step.


**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

dongenergy.com/job

Everybody is talking...

...about future energy supply. We are not.
Stop talking and make a career
moving energy forward.

Ambitious engineers and
finance students go to
dongenergy.com/job



Please click the advert

The simulation has again been run for 13.5 seconds with $v = -0.5$, as before. The time step has been increase to 13.5 so that the end result is achieved in a *single iteration*. As before the concentration profile moves to the left. However results look bad and there is evidence that the numerical solution has started to ‘blow up’! The time step, Δt , could be too big.

A number of questions arise naturally from the results of these numerical experiments:

Q1) Can we design other FD schemes to get more accurate results?

Q2) How do we know the numerical results won’t ‘blow up’ at some future time?

Q3) How do we know we have an accurate solution (in the absence of an analytical solution)?

These questions are addressed by the theory in Appendix C which should be read now.

4.4 Scheme Design

We will design FD schemes to solve the 1D linear advection equation (4.1a). First we introduce some useful compact notation.

4.4.1 Operator Notation

Let, ∂_t denote $\frac{\partial}{\partial t}$, ∂_{tt} denote $\frac{\partial^2}{\partial t^2}$, ∂_{xy} denote $\frac{\partial^2}{\partial y \partial x}$ etc.

Then (4.1a) can be written,

$$\partial_t U + v \partial_x U = 0 \quad (4.2a)$$

$$\therefore \partial_t U = -v \partial_x U \quad (4.2b)$$

By definition,

$$\partial_{tt} U = \partial_t (\partial_t U) \quad (4.3a)$$

Using (4.2b), (4.3a) gives,

$$\partial_{tt} U = \partial_t (-v \partial_x U) \quad (4.3b)$$

$$= -v \partial_t (\partial_x U)$$

$$= -v \partial_{xt} U$$

$$= -v \partial_{tx} U$$

$$= -v \partial_x (\partial_t U)$$

$$= -v \partial_x (-v \partial_x U)$$

$$= v^2 \partial_{xx} U \quad (4.3c)$$

Now we can design some FD schemes to solve Equation (4.1a).

For fixed x , the Taylor expansion of $U(t + \Delta t, x)$ to order 3 gives,

$$U(t + \Delta t, x) = U(t, x) + \Delta t U_t + \frac{\Delta t^2}{2!} U_{tt} + O(\Delta t^3) \quad (4.4a)$$

which, in operator notation is,

$$U(t + \Delta t, x) = U(t, x) + \Delta t \partial_t U + \frac{\Delta t^2}{2!} \partial_{tt} U + O(\Delta t^3) \quad (4.4b)$$

Using (4.2b) and (4.3c) the partial derivatives with respect to t can be replaced by partial derivatives with respect to x to give,

$$U(t + \Delta t, x) = U(t, x) - \Delta t v \partial_x U + \frac{\Delta t^2}{2!} v^2 \partial_{xx} U + O(\Delta t^3) \quad (4.4c)$$

Let,

$$L_x(\Delta t) = 1 - \Delta t v \partial_x + \frac{\Delta t^2}{2} v^2 \partial_{xx} \quad (4.5)$$

Then (4.4c) can be written,

$$U(t+\Delta t, x) = L_x(\Delta t)U(t, x) + O(\Delta t^3) \quad (4.6)$$

$L_x(\Delta t)$ is a *differential* marching operator. To design FD schemes to solve (4.8a) we simply redefine $L_x(\Delta t)$ by replacing each continuous partial derivative by a finite difference approximation (denoted by δ_x, δ_{xx}) to give,

$$L_x(\Delta t) = 1 - \Delta t v \delta_x + \frac{\Delta t^2}{2} v^2 \delta_{xx} \quad (4.7)$$

$L_x(\Delta t)$ is now a *difference* marching operator. Different FD approximation choices for δ_x, δ_{xx} give rise to different FD time marching schemes. The general (second order in time) FD time marching scheme for the 1D linear advection equation (4.8a) can now be written as,

$$u_i^{n+1} = L_x(\Delta t)u_i^n \quad (4.8a)$$

Please click the advert





Dear highly educated engineering and finance students,

if you are driven, ambitious, open-minded and focused - we have a challenge for you. Actually, the greatest challenge in the world. Curious? Visit dongenergy.com/job

Best wishes

DONG Energy

dongenergy.com/job

Written out in full this is,

$$u_i^{n+1} = u_i^n - \Delta t v \delta_x u_i^n + \frac{\Delta t^2}{2} v^2 \delta_{xx} u_i^n \quad (4.8b)$$

(As previously defined, u_i^n is the approximation to the exact solution $U(t_n, x_i)$ at the i^{th} grid point and the n^{th} time step). Some examples of FD schemes are now given.

4.4.2 Example 1: Forward Time Centred Space (FTCS) Scheme

From our FD toolkit we choose $\delta_x u_i^n = \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x}$ and $\delta_{xx} u_i^n = 0$.

Equation (4.8b) becomes,

$$u_i^{n+1} = u_i^n - \frac{v\Delta t}{2\Delta x} (u_{i+1}^n - u_{i-1}^n) \quad (4.9)$$

This is the FTCS scheme and it has the following stencil.

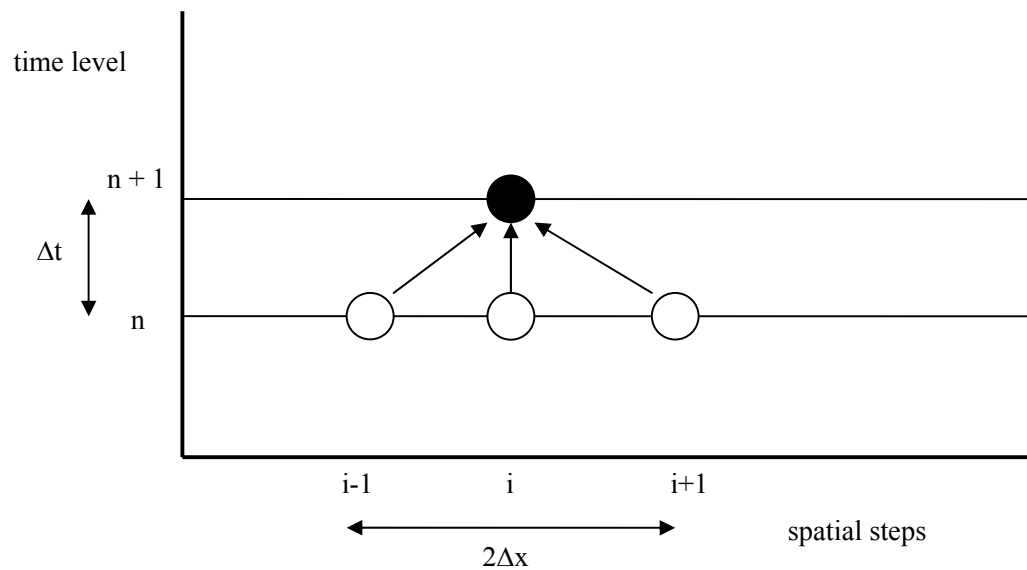


Figure 4.3 Stencil for the FTCS Scheme

Notes:

- 1) The scheme is *first* order in time and *second* order in space (see Appendix C for definition of the order of a scheme).
- 2) Ghost values are required at both left and right ends of the computational domain (see Appendix B for boundary conditions).

4.4.3 Example 2: First Order Upwind (FOU) Scheme

From our FD toolkit we choose $\delta_x u_i^n = \frac{u_i^n - u_{i-1}^n}{\Delta x}$ and $\delta_{xx} u_i^n = 0$. Equation (4.8b) becomes,

$$u_i^{n+1} = u_i^n - \frac{v\Delta t}{\Delta x} (u_i^n - u_{i-1}^n) \quad (4.10)$$

This is the FOU scheme and it has the following stencil.

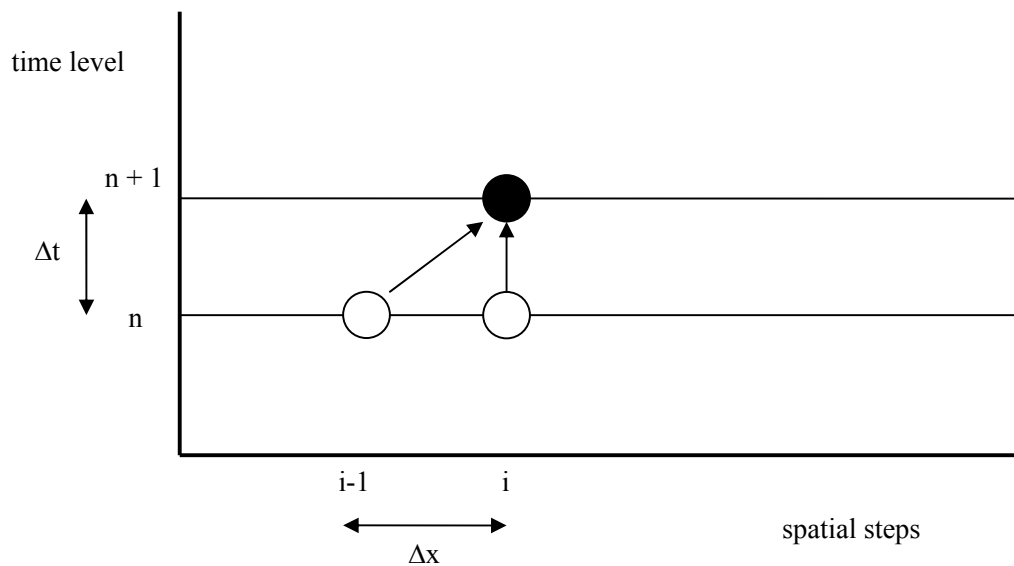


Figure 4.4 Stencil for the FOU Scheme

Notes:

- 1) The scheme is *first* order in time and first order in space.
- 2) A ghost value is required at the left end of the computational domain.

4.4.4 Example 3: Lax-Wendroff Scheme

From our FD toolkit choose $\delta_x u_i^n = \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x}$, $\delta_{xx} u_i^n = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$.

Equation (4.8b) becomes,

$$u_i^{n+1} = u_i^n - \frac{v\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) + \frac{v^2\Delta t^2}{2\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \quad (4.11)$$

Please click the advert

dongenergy.com/job



May we offer you one of the world's greatest challenges?
In all humility.

We are looking for highly educated engineers and finance students.

Join us at dongenergy.com/job





This is the Lax-Wendroff scheme and it has the following stencil.

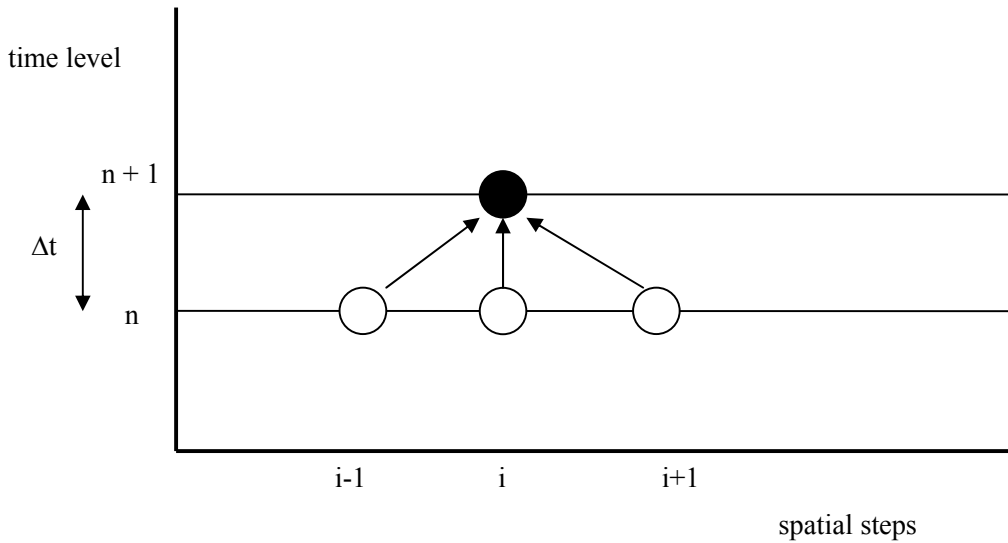


Figure 4.5 Stencil for the Lax-Wendroff Scheme

Notes:

- 1) The scheme is *second* order in time and second order in space.
- 2) Ghost values are required at both left and right ends of the computational domain.

4.4.5 Example 4: Lax-Friedrichs Scheme

This is the same as the FTCS scheme except that the first term on the right of (4.9) is replaced by the average of its 2 neighbouring values, i.e. u_i^n is replaced by $\frac{u_{i+1}^n + u_{i-1}^n}{2}$. As in the FTCS scheme we choose

$$\delta_x u_i^n = \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} \text{ and } \delta_{xx} u_i^n = 0.$$

Equation (4.8b) becomes,

$$u_i^{n+1} = \frac{u_{i+1}^n + u_{i-1}^n}{2} - \frac{v\Delta t}{2\Delta x} (u_{i+1}^n - u_{i-1}^n) \quad (4.12)$$

This is the Lax-Friedrichs scheme and it has the following stencil.

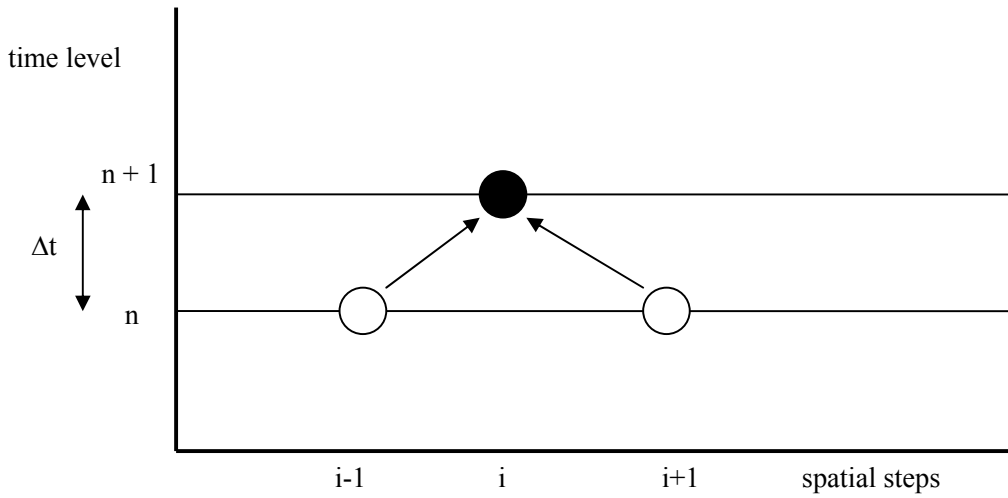


Figure 4.6 Stencil for the Lax-Friedrichs Scheme

Notes:

- 1) The scheme is *first* order in time and *first* order in space.
- 2) Ghost values are required at both left and right ends of the computational domain.
- 3) Although this scheme appears to be quite similar to the FTCS scheme its performance is very different (see Appendix C).

4.5 Multi-Level Scheme Design

So far all our schemes have been based on using data at the current time level (n) to advance to the next time level ($n+1$). This approach can be extended to multi-level schemes by performing Taylor approximations at $t - \Delta t$ and using algebraic manipulation as we shall now see. Replacing Δt by $-\Delta t$ in (4.4a) gives,

$$U(t - \Delta t, x) = U(t, x) - \Delta t U_t + \frac{\Delta t^2}{2!} U_{tt} + O(\Delta t^3) \quad (4.13a)$$

Subtracting (4.13b) from (4.13a) and gives,

$$U(t + \Delta t, x) - U(t - \Delta t, x) = 2\Delta t U_t + O(\Delta t^3) \quad (4.13b)$$

In operator notation this is,

$$U(t + \Delta t, x) - U(t - \Delta t, x) = 2\Delta t \partial_t U + O(\Delta t^3) \quad (4.13c)$$

Using (4.2b) this becomes,

$$U(t + \Delta t, x) - U(t - \Delta t, x) = -2v\Delta t \partial_x U + O(\Delta t^3) \quad (4.13d)$$

Dropping the error term, replacing the differential operator by the difference operator and using the usual discrete notation gives the general FD scheme,

$$u_i^{n+1} = u_i^{n-1} - 2v\Delta t \delta_x u \quad (4.14)$$

Please click the advert

dongenergy.com/job

**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

Ambitious and curious?

We are looking for young and highly educated engineers and finance students.

Join us at dongenergy.com/job

DONG
energy

4.5.1 Example 5: Leap-Frog Scheme

In (4.14) we choose $\delta_x u_i^n = \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x}$ to give,

$$u_i^{n+1} = u_i^{n-1} - \frac{v\Delta t}{\Delta x} (u_{i+1}^n - u_{i-1}^n) \quad (4.15)$$

This is the Leap-Frog scheme and it has the following stencil.

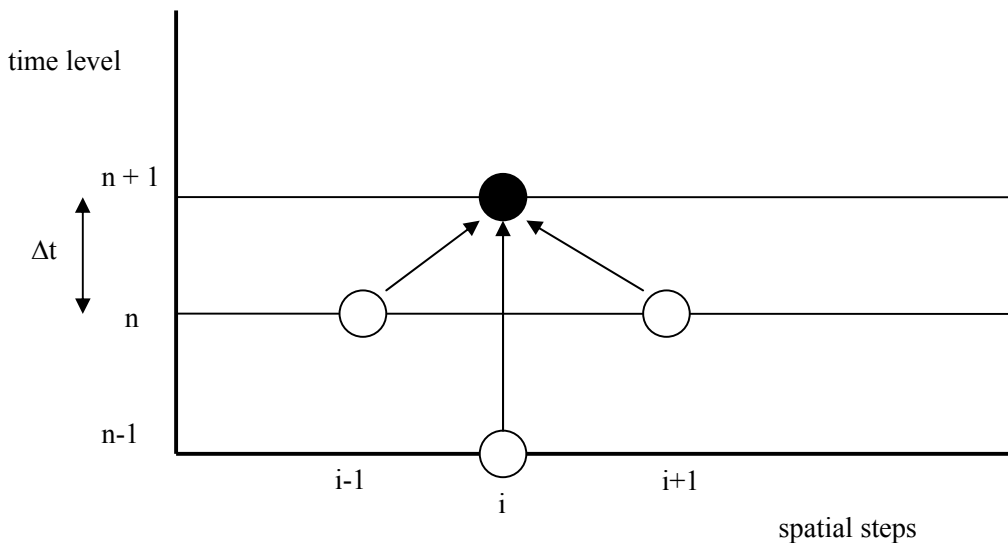


Figure 4.7 Stencil for the Leap-Frog Scheme

Notes:

- 1) The scheme is *second* order in time and *second* order in space.
- 2) Ghost values are required at both left and right ends of the computational domain.
- 3) Initial conditions are required at *two* time levels!

4.6 Exercise 4a

- 1a. Download the simple linear advection solver ‘linearadvection’ (from Chapter 2).
- 1b. Run ‘linearadvection’ on Test Cases 1-5 and check that the output agrees with the figures in the notes.

- 1c. Run ‘linearadvection’ with different run times, time steps, spatial steps and values of v (both positive and negative). In each case write down in your own words how the numerical scheme behaves in comparison to the exact solution.
- 1d. Are there any values of Δt , Δx , v , $n_{\text{timesteps}}$ that give good results?
- 2a. Copy ‘linearadvection’ to ‘linearadvectionFOU’. Change the solver in this new file so that it implements the FOU scheme. Verify your code by comparison to a pen and paper calculation as per Table
- 2.2. Important: the FOU scheme requires a left hand ghost point and hence in the first non-ghost point has index 2 and the last non-ghost point has index $N+1$.
- 2b. Run ‘linearadvectionFOU’ using the parameters from Test Cases
- 1-4. In each case write down in your own words how the numerical scheme behaves with reference to the analytical results.
- 2c. Run ‘linearadvectionFOU’ with different run times, time steps, spatial steps and values of v (both positive and negative). In each case write down in your own words how the numerical scheme behaves with reference to the analytical results.
- 2d. Are there any values of Δt , Δx , v , $n_{\text{timesteps}}$ that give good results?
- 2e. Show *experimentally* that the FOU scheme is first order in space.
3. Copy ‘linearadvection’ to ‘linearadvectionLW’. Change the solver in this new file so that it implements the Lax-Wendroff scheme and verify the code. Repeat 2b-e for ‘linearadvectionLW’.
4. Do all exercises in Appendix C and relate theoretical stability results to your experimentally derived results.

4.7 Implicit Schemes

The previous schemes are called *explicit* schemes because data at the next time level is obtained from an explicit formula involving data from previous time levels. This leads to a (stability) restriction on the maximum allowable time step, Δt (see Appendix C). Now we come to a different type of scheme – implicit, in which data from the next time level occurs on both sides of the difference scheme that necessitates solving a system of linear equations. There is no stability restriction on the maximum time step which may be much larger than an explicit scheme for the same problem. In implicit schemes the time step is chosen on the basis of accuracy considerations.

4.7.1 Example 6: Crank-Nicolson Scheme

This is an implicit scheme. In previous examples spatial derivatives are approximated at time level n . However values change between time level n and time level $n+1$ so a better approximation to spatial derivatives could make use of data at *both* time levels. Let,

$$\delta_x u_i^n = \alpha \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} + (1-\alpha) \frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2\Delta x}, \quad 0 \leq \alpha \leq 1. \quad (4.16)$$

This is a weighted average of central difference approximations at times levels n and $n+1$. Choose $\delta_{xx} u_i^n = 0$. Then for $\alpha = 1/2$ (4.8b) becomes,

$$u_i^{n+1} = u_i^n - \frac{v\Delta t}{2\Delta x} \left(\frac{u_{i+1}^n - u_{i-1}^n}{2} + \frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2} \right) \quad (4.17)$$


dongenergy.com/job

**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

Everybody is talking...

...about future energy supply. We are not.
Stop talking and make a career
moving energy forward.

Ambitious engineers and
finance students go to
dongenergy.com/job



Please click the advert

This is the Crank-Nicolson scheme and it has the following stencil.

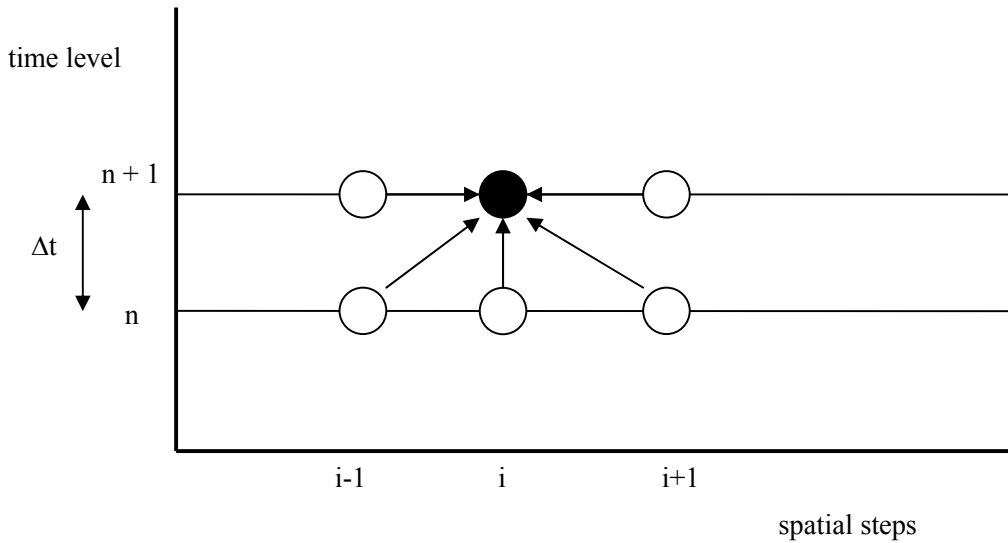


Figure 4.8 Stencil for the Crank-Nicolson Scheme

Notes:

- 1) The scheme is first order in time and second order in space.
- 2) Ghost values are required at both left and right ends of the computational domain.
- 3) The scheme is *implicit* so values at time level $n+1$ are found by solving a system of linear equations.

4.7.2 Implementation of the Crank-Nicolson Scheme

Letting $c = \frac{v \Delta t}{\Delta x}$, (4.17) is,

$$u_i^{n+1} = u_i^n - \frac{c}{2} \left(\frac{u_{i+1}^n - u_{i-1}^n}{2} + \frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2} \right) \quad (4.18)$$

Re-writing (4.18) gives,

$$4u_i^{n+1} = 4u_i^n - c(u_{i+1}^n - u_{i-1}^n + u_{i+1}^{n+1} - u_{i-1}^{n+1}) \quad (4.19a)$$

Rearranging so that data from the same time level is on the same side gives,

$$-cu_{i-1}^{n+1} + 4u_i^{n+1} + cu_{i+1}^{n+1} = cu_{i-1}^n + 4u_i^n - cu_{i+1}^n \quad (4.19b)$$

The data at time level n is assumed known so (4.19b) is simplified by replacing the right hand side by d_i^n to give,

$$cu_{i-1}^{n+1} + 4u_i^{n+1} - cu_{i+1}^{n+1} = d_i^n \quad (4.19c)$$

For each grid point $i = 1, 2, \dots, N$, we write out (4.19c) to give,

$$\begin{aligned} -cu_0^{n+1} + 4u_1^{n+1} + cu_2^{n+1} &= d_1^n \\ -cu_1^{n+1} + 4u_2^{n+1} + cu_3^{n+1} &= d_2^n \\ -cu_2^{n+1} + 4u_3^{n+1} + cu_4^{n+1} &= d_3^n \\ &\vdots \\ -cu_{N-2}^{n+1} + 4u_{N-1}^{n+1} + cu_N^{n+1} &= d_{N-1}^n \\ -cu_{N-1}^{n+1} + 4u_N^{n+1} + cu_{N+1}^{n+1} &= d_N^n \end{aligned} \quad (4.19d)$$

(4.19d) is a system of N linear equations in what looks like $N+2$ unknowns! However u_0^{n+1} and u_{N+1}^{n+1} on the left hand side of (4.19d) are ghost values which may be known directly or can be calculated in terms of neighbouring values depending on the type of boundary condition given in the problem (see Appendix B). In the former case we can move these known values to the right hand side of (4.19d) and, letting $d_1^{n'} = d_1^n + cu_0^{n+1}$, $d_N^{n'} = d_N^n - cu_{N+1}^{n+1}$, (4.19d) becomes,

$$\begin{aligned} 4u_1^{n+1} + cu_2^{n+1} &= d_1^{n'} \\ -cu_1^{n+1} + 4u_2^{n+1} + cu_3^{n+1} &= d_2^n \\ -cu_2^{n+1} + 4u_3^{n+1} + cu_4^{n+1} &= d_3^n \\ &\vdots \\ -cu_{N-2}^{n+1} + 4u_{N-1}^{n+1} + cu_N^{n+1} &= d_{N-1}^n \\ -cu_{N-1}^{n+1} + 4u_N^{n+1} &= d_N^{n'} \end{aligned} \quad (4.19e)$$

This system is expressed as the matrix equation,

$$A \underline{u}^{n+1} = \underline{d}^n \quad (4.19f)$$

where,

$$A = \begin{pmatrix} 4 & c & 0 & \dots & 0 \\ -c & 4 & c & 0 & \dots & 0 \\ 0 & -c & 4 & c & 0 & \dots & 0 \\ 0 & 0 & -c & 4 & c & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & & 0 & -c & 4 & c \\ 0 & \dots & & & 0 & -c & 4 \end{pmatrix}, \underline{u}^{n+1} = \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_{N-1}^{n+1} \\ u_N^{n+1} \end{pmatrix}, \underline{d}^n = \begin{pmatrix} d_1^{n'} \\ d_2^n \\ \vdots \\ d_{N-1}^n \\ d_N^{n'} \end{pmatrix}.$$

The solution to (4.19f) is obviously,

$$\underline{u}^{n+1} = A^{-1} \underline{d}^n \quad (4.19g)$$

Please click the advert

**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

*Dear highly educated engineering
and finance students,*

if you are driven, ambitious, open-minded
and focused - we have a challenge for you.
Actually, the greatest challenge in the
world. Curious? Visit dongenergy.com/job

Best wishes

DONG Energy

dongenergy.com/job



DONG
energy

Notes:

1. (4.19g) is solved at each time step and the solution updated iteratively.
2. In practical problems A may be very large (e.g. 1000×1000) so an efficient matrix inversion method may be needed (see Chapter 3).
3. In this example boundary values are known so A is constant and needs only to be inverted once. For problems where the left hand side boundary values are calculated in terms of neighbouring values (e.g. Derivative boundary conditions, see Appendix B) the first and last rows of A will vary from time step to time step.
4. In our example A has a special structure – it is tridiagonal. This special structure permits the use of the efficient Thomas algorithm for matrix inversion algorithm (see downloadable code for Chapter 3).

Results for the usual test case are given in Figure 4.9.

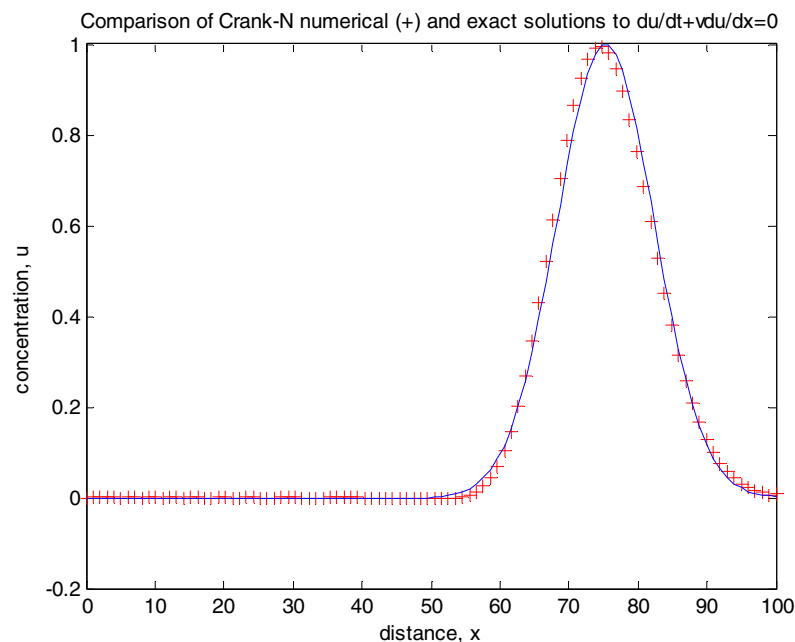


Figure 4.9 Comparison of numerical (+) and exact solutions (o) to the 1D linear advection equation using the Crank-Nicolson scheme with $v = 0.5$, $C = 2.0$, 15 time steps.

4.8 Exercise 4b

1. Use von Neumann stability analysis (Appendix C) to show that the Crank-Nicolson scheme is unconditionally stable.
2. Using the Crank-Nicolson scheme for the 1D linear advection equation with $p = 0$, $q = 100$, $v = 0.5$, $N = 5$, $C = 1$ and Dirichlet boundary conditions $u_0^n = 0 = u_{N+1}^n \forall n$, and the usual initial profile. Write down the system of linear equations for the first time step of the Crank-Nicolson scheme and express them as a matrix equation. Invert the (5×5) matrix (by pen and paper or use a package) and hence find the concentration values at each grid point after the first time step.
3. Using your calculation from Q2 to verify your program and write a program to implement the Crank-Nicolson scheme using $N = 50$ grid points. Experiment with different time steps and compare your numerical results to the exact solution. Write a short report detailing the characteristics of the scheme (tip: use the code from a previously written scheme as a basis for your new code).

Please click the advert



May we offer you one
of the world's greatest
challenges?
In all humility.

We are looking for highly
educated engineers and
finance students.

Join us at dongenergy.com/job

dongenergy.com/job





5. Parabolic Equations: the Advection-Diffusion Equation

5.1 Introduction

The advection-diffusion equation belongs to the class of parabolic PDEs. In 1 spatial dimension it is,

$$U_t + vU_x = K_x U_{xx} \quad (5.1)$$

K_x is called the diffusion coefficient (in the x direction). If $K_x = 0$ then (5.1) is the linear advection equation which we studied in Chapter 4. Using our previous interpretation of the linear advection equation in which $U = U(t, x)$ is river pollutant concentration and v is the speed of the flow, (5.1) is a more realistic description of pollutant transport. Not only does the initial pollutant move downstream with velocity v , the pollutant also diffuses into the surrounding water at rate K_x (the presence of second order spatial derivatives often indicates a diffusive process).

Figure 5.1 illustrates a pure advective process ($K = 0$ in (5.1a)) compared to an advection-diffusion process ($K > 0$ in (5.1b)). As we have seen previously for pure advection there is no change in the initial concentration profile as it moves downstream at speed v . When diffusion is introduced the initial concentration profile moves downstream at speed v but also spreads out (diffuses) and reduces in height over time. It is important to realise that Figure 5.1b shows the *exact solution* – the spreading and reduction in height is a consequence of the underlying physical process. Of course we have seen similar behaviour with numerical solutions hence we talk of ‘numerical diffusion’ that is a feature of most numerical schemes.

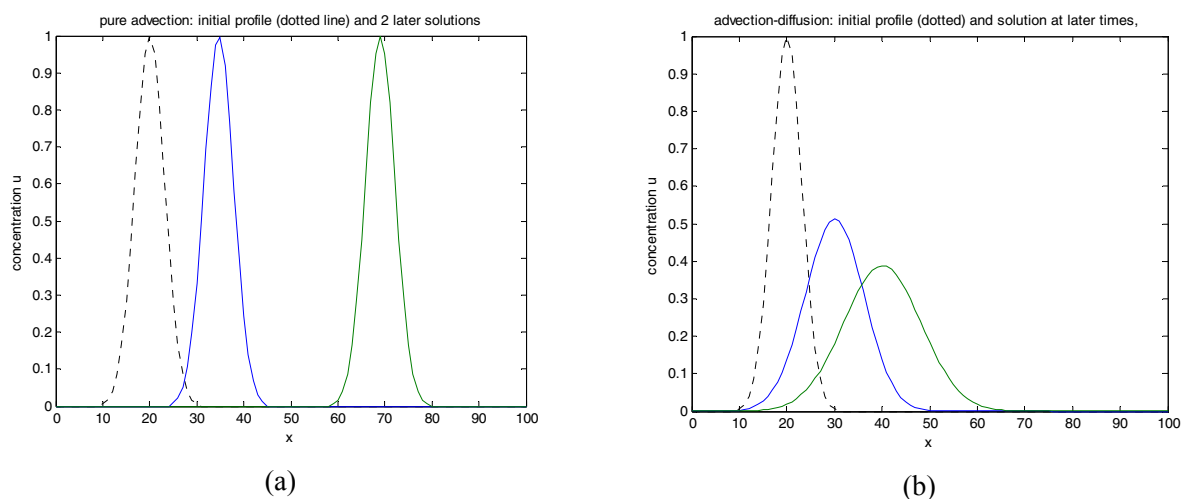


Figure 5.1 Time evolution of solutions for advection (a) and advection-diffusion (b)

5.2 Pure Diffusion

To keep things simple we will start with the pure diffusion equation,

$$U_t = K_x U_{xx} \quad (5.2a)$$

5.2.1 Analytical Solution of the Pure Diffusion Equation

Equation (5.2a) has an analytical solution for special conditions that is useful for checking numerical schemes. It can be shown that over the x interval $[0, 1]$ with initial condition, $U(0, x) = \sin(\pi x)$ and boundary conditions, $U(t, 0) = U(t, 1) = 0$ for all t , the solution of (5.2a) is,

$$U(t, x) = e^{-K_x \pi^2 t} \sin(\pi x) \quad (5.2b)$$

5.2.2 Finite Difference Scheme for the Pure Diffusion Equation

We start simply and use our FD toolbox to replace the time derivative by a first order forward difference and the spatial derivative by a symmetric difference so that (5.2a) becomes,

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = K_x \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \quad (5.3a)$$

This can be rearranged to give the explicit FD scheme,

$$u_i^{n+1} = u_i^n + \frac{\Delta t K_x}{\Delta x^2} (u_{i+1}^n - 2u_i^n + u_{i-1}^n) \quad (5.3b)$$

Note that the presence of u_{i+1}^n and u_{i-1}^n indicates the need for left and right ghost values. Equation (5.3b) is written compactly as,

$$u_i^{n+1} = (1 - 2r)u_i^n + ru_{i+1}^n + ru_{i-1}^n \quad (5.3c)$$

where,

$$r = \frac{\Delta t K_x}{\Delta x^2} \quad (5.4)$$

Since this scheme is explicit the time step, Δt , will be limited by stability constraints. A von Neumann stability analysis (Appendix C) gives, after some manipulation,

$$G = (1 - 2r) + 2r \cos(k \Delta x) \quad (5.5a)$$

where G is the amplification factor. For stability, $|G| \leq 1$ which gives,

$$\Delta t \leq \frac{\Delta x^2}{2K_x} \quad (5.5b)$$

The problem with the analytical solution (5.2b) was run for 300 time steps with 51 grid points using scheme (5.3b) with $K_x = 1$. The maximum time step given by (5.5b) was multiplied by a safety factor of $F=0.9$. Figure 5.2 shows the results. It can be seen that the initial profile reduces in height but does not change its location. This is what we expect from a pure diffusion problem. The numerical and exact solutions are close.

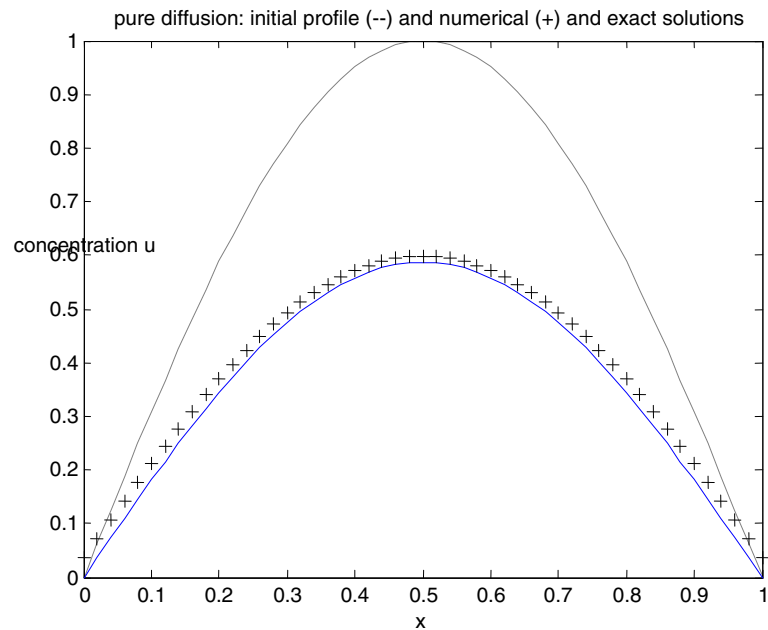


Figure 5.2 Comparison of exact and numerical solutions for a special pure diffusion problem.

5.2.3 Exercise 5a

1. State the units of K_x given standard units for the other variables in (5.1).
2. Given the initial concentration in Figure 5.1 draw plausible solutions to (5.1) at two later times on the same graph when $v = 0$.
3. Show that (5.2b) satisfies the special initial and boundary conditions.
4. Show by partial differentiation that (5.2b) is a solution of (5.2a).
5. Obtain (5.3b) from (5.3a).
6. Obtain (5.3c) from (5.3b).
7. By conducting a von Neumann stability analysis obtain (5.5a) then (5.5b).
8. Write a program to solve (5.2a) using the scheme (5.3c). Check your program on the special diffusion case and reproduce Figure 5.2.

Please click the advert



Ambitious and curious?

We are looking for young and highly educated engineers and finance students.

Join us at dongenergy.com/job

dongenergy.com/job




5.3 Advection-Diffusion Equation

We discretise the advection-diffusion equation (5.1) using a first order forward difference for the time derivative, a first order backward difference for the first space derivative (assuming $v > 0$) and a (second order) symmetric difference for the second space derivative to give,

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + v \frac{u_i^n - u_{i-1}^n}{\Delta x} = K_x \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \quad (5.6a)$$

which can be rewritten as,

$$u_i^{n+1} = u_i^n - \frac{v\Delta t}{\Delta x} (u_i^n - u_{i-1}^n) + \frac{K_x \Delta t}{\Delta x^2} (u_{i+1}^n - 2u_i^n + u_{i-1}^n) \quad (5.6b)$$

(5.6b) is an *explicit* FD scheme for solving the advection-diffusion equation. It is first order in time and space. As usual we need to find the allowable time step by a stability analysis. For convenience (5.6b) is written,

$$u_i^{n+1} = Su_{i+1}^n + (1-R-2S)u_i^n + (R+S)u_{i-1}^n \quad (5.6c)$$

where,

$$R = \frac{v\Delta t}{\Delta x}. \quad (5.7a)$$

$$S = \frac{K_x \Delta t}{\Delta x^2}. \quad (5.7b)$$

By von Neumann stability analysis it can be shown that,

$$1 - R - 2S \geq 0 \quad (5.8)$$

from which it follows that,

$$\Delta t \leq \frac{\Delta x^2}{v\Delta x + 2K_x} \quad (5.9)$$

Note that when $v = 0$, (5.9) reduces to the previously calculated stability limit for pure diffusion (5.5b) and when $K_x = 0$ (5.9) gives the well known stability limit for pure advection. Results from scheme (5.6b) are given in Figure 5.3 for $v = 0.5$, $K_x = 0.1$, $N = 101$ at time $t = 57s$ where Δt is its maximum value in (5.9) multiplied by a safety factor of 0.9. The simulation required 45 time steps. By inspection the peak of the concentration profile has moved about 28m. This correct as $v = 0.5$ and $0.5 \times 57 = 28.5m$. The peak value of the profile has decreased and its width has increased as would be expected from a diffusive process.

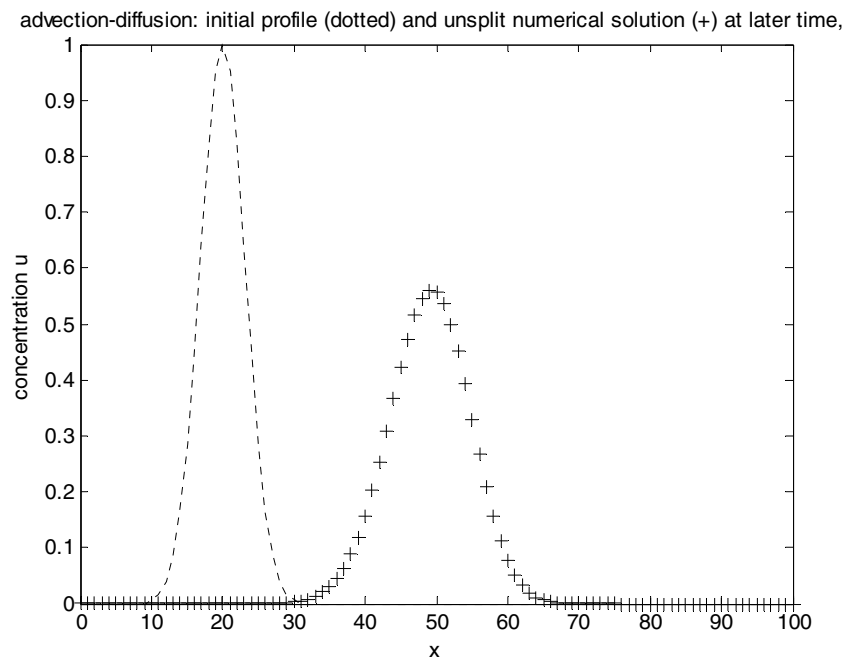


Figure 5.3 Initial profile and numerical solution (+) to the 1D advection-diffusion equation at $t = 57s$.

Important:

When running a simulation always check your results as much as you can. There are various ways to do this:

- Check 1. If there is an exact solution to a special problem run the simulation on it and compare numerical and exact results.
- Check 2. Do a pen and paper calculation on a small problem and compare results with your numerical output.
- Check 3. Subject your results to a critical scrutiny – are they what you expect to get given the physical process you are modelling?

Of course the whole purpose of a simulation is to model some aspect of reality so the *only real check* (of the model) is to compare it against real data.

5.4 Exercise 5b

1. Derive (5.6a)
2. Obtain (5.6b) from (5.6a).
3. Obtain (5.6c) from (5.6b).
4. Starting with (5.6c) carry out a von Neumann stability analysis on the solution and obtain (5.8) and hence (5.9).
5. Calculate Δx and hence Δt for the simulation whose results are shown in Figure 5.3.
6. Draw an approximate sketch of Figure 5.3 if the simulation had been run for 70 seconds.
7. For $v > 0$ show that, a) $\lim_{\Delta x \rightarrow 0} \frac{\Delta t_A}{\Delta t_{AD}} = \infty$, b) $\lim_{\Delta x \rightarrow 0} \frac{\Delta t_D}{\Delta t_{AD}} = 1$. What do you conclude from this?
8. Write a program to implement scheme (5.6b) and reproduce Figure 5.2. (hint: extend code from a previous scheme).

Please click the advert

AMBITIOUS
PEOPLE
MOVING ENERGY FORWARD

dongenergy.com/job

Everybody
is talking...

...about future energy supply. We are not.
Stop talking and make a career
moving energy forward.

Ambitious engineers and
finance students go to
dongenergy.com/job

DONG
energy

6. Extension to Multi-dimensions and Operator Splitting

6.1 Introduction

So far we have looked at PDEs in one (spatial) dimension. The real world is 3D so we must extend schemes to multi-dimensions (it should be noted however that there are many useful 1D and 2D models of 3D processes). For simplicity and to illustrate key concepts we will consider the 2D linear advection equation,

$$U_t + v_x U_x + v_y U_y = 0 \quad (6.1a)$$

Like the previous 1D linear advection equation this is another so-called ‘model’ equation in that it is a simplified version of reality. As for the 1D linear advection equation we can interpret (6.1a) as a (partial) model of pollutant transport in a 2D river. $U = U(t, x, y)$ is the pollutant concentration at time t at position (x, y) in the river (plan view) and v_x and v_y are the (constant) components of river velocity in the x and y directions respectively. Given initial conditions,

$$U(0, x, y) = g(x, y),$$

it can be shown that (6.1a) has the exact solution,

$$U(t, x, y) = g(x - v_x t, y - v_y t) \quad (6.1b)$$

i.e. the concentration profile is simply translated (advected) along at the speed of the river without changing shape (there is no diffusion term in (6.1a) – it models pure advection).

6.2 2D Scheme Design (unsplit)

Our first approach to 2D scheme design is to directly discretise the PDE. These schemes are said to be *unsplit* as opposed to *split* that we will look at later. We will design FD schemes to solve (6.1a) in an exactly similar way to Chapter 4. In operator notation for partial derivatives (6.1a) can be written,

$$\partial_t U + v_x \partial_x U + v_y \partial_y U = 0 \quad (6.2a)$$

$$\therefore \partial_t U = -v_x \partial_x U - v_y \partial_y U \quad (6.2b)$$

By definition,

$$\partial_{tt} U = \partial_t (\partial_t U) \quad (6.2c)$$

Using (6.2b, 6.2c) gives,

$$\partial_{tt} U = \partial_t (-v_x \partial_x U - v_y \partial_y U) \quad (6.2d)$$

$$= -v_x \partial_t (\partial_x U) - v_y \partial_t (\partial_y U)$$

$$= -v_x \partial_x (\partial_t U) - v_y \partial_y (\partial_t U)$$

$$= -v_x \partial_x (-v_x \partial_x U - v_y \partial_y U) - v_y \partial_y (-v_x \partial_x U - v_y \partial_y U)$$

$$= v_x^2 \partial_{xx} U + v_x v_y \partial_{yx} U + v_y v_x \partial_{xy} U + v_y^2 \partial_{yy} U$$

$$= v_x^2 \partial_{xx} U + 2v_x v_y \partial_{xy} U + v_y^2 \partial_{yy} U \quad (6.2e)$$

Now we can design some FD schemes to solve the 2D linear advection equation. Holding x and y constant, the Taylor series for $U(t+\Delta t, x, y)$ expanded about $U(t, x, y)$ to $O(\Delta t^3)$ is,

$$U(t + \Delta t, x, y) = U(t, x, y) + \Delta t \partial_t U + \frac{\Delta t^2}{2} \partial_{tt} U + O(\Delta t^3) \quad (6.3a)$$

Using (6.2b, 6.2e) to replace the time derivatives in (6.3a) and rearranging gives,

$$U(t + \Delta t, x, y) = U(t, x, y) - \Delta t (v_x \partial_x + v_y \partial_y) U + \frac{\Delta t^2}{2} (v_x^2 \partial_{xx} + 2v_x v_y \partial_{xy} + v_y^2 \partial_{yy}) U + O(\Delta t^3) \quad (6.3b)$$

Let,

$$L_{XY}(\Delta t) = 1 - \Delta t (v_x \partial_x + v_y \partial_y) + \frac{\Delta t^2}{2} (v_x^2 \partial_{xx} + 2v_x v_y \partial_{xy} + v_y^2 \partial_{yy}) \quad (6.4)$$

Then (6.3b) can be written,

$$U(t + \Delta t, x, y) = L_{XY}(\Delta t) U(t, x, y) + O(\Delta t^3) \quad (6.5)$$

$L_{XY}(\Delta t)$ is a *differential* marching operator and is second order in time. To design FD schemes to solve (6.1a) we simply redefine $L_{XY}(\Delta t)$ by replacing each continuous partial derivative by a FD approximation (denoted by δ_x, δ_{xy} etc.) to give,

$$L_{XY}(\Delta t) = 1 - \Delta t(v_x \delta_x + v_y \delta_y) + \frac{\Delta t^2}{2}(v_x^2 \delta_{xx} + 2v_x v_y \delta_{xy} + v_y^2 \delta_{yy}) \quad (6.6)$$

This is now a 2D *difference* marching operator. Different FD choices for δ_x , δ_{xy} etc. give rise to different FD time marching schemes.

The computational domain is 2D so we need a grid of points in x and y directions. We assume a rectangular computational domain with constant grid spacing of Δx and Δy in the x and y directions respectively. Δt is the time step. Extending our usual notation by using subscript j for the index in the y direction let,

$$u_{i,j}^n \equiv U(t_n, x_i, y_j) \quad (6.7)$$

The general 2D FD time marching scheme for the 2D linear advection equation (6.1a) can now be written as,

$$u_{i,j}^{n+1} = L_{XY}(\Delta t) u_{i,j}^n \quad (6.8a)$$

Please click the advert

dongenergy.com/job



Dear highly educated engineering and finance students,

if you are driven, ambitious, open-minded and focused - we have a challenge for you. Actually, the greatest challenge in the world. Curious? Visit dongenergy.com/job

Best wishes

DONG Energy





Written out in full this is,

$$u_{i,j}^{n+1} = u_{i,j}^n - \Delta t (v_x \delta_x + v_y \delta_y) u_{i,j}^n + \frac{\Delta t^2}{2} (v_x^2 \delta_{xx} + 2v_x v_y \delta_{xy} + v_y^2 \delta_{yy}) u_{i,j}^n \quad (6.8b)$$

Some examples of FD schemes are now given.

6.2.1 Example 1: First Order Upwind (FOU2D) Scheme

From our FD toolkit we choose, $\delta_x u_{i,j}^n = \frac{u_{i,j}^n - u_{i-1,j}^n}{\Delta x}$, $\delta_y u_{i,j}^n = \frac{u_{i,j}^n - u_{i,j-1}^n}{\Delta y}$,

$\delta_{xx} u_{i,j}^n = \delta_{yy} u_{i,j}^n = \delta_{xy} u_{i,j}^n = 0$. Equation (6.8b) becomes,

$$u_{i,j}^{n+1} = u_{i,j}^n - C_x (u_{i,j}^n - u_{i-1,j}^n) - C_y (u_{i,j}^n - u_{i,j-1}^n) \quad (6.9)$$

where,

$C_x = \frac{v_x \Delta t}{\Delta x}$, $C_y = \frac{v_y \Delta t}{\Delta y}$ are the Courant numbers in the x and y directions respectively. This scheme is simply the FOU scheme applied to each dimension.

Notes:

1. The scheme is first order in time and first order in each spatial dimension.
2. Ghost values are required at the left side and the lower side of the computational domain. This is shown in Figure 6.1 for a grid with 5 points in the x direction and 3 points in the y direction.
3. The 2D FD scheme (6.9) could simply have been obtained by direct replacement of the partial derivatives in the PDE (6.1a) using approximations from the FD toolkit. Our scheme is an example of an *unsplit* scheme because it is obtained directly from the 2D PDE.
4. The scheme uses backward differences for the spatial derivatives and is therefore only an upwind scheme for positive velocities. When one or more velocities are negative the scheme will fail unless forward differences are used appropriately.

5. The scheme is explicit and will be subject to a stability constraint on the allowable time step. In the 1D case the heuristic stability approach reasoned that information could not travel across 2 grid points in a single time step giving the condition, $\Delta t \leq \Delta x / |v|$. In 2D it seems reasonable to take,

$$\Delta t \leq F \min(\Delta x / |v_x|, \Delta y / |v_y|) \quad (6.10a)$$

where $F < 1$, is a ‘safety factor’ which can be determined from numerical experiments if necessary. In principle it is possible to undertake a stability analysis for a 2D scheme in the same way as for the corresponding 1D scheme. This is algebraically complicated. A von Neumann analysis gives,

$$C_x + C_y \leq 1 \quad (6.10b)$$

which is a very restrictive condition on Δt .

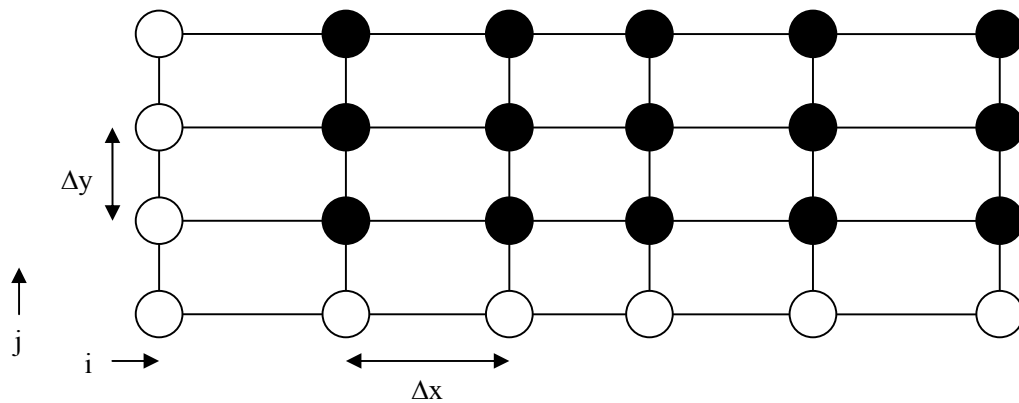


Figure 6.1 2D Computational mesh showing real grid points (black) and ghost grid points (white) for the FOU2D scheme

Code to implement the FOU2D scheme (6.9) may be downloaded from the website. Graphical output from the above scheme is given in Figure 6.2.

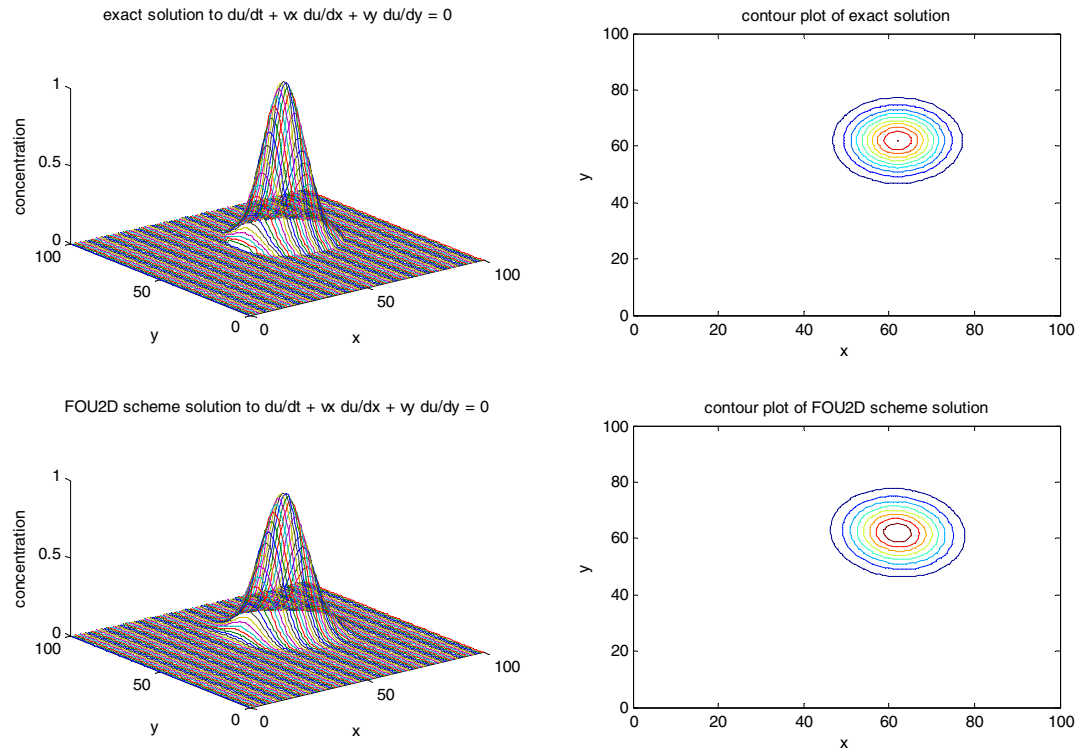


Figure 6.2 Comparison of exact and numerical solutions for the (unsplit) FOU2D scheme

6.2.2 Example 2: Lax-Friedrichs Scheme in 2D

From our FD toolkit we choose,

$$\delta_x u_{i,j}^n = \frac{u_{i+1,j}^n - u_{i-1,j}^n}{2\Delta x}, \delta_y u_{i,j}^n = \frac{u_{i,j+1}^n - u_{i,j-1}^n}{2\Delta y}, \delta_{xx} u_{i,j}^n = \delta_{yy} u_{i,j}^n = \delta_{xy} u_{i,j}^n = 0$$

and estimate $u_{i,j}^n$ by $\frac{u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n}{4}$ (i.e. the mean of surrounding values). Equation (6.8b)

becomes,

$$u_{i,j}^{n+1} = \frac{u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n}{4} - \frac{C_x}{2}(u_{i+1,j}^n - u_{i-1,j}^n) - \frac{C_y}{2}(u_{i,j+1}^n - u_{i,j-1}^n) \quad (6.11)$$

Notes:

1. The scheme is first order in t and in both x and y .
2. Ghost values are required at all sides of the computational domain.
3. This scheme is less diffusive than the FOU2D scheme and also works for positive and negative velocities.
4. A von Neumann stability analysis gives,

$$C_x^2 + C_y^2 \leq 1/2 \quad (6.12)$$

which is a very restrictive time step condition.

Please click the advert



May we offer you one of the world's greatest challenges?
In all humility.

We are looking for highly educated engineers and finance students.

Join us at dongenergy.com/job

dongenergy.com/job




6.2.3 Example 3: Crank-Nicolson Scheme in 2D

By an obvious extension of the 1D scheme (6.8b) becomes,

$$u_{i,j}^{n+1} = u_{i,j}^n - \frac{C_x}{2} \left(\frac{u_{i+1,j}^n - u_{i-1,j}^n}{2} + \frac{u_{i+1,j}^{n+1} - u_{i-1,j}^{n+1}}{2} \right) - \frac{C_y}{2} \left(\frac{u_{i,j+1}^n - u_{i,j-1}^n}{2} + \frac{u_{i,j+1}^{n+1} - u_{i,j-1}^{n+1}}{2} \right) \quad (6.13)$$

Notes:

1. The scheme is first order in t and second order in both x and y .
2. Ghost values are required at all sides of the computational domain.
3. The scheme is *implicit* so values at time level $n+1$ are found by solving a system of linear equations.
4. Von Neumann stability analysis shows that this scheme is unconditionally stable.
5. Implementation of the Crank-Nicolson scheme involves solving MN linear equations in MN unknowns at each time step where MN is the number of computational grid points. The matrix can be written as a penta-diagonal matrix. For large MN the computational effort at each time step may be prohibitive unless an efficient matrix inversion algorithm is used.

6.2.4 Conclusions

From the above analysis we can see that FD schemes extend naturally from 1D to 2D (and 3D). The main issues are:

1. It is often difficult to perform stability analysis.
2. For explicit schemes the allowable time step may be prohibitively small.
3. For implicit schemes the computational effort at each time step may be prohibitively large.
4. It would be nice to be able to exploit the relative simplicity and larger time steps of 1D schemes for 2D problems. The following analysis shows that we can actually do this and more!

6.3 Operator Splitting (Approximate Factorisation)

This method replaces a single scheme to solve a complicated PDE by a *sequence* of simpler schemes which solve related PDEs and which together solve the original PDE (up to a specified order of accuracy). Operator splitting can be used to split up a PDE by dimension or by components or a combination of both. We begin by looking at dimensional splitting.

6.3.1 Dimensional Splitting

The basis of this method is to split our N dimensional PDE into N , 1D PDEs and use a sequence of 1D FD schemes to solve the problem. For simplicity we will look at the 2D linear advection equation (6.1a). We split the (6.1a) into the following 2 PDEs (which are both 1D in space),

$$U_t + v_x U_x = 0 \quad (6.14a)$$

$$U_t + v_y U_y = 0 \quad (6.14b)$$

From our 1D work in Chapter 4 we know that a (second order in time) differential marching operator for solving (6.14a) is,

$$L_x(\Delta t) = 1 - \Delta t v_x \partial_x + \frac{\Delta t^2}{2} v_x^2 \partial_{xx} \quad (6.15a)$$

Similarly a differential marching operator for solving (6.14b) is,

$$L_y(\Delta t) = 1 - \Delta t v_y \partial_y + \frac{\Delta t^2}{2} v_y^2 \partial_{yy} \quad (6.15b)$$

Consider the operator sequence,

$$L_y(\Delta t) L_x(\Delta t) \quad (6.16)$$

We show that this sequence is ‘the same’ as L_{xy} defined in (6.4) which we know solves the 2D PDE (6.1a). By ‘the same’ it is enough to show that the difference is of an appropriate order of Δt . Multiplying out (6.16) gives,

$$L_y(\Delta t) L_x(\Delta t) = \left(1 - \Delta t v_y \partial_y + \frac{\Delta t^2}{2} v_y^2 \partial_{yy}\right) \left(1 - \Delta t v_x \partial_x + \frac{\Delta t^2}{2} v_x^2 \partial_{xx}\right) \quad (6.17a)$$

$$= 1 - \Delta t (v_x \partial_x + v_y \partial_y) + \frac{\Delta t^2}{2} (v_x^2 \partial_{xx} + 2v_x v_y \partial_{xy} + v_y^2 \partial_{yy}) + O(\Delta t^3) \quad (6.17b)$$

$$= L_{XY}(\Delta t) + O(\Delta t^3) \quad (6.17c)$$

Hence the sequence of 1D operators, $L_Y(\Delta t) L_X(\Delta t)$, is the same as the 2D operator $L_{XY}(\Delta t)$ up to $O(\Delta t^3)$. This shows that the *split* scheme,

$$u_{i,j}^{n+1} = L_Y(\Delta t) [L_X(\Delta t) u_{i,j}^n] \quad (6.18)$$

can be used to solve (6.1a) instead of the unsplit scheme (6.8a). It is now just a matter of replacing the differential operators, by appropriate difference operators like we did before: e.g. L_X becomes the difference marching operator,

$$L_X(\Delta t) = 1 - \Delta t v_x \delta_x + \frac{\Delta t^2}{2} v_x^2 \delta_{xx}.$$

Please click the advert



Ambitious and curious?

We are looking for young and highly educated engineers and finance students.

Join us at dongenergy.com/job

dongenergy.com/job





Notes:

1. (6.18) has the same temporal order as (6.8a).
2. The time step in (6.18) is the *minimum* allowable time step for each 1D operator and may be much larger than the time step for the unsplit 2D scheme (6.8a) which is one of the advantages of dimensional splitting.
3. (6.18) is implemented by first applying the L_X operator to each *row* of the grid. This produces new values of the dependent variable at each grid point. The L_Y operator is then applied to the new data along each *column* of the grid.
4. The split operator sequence (6.18) is one of many split operator sequences which solve (6.1a).
5. L_X and L_Y may be totally different FD schemes (although they should be the same order).
6. Our approach is called ‘dimensional splitting’ because a 2D PDE (6.1a) has been split into 2, 1D PDEs (6.14a, 6.14b).

6.3.1.1 Example 4: Split FOU2D Scheme

The FOU scheme is used for both L_X and L_Y difference operators. i.e. (for positive velocities) we choose,

$$\delta_x u_{i,j}^n = \frac{u_{i,j}^n - u_{i-1,j}^n}{\Delta x}, \delta_y u_{i,j}^n = \frac{u_{i,j}^n - u_{i,j-1}^n}{\Delta y}, \delta_{xx} u_{i,j}^n = \delta_{yy} u_{i,j}^n = 0. \text{ So that,}$$

$$L_X(\Delta t) = 1 - \Delta t v_x \delta_x, L_Y(\Delta t) = 1 - \Delta t v_y \delta_y.$$

Neglecting terms of $O(\Delta t^2)$ the split sequence (6.16) is,

$$\begin{aligned} L_Y(\Delta t) L_X(\Delta t) &= (1 - \Delta t v_x \delta_x)(1 - \Delta t v_y \delta_y) \\ &= 1 - \Delta t (v_x \delta_x + v_y \delta_y) \end{aligned}$$

Equation (6.18) can be written,

$$\text{x sweep: } \bar{u}_{i,j} = u_{i,j}^n - C_x (u_{i,j}^n - u_{i-1,j}^n) \quad (6.19a)$$

$$\text{y sweep: } u_{i,j}^{n+1} = \bar{u}_{i,j} - C_y (\bar{u}_{i,j} - \bar{u}_{i,j-1}) \quad (6.19b)$$

where,

$\bar{u}_{i,j}$ ('u bar') is the new data at the grid point i,j produced by the L_x operator and $C_x = \frac{v_x \Delta t}{\Delta x}$, $C_y = \frac{v_y \Delta t}{\Delta y}$ are the usual Courant numbers.

Notes:

1. (6.19a) corresponds to $\bar{u}_{i,j} = L_x(\Delta t)u_{i,j}^n$ and (6.19b) corresponds to $u_{i,j}^{n+1} = L_y(\Delta t)\bar{u}_{i,j}$.

2. $\Delta t \leq \min\left(\frac{\Delta x}{|v_x|}, \frac{\Delta y}{|v_y|}\right)$ where $\frac{\Delta x}{|v_x|}, \frac{\Delta y}{|v_y|}$ are the maximum time steps for the L_x and L_y operators

respectively. These expressions were found previously by von Neumann stability analysis of the 1D FOU scheme.

Code to implement the split FOU2D scheme (6.19a, 6.19b) may be downloaded from the website.

6.3.1.2 Operator Splitting for Stiff Problems

The allowable time step, Δt , for an explicit scheme depends on the 'flow' velocity and the grid spacing in each direction. Ideally we want to time-march with as large a time step as possible for computational speed. This cannot always be achieved by the 'standard' splitting (6.18) as we will see from the following example. Let the allowable time steps in x and y directions be Δt_x and Δt_y respectively. For the split

FOU2D scheme, $\Delta t = \min(\Delta t_x, \Delta t_y)$ where $\Delta t_x = \frac{\Delta x}{|v_x|}$, $\Delta t_y = \frac{\Delta y}{|v_y|}$. Now suppose that $|v_x| \gg |v_y|$ (\gg means

'much greater than') then, assuming that the grid spacings are of the same order,

$$\Delta t_x \ll \Delta t_y.$$

This defines a 'stiff' problem: the time scales for each component of the problem are very different. The 'standard' split operator sequence,

$$L_x(\Delta t) L_y(\Delta t) \quad (6.20)$$

requires that each operator marches using the *same* time step which has to be the minimum of Δt_x and Δt_y for stability. Clearly the time step for L_Y is being *constrained* by the much smaller allowable time step in the x direction. It might be more efficient if we could change the splitting so that L_Y could march in steps of Δt_y . We can!

Suppose that $n\Delta t_x = \Delta t_y$ where n is an integer. i.e. Δt_x is n times as small as Δt_y . Consider the new split operator sequence,

$$L_Y(\Delta t_y)[L_X(\Delta t_y / n)]^n \quad (6.21)$$

In this sequence L_Y is marching at its maximum time step, Δt_y , and L_X is marching at $(1/n)$ of Δt_y so to ‘catch up’ we need to use L_X n times. The following analysis shows that this approach works.

dongenergy.com/job

**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

Everybody is talking...

...about future energy supply. We are not.
Stop talking and make a career
moving energy forward.

Ambitious engineers and
finance students go to
dongenergy.com/job



Please click the advert

From the split FOU2D scheme example,

$L_X(\Delta t) = 1 - \Delta t v_x \delta_x$, $L_Y(\Delta t) = 1 - \Delta t v_y \delta_y$. Substituting into (6.21) and using the Binomial expansion gives,

$$\begin{aligned}
 & (1 - \Delta t_y v_y \delta_y) (1 - (\Delta t_y / n) v_x \delta_x)^n \\
 &= (1 - \Delta t_y v_y \delta_y) (1 - n(\Delta t_y / n) v_x \delta_x + O(\Delta t^2)) \\
 &= (1 - \Delta t_y v_y \delta_y) (1 - \Delta t_y v_x \delta_x + O(\Delta t^2)) \\
 &= 1 - \Delta t_y (v_y \delta_y + v_x \delta_x) + O(\Delta t^2)
 \end{aligned} \tag{6.22}$$

which is $L_Y(\Delta t_y) L_X(\Delta t_y)$ up to $O(\Delta t^2)$. This shows that the sequence (6.21) is valid.

Notes:

1. If Δt_x is not exactly a whole number times as small as Δt_y replace Δt_y by $n\Delta t_x$ where n is the largest integer such that $n\Delta t_x < \Delta t_y$.
2. (6.21) says apply the operator L_X to each row of the grid n times using a time step $\Delta t_y/n$ each time and using the updated values of u at each application. Then apply the L_Y operator to each column of the grid once using a time step Δt_y .

3. Since the multiplication in (6.22) is commutative another valid operator sequence is,

$$[L_X(\Delta t_y / n)]^{n-m} L_Y(\Delta t_y) [L_X(\Delta t_y / n)]^m.$$

4. If L_X and L_Y are second order (they aren't in our example) and n is even then the symmetric sequence,

$$[L_X(\Delta t_y / n)]^{n/2} L_Y(\Delta t_y) [L_X(\Delta t_y / n)]^{n/2}$$

is also second order. Where possible it is almost always better to use symmetric sequences.

6.3.1.3 Exercise 6a

1. Show that (6.1b) is a solution to (6.1a).
2. Go through the calculations to get (6.2e).
3. Derive (6.3b).

4. Write out (6.8a) in full to get (6.8b).
- 5a. Download ‘linearadvectionFOU2D’ and make sure you understand every line of code.
- 5b. Run ‘linearadvectionFOU2D’ and describe the output for the given parameters.
- 5c. By gradually increasing the safety factor F and running the code find its maximum value for a stable scheme based on the heuristic time step formula.
- 5d. Find the maximum time step for the FOU2D scheme based on the von Neumann stability analysis result and alter your program to use this time step.
- 5e. Change the sign of the x component of velocity and run the code. What happens to the numerical solution and why?
 - i) Fix the problem so that your code can deal with $v_x < 0$ and $v_y > 0$. Test your program.
 - ii) Fix the problem so that your code can deal with any velocities. Test your program.
6. Find the expression for the time step based on von Neumann stability analysis for the 2D Lax-Friedrichs scheme to solve the 2D linear advection equation.
7. Copy ‘linearadvectionFOU2D’ to ‘linearadvectionLF2D’ and change the solver in this new file so that it uses the 2D Lax-Friedrichs scheme (don’t forget ghost values) and repeat Q5.
8. By multiplying out and collecting terms, derive 6.17c from 6.17a.
9. Show that $L_X(\Delta t) L_Y(\Delta t)$, is the same as $L_{XY}(\Delta t)$ up to $O(\Delta t^3)$ and hence write down another operator sequence to solve the 2D linear advection equation.
10. Show that for the 2D linear advection equation with equal velocities in x and y directions, the time step for the $L_X L_Y$ split FOU2D scheme is twice that for the unsplit FOU2D scheme when grid spacings in x and y directions are equal.
11. Compare time steps in Q10 when the Lax-Friedrichs scheme is used for all operators.
12. Using the notation of (6.19ab) write down the FD scheme to solve the 2D linear advection equation with the Lax-Friedrichs scheme for all operators using the $L_X L_Y$ sequence.

13. Copy a suitable previous code to 'linearadvectionFOU2Dsplit' and modify it to solve the 2D linear advection equation using the split operator sequence, $L_Y(\Delta t) L_X(\Delta t)$, using the FOU scheme for each operator.
14. Copy a suitable previous code to 'linearadvectionFOU2Dsplit' and change it to solve the 2D linear advection equation using the split operator sequence $[L_X(\Delta t/2)]L_Y(\Delta t)[L_X(\Delta t/2)]$ using the Lax-Friedrichs scheme for each operator.

6.3.2 Term Splitting

PDEs may contain several terms corresponding to different physical processes. As an example we use the 1D advection-diffusion equation (5.1). Rather than solving the advection-diffusion equation directly it may be more efficient to use *term splitting* to solve each part separately and combine solutions. The following time step analysis shows why this could be a preferred option.

Assuming $v > 0$, using a first order forward difference for the time derivative and a first order backward difference for the first space derivative, by previous analysis the time step for pure advection is,

$$\Delta t_A \leq \Delta x / v \quad (6.23a)$$

Please click the advert



Dear highly educated engineering and finance students,

if you are driven, ambitious, open-minded and focused - we have a challenge for you. Actually, the greatest challenge in the world. Curious? Visit dongenergy.com/job

Best wishes

DONG Energy

dongenergy.com/job





Using a first order forward difference for the time derivative and a symmetric difference for the space derivative, by previous analysis the time step for pure diffusion is,

$$\Delta t_D \leq \frac{\Delta x^2}{2K_x} \quad (6.23b)$$

Assuming $v > 0$, the time step for the unsplit scheme for advection-diffusion is,

$$\Delta t_{AD} \leq \frac{\Delta x^2}{v\Delta x + 2K_x} \quad (6.23c)$$

From which we can see that,

$$\Delta t_{AD} \leq \Delta t_A \quad (6.24a)$$

and,

$$\Delta t_{AD} \leq \Delta t_D \quad (6.24b)$$

This leads us to conclude that the corresponding direct (unsplit) scheme will have a smaller allowable time step than the split scheme (note also that as Δx gets smaller Δt_{AD} becomes much smaller than Δt_A). From our previous theory of split schemes we can derive a split scheme for the advection diffusion equation (5.1). We replace (5.1) by two PDEs,

$$U_t + vU_x = 0 \quad (6.25a)$$

and

$$U_t = K_x U_{xx} \quad (6.25b)$$

(6.25a) is the pure advection equation and (6.25b) is the pure diffusion equation. Let $L_A(\Delta t_A)$ and $L_D(\Delta t_D)$ be FD operators for solving (6.25a) and (6.25b) respectively. We will show that $L_D(\Delta t) L_A(\Delta t)$ is a (term) split operator sequence for solving (5.1) where $\Delta t = \min(\Delta t_A, \Delta t_D)$. From the standard Taylor expansion with x constant,

$$U(t + \Delta t, x) = U(t, x) + \Delta t \partial_t U + \frac{\Delta t^2}{2} \partial_{tt} U + O(\Delta t^3) \quad (6.26)$$

Firstly we use the advection-diffusion equation (5.1) to replace the time derivatives in (6.26) by spatial derivatives. (5.1) can be written,

$$\partial_t U = (-v \partial_x + K_x \partial_{xx}) U \quad (6.27a)$$

Hence,

$$\begin{aligned} \partial_{tt} U &= (\partial_t \partial_t) U = (-v \partial_x + K_x \partial_{xx}) (-v \partial_x + K_x \partial_{xx}) U \\ &= (v^2 \partial_{xx} - 2v K_x \partial_{xxx} + K_x^2 \partial_{xxxx}) U \end{aligned} \quad (6.27b)$$

Substituting (6.27ab) into (6.26) gives,

$$U(t + \Delta t, x) = U(t, x) + \Delta t (-v \partial_x + K_x \partial_{xx}) U + \frac{\Delta t^2}{2} (v^2 \partial_{xx} - 2v K_x \partial_{xxx} + K_x^2 \partial_{xxxx}) U + O(\Delta t^3) \quad (6.28)$$

$$\therefore U(t + \Delta t, x) = L_{AD}(\Delta t) U(t, x) + O(\Delta t^3) \quad (6.29a)$$

where $L_{AD}(\Delta t)$ is an unsplit *differential* marching operator to solve the advection-diffusion equation given by,

$$L_{AD}(\Delta t) = 1 + \Delta t (-v \partial_x + K_x \partial_{xx}) + \frac{\Delta t^2}{2} (v^2 \partial_{xx} - 2v K_x \partial_{xxx} + K_x^2 \partial_{xxxx}) + O(\Delta t^3) \quad (6.29b)$$

(6.29b) is made into a *difference* marching operator by replacing all partial derivatives by FD approximations. By previous work we know that a *differential* marching operator, $L_A(\Delta t)$, to solve the advection equation (6.25a) is given by,

$$L_A(\Delta t) = 1 - \Delta t v \partial_x + \frac{\Delta t^2}{2} v^2 \partial_{xx} + O(\Delta t^3) \quad (6.30)$$

We derive a *differential* marching operator, $L_D(\Delta t)$, to solve the diffusion equation (6.25b). (6.25b) can be written,

$$\partial_t U = K_x \partial_{xx} U \quad (6.31a)$$

hence,

$$\begin{aligned} \partial_{tt} U &= \partial_t \partial_t U = (K_x \partial_{xx}) (K_x \partial_{xx}) U \\ &= K_x^2 \partial_{xxxx} U \end{aligned} \quad (6.31b)$$

Substituting (5.18ab) into (6.26) gives,

$$U(t + \Delta t, x) = U(t, x) + \Delta t K_x \partial_{xx} U + \frac{\Delta t^2}{2} K_x^2 \partial_{xxxx} U + O(\Delta t^3) \quad (6.32)$$

$$\therefore U(t + \Delta t, x) = L_D(\Delta t) U(t, x) + O(\Delta t^3) \quad (6.33a)$$

where $L_D(\Delta t)$ is an unsplit *differential* marching operator to solve the diffusion equation given by,

$$L_D(\Delta t) = 1 + \Delta t K_x \partial_{xx} + \frac{\Delta t^2}{2} K_x^2 \partial_{xxxx} + O(\Delta t^3) \quad (6.33b)$$

Please click the advert

dongenergy.com/job



May we offer you one of the world's greatest challenges?
In all humility.

We are looking for highly educated engineers and finance students.

Join us at dongenergy.com/job





(6.33b) is made into a *difference* marching operator by replacing all partial derivatives by FD approximations. It now remains to show that $L_{AD}(\Delta t)$ is approximated by the split sequence $L_D(\Delta t) L_A(\Delta t)$.

$$\begin{aligned} L_D(\Delta t)L_A(\Delta t) &= \left(1 + \Delta t K_x \partial_{xx} + \frac{\Delta t^2}{2} K_x^2 \partial_{xxxx} + O(\Delta t^3)\right) \left(1 - \Delta t v \partial_x + \frac{\Delta t^2}{2} v^2 \partial_{xx} + O(\Delta t^3)\right) \\ &= 1 + \Delta t K_x \partial_{xx} + \frac{\Delta t^2}{2} K_x^2 \partial_{xxxx} - \Delta t v \partial_x - \Delta t^2 v K_x \partial_{xxx} + \frac{\Delta t^2}{2} v^2 \partial_{xx} + O(\Delta t^3) \\ &= L_{AD}(\Delta t) + O(\Delta t^3) \end{aligned}$$

Hence the unsplit operator and the split operator sequence are the same up to $O(\Delta t^3)$.

Using the previous FD schemes the corresponding split scheme to solve the 1D advection-diffusion equation (5.1) is,

$$L_A: \bar{u}_i = u_i^n - \frac{v \Delta t}{\Delta x} (u_i^n - u_{i-1}^n) \quad (6.34a)$$

$$L_D: u_i^{n+1} = \bar{u}_i^n + \frac{K_x \Delta t}{\Delta x^2} (\bar{u}_{i+1} - 2\bar{u}_i + \bar{u}_{i-1}) \quad (6.34b)$$

where,

$$\Delta t = \min\left(\frac{\Delta x}{v}, \frac{\Delta x^2}{2K_x}\right). \quad (6.35)$$

Output is given in Figure 6.3 for the same conditions as for Figure 5.3. It can be seen that the peak concentration is in the correct place as before. This was achieved in fewer time steps. However the shape of the concentration profile is slightly different to that given by the unsplit scheme - it is higher and narrower. Perfect numerical schemes would give the same results for split and unsplit algorithms but no scheme is perfect and the effect of numerical diffusion in the advective solvers plays a different role in each algorithm.

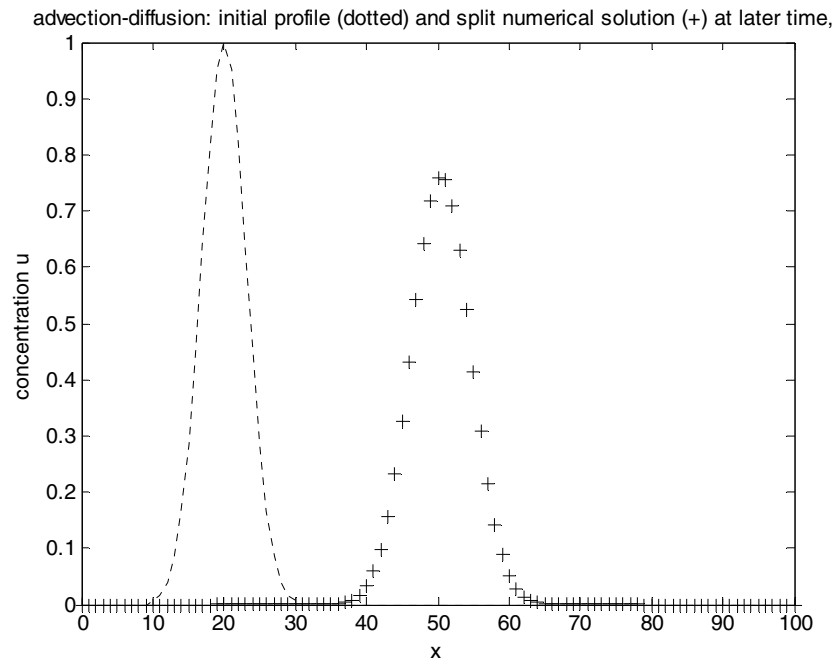


Figure 6.3 Initial profile and split numerical solution (+) to the 1D advection-diffusion equation at $t = 57s$.

6.3.3 Exercise 6b

- For $v > 0$ show that, a) $\lim_{\Delta x \rightarrow 0} \frac{\Delta t_A}{\Delta t_{AD}} = \infty$, b) $\lim_{\Delta x \rightarrow 0} \frac{\Delta t_D}{\Delta t_{AD}} = 1$.

What do you conclude from this?

- Go through the derivation of L_{AD} , L_D and L_A and show that

$$L_{AD} = L_D L_A + O(\Delta t^3).$$

- Write a program for the split scheme (6.34ab) and reproduce Figure 5.3.

7. Systems of Equations

7.1 Introduction

The world is governed by natural laws many of which can be expressed as *systems* of PDEs. An important example are the Navier-Stokes equations that, together with the Continuity Equation form a system of 6 coupled PDEs which describe fluid flow in 3D. These equations are difficult to solve even approximately. In the following, a relatively simple system of PDEs has been chosen to illustrate the FD approach.

7.2 The Shallow Water Equations

The Shallow Water Equations (SWE) may be expressed in 1D or 2D and provide a simplified model of water flow which may be used to simulate many situations including river flow and tsunami propagation. For simplicity we consider only 1D. This is still useful and there are many 1D SWE software packages used by hydraulic engineers to make flow calculations for real life applications.

The 1D SWE form a system of two coupled *non-linear* hyperbolic PDEs with independent variables, t (time) and x (distance along the flow) and dependent variables $h = h(t, x)$ (flow depth) and $v = v(t, x)$ (flow velocity).

Please click the advert

dongenergy.com/job



**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

Ambitious and curious?

We are looking for young and highly educated engineers and finance students.

Join us at dongenergy.com/job



Using our usual subscript notation to indicate partial differentiation the 1D SWE can be written compactly as,

$$\underline{U}_t + \underline{F}(\underline{U})_x = \underline{S}(\underline{U}) \quad (7.1)$$

where,

$$\underline{U} = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} h \\ hv \end{bmatrix}, \quad (7.2a)$$

$$\underline{F}(\underline{U}) = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = \begin{bmatrix} hv \\ hv^2 + g h^2 / 2 \end{bmatrix}, \quad (7.2b)$$

$$\underline{S}(\underline{U}) = \begin{bmatrix} 0 \\ g(S_0 - S_f) \end{bmatrix} \quad (7.2c)$$

where g is the acceleration due to gravity, \underline{U} is the matrix of dependent variables, $\underline{F}(\underline{U})$ is called the flux vector and $\underline{S}(\underline{U})$ is called the matrix of source terms which here only consists of bed slope (S_0 , measured positive *downwards*) and friction (S_f) terms.

7.3 Solving the Shallow Water Equations

7.3.1 Theoretical Background

Explicit FD schemes need a limited time step for stability. Previous stability analyses assume linearity but the SWE are not linear and so we cannot use these approaches directly. It is therefore necessary to examine the SWE in some detail to determine stable time steps before looking at explicit FD schemes. Neglecting source terms and using the Chain-Rule (7.1) can be expressed in *quasi-linear* form as,

$$\underline{U}_t + \underline{J}\underline{U}_x = \underline{0} \quad (7.3)$$

where \underline{J} is the Jacobian matrix given by,

$$\underline{J} = \begin{bmatrix} \frac{\partial \underline{F}}{\partial \underline{U}} \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial U_1} & \frac{\partial F_1}{\partial U_2} \\ \frac{\partial F_2}{\partial U_1} & \frac{\partial F_2}{\partial U_2} \end{bmatrix}. \quad (7.4)$$

It is easily shown that,

$$J = \begin{bmatrix} 0 & 1 \\ -v^2 + gh & 2v \end{bmatrix} \quad (7.5)$$

J has two real eigenvalues,

$$\lambda_1 = v + \sqrt{gh}, \lambda_2 = v - \sqrt{gh} \quad (7.6)$$

These eigenvalues are real and distinct and therefore give rise to corresponding linearly independent eigenvectors \underline{q}_1 and \underline{q}_2 (which we write as column vectors). By standard Linear Algebra theory there is a transformation of coordinates that diagonalizes J . This is now shown. Let, $T = [\underline{q}_1 \underline{q}_2]$, then,

$$\begin{aligned} J T &= J [\underline{q}_1 \underline{q}_2] \\ &= [J \underline{q}_1 \ J \underline{q}_2] \\ &= [\lambda_1 \underline{q}_1 \ \lambda_2 \underline{q}_2] \text{ (by definition of eigenvalue)} \\ &= [\underline{q}_1 \ \underline{q}_2] \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \\ &= T \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \end{aligned}$$

therefore,

$$T^{-1} J T = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} = D \quad (7.7)$$

We introduce a change of variable by defining $\underline{W} = \begin{bmatrix} W_1 \\ W_2 \end{bmatrix}$ by,

$$\underline{W} = T^{-1} \underline{U} \quad (7.8)$$

Assuming that T^{-1} is locally constant, then (7.3) becomes,

$$T \underline{W}_t + J T \underline{W}_x = \underline{0} \quad (7.9a)$$

Multiplying through on the left by T^{-1} and using (7.7) gives,

$$\underline{W}_t + D\underline{W}_x = \underline{0} \quad (7.9b)$$

Since D is a diagonal matrix this system of equations has been decoupled by the change of variable. Writing out each row of (7.9b) gives,

$$\frac{\partial W_1}{\partial t} + (v + \sqrt{gh}) \frac{\partial W_1}{\partial x} = 0 \quad (7.10a)$$

$$\frac{\partial W_2}{\partial t} + (v - \sqrt{gh}) \frac{\partial W_2}{\partial x} = 0 \quad (7.10b)$$

(7.10a) and (7.10b) are *two* linear advection equations with *two* wave speeds $(v + \sqrt{gh}), (v - \sqrt{gh})$ respectively. Any explicit scheme to solve the SWE must take account of the above wave speeds to ensure stability. Note that these wave speeds vary over both space and time.

dongenergy.com/job

**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

Everybody is talking...

...about future energy supply. We are not.
Stop talking and make a career
moving energy forward.

Ambitious engineers and
finance students go to
dongenergy.com/job



7.3.2 Heuristic Time Step Calculation

The following is a treatment of the time step calculation for the SWE based on the previous heuristic analysis of Appendix C. Suppose that a given explicit scheme to solve the advection equation $u_t + v u_x = 0$ is stable when,

$$\Delta t < \Delta x / (5v) \quad (7.11)$$

where v is the wave speed (assume $v > 0$) and Δt and Δx are the time and space steps respectively. By the previous analysis, the *same* scheme will be stable when solving the SWE if the *maximum* wave speed (chosen over the spatial interval at a given time level) is used in the stability inequality.

$$\text{i.e. } \Delta t < \frac{\Delta x}{5 \max_i \{ (v + \sqrt{gh}), (v - \sqrt{gh}) \}} \quad (7.12)$$

Furthermore, because the wave speeds also vary with time, this constraint must be applied *at each time step*.

7.4 Example Scheme to Solve the SWE

The SWE (7.1) with $\underline{S}(\underline{U}) = \underline{0}$ look very similar to the linear advection equation, $u_t + v u_x = 0$. We see that U_t is replaced by u_t and F_x is replaced by $v u_x$. The FOU scheme (Chapter 4) to solve the linear advection equation is,

$$u_i^{n+1} = u_i^n - \frac{\Delta t v (u_i^n - u_{i-1}^n)}{\Delta x} \quad (7.13a)$$

The FOU scheme can be shown to be stable (for $v > 0$) if,

$$\Delta t < \Delta x / v. \quad (7.13b)$$

We apply the *same* scheme (in the sense that the time derivative is replaced by the first order forward difference approximation and the spatial derivative is replaced by the first order backward difference approximation) to solve the SWE giving,

$$\underline{U}_i^{n+1} = \underline{U}_i^n - \frac{\Delta t (\underline{F}_i^n - \underline{F}_{i-1}^n)}{\Delta x} \quad (7.14a)$$

By the previous analysis the time step inequality (7.13b) becomes,

$$\Delta t < \frac{\Delta x}{\max_i \{v + \sqrt{gh}, v - \sqrt{gh}\}} \quad (7.14b)$$

(7.14b) *must* be applied at each time step.

The above analysis shows that a program written to solve the linear advection equation may be modified to solve the SWE by appending an extra row to each array variable matrix. Prior to each time step a routine to calculate the new time step must be invoked. After each time step h and v are found from U ($h=U_1$, $v=U_2/U_1$) and then U and F are initialized before the next time step.

Figure 7.1 shows the Lax-Friedrichs scheme solution to the SWE for a collapsing water column (not shown is the graph for the associated water velocity). Initially the water is still and the surface profile is given in Figure 7.1. The program was run for 6 seconds using 201 grid points over $[0, 200]$ and a time step safety factor of 0.95. Zero-gradient boundary conditions were used for both water height and velocity. Note that this problem has discontinuous initial conditions for h that can cause classical schemes problems (see the oscillations in Figure 7.1). There exist modern schemes to cope with discontinuities but they are beyond the scope of this text.

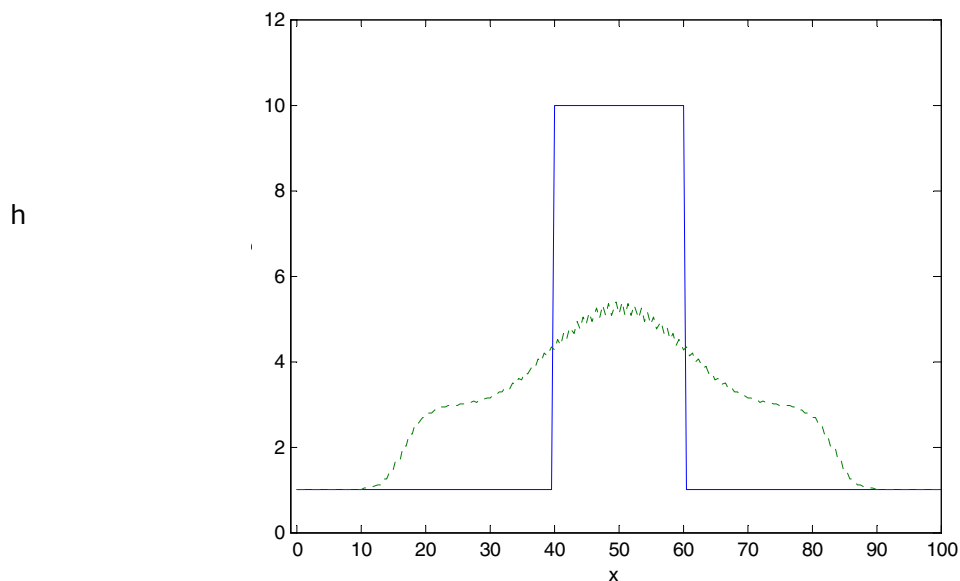


Figure 7.1 Lax-Friedrichs solution to the Shallow Water Equations for a collapsing water column. Initial conditions (—), numerical solution (---) at $t = 6s$.

Code to implement the above scheme may be downloaded. A Case Study to illustrate the use of the 1D SWE is given on the website. This study also gives useful information on how to set up a problem and assess results.

7.5 Exercise 7

1. Show that J is as given and use the Chain Rule to express (7.1) in the form of (7.3).
2. Find the eigenvalues of J .
3. Given that an explicit scheme to solve the linear advection equation with wave speed v is stable for $\Delta t < 2\Delta x / (3v)$ state the (heuristic) stability criteria when this scheme is used to solve the SWE.
4. Comment on the sign of the wave speeds for the SWE when the flow is such that $v > (gh)^{1/2}$.
5. Modify one of your existing codes to solve the linear advection equation using the Lax-Friedrichs algorithm (use previous initial and boundary conditions). Verify your code.
6. Modify your program in Q5 to solve the SWE using the conditions of Figure 7.1. Reproduce Figure 7.1.
7. Animate the solutions for height and velocity in Q6 for and run until the water passes out of the domain.
8. Repeat Q7 with solid left and right hand boundary conditions (see Appendix B).

Appendix A: Definition and Properties of Order

A.1 Definition of $O(h^n)$

For our purposes,

$$f(h) = O(h^n)$$

(pronounced, ‘f of h is order h to the n’) means,

$$\lim_{h \rightarrow 0} \frac{f(h)}{h^n} = C, \quad (\text{A.1})$$

where C is a *non-zero* constant.

e.g. $500 h^6 + 3 h^4 - 2 h = O(h)$ because, $\lim_{h \rightarrow 0} \frac{500 h^6 + 3 h^4 - 2 h}{h} = -2$

e.g. $9h^4 = O(h^4)$ because, $\lim_{h \rightarrow 0} \frac{9h^4}{h^4} = 9$

Please click the advert

dongenergy.com/job




**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

Dear highly educated engineering and finance students,

if you are driven, ambitious, open-minded and focused - we have a challenge for you. Actually, the greatest challenge in the world. Curious? Visit dongenergy.com/job

Best wishes

DONG Energy



A.2 The Meaning of $O(h^n)$

If $f(h) = O(h^n)$ then, for small h , equation (A.1) gives,

$$\frac{f(h)}{h^n} \approx C,$$

$$\therefore f(h) \approx C h^n. \quad (\text{A.2})$$

Equation (A.2) says that for small h , an error which is $O(h^n)$ is proportional to h^n . In particular if the error is $O(h)$ then it is proportional to h which means that halving h halves the error. If the error is $O(h^2)$ then it is proportional to h^2 which means that halving h reduces the error by a factor of $2^2 = 4$.

A.3 Properties of $O(h^n)$

In the analysis of finite difference schemes we will need to use the following properties of Order notation. Let $f(h) = O(h^n)$, $g(h) = O(h^m)$ where $0 < m < n$. Let K be a non-zero constant. Then,

A.3.1. $K f(h) = O(h^n)$

A.3.2. $f(h) + g(h) = O(h^m)$

A.3.3. $f(h)/h = O(h^{n-1})$

A.3.4. $f(h)/g(h) = O(h^{n-m})$

A.3.5. $f(h) g(h) = O(h^{n+m})$

Proof of A.3.3

By definition of Order we have to show that,

$$\lim_{h \rightarrow 0} \frac{f(h)/h}{h^{n-1}} \text{ is a non-zero constant.}$$

Since $f(h) = O(h^n)$,

$$\lim_{h \rightarrow 0} \frac{f(h)}{h^n} = C, \text{ where } C \text{ is a non-zero constant.}$$

$$\begin{aligned}\therefore \lim_{h \rightarrow 0} \frac{f(h)/h}{h^{n-1}} &= \lim_{h \rightarrow 0} \frac{f(h)}{h^n} \\ &= C\end{aligned}$$

End of proof of A.3.3

A.4 Explanation of the Properties of $O(h^n)$

A.3.1 says that scaling a function by a constant doesn't change its order. In particular $f(h)$ and $-f(h)$ have the same order.

A.3.2 says that the order of the sum of two functions of different orders is the smaller of the orders of the two functions. e.g.

$$O(h^2) + O(h^3) = O(h^2).$$

A.3.3 says that dividing a function by h reduces its order by 1. A.3.3 is a special case of A.3.4 that says that dividing a function by a function of order m reduces its order by m . e.g. $O(h^6)/h^2 = O(h^4)$.

A.3.5 says that the order of the product of two functions is the sum of their orders. e.g.

$$O(h^3) O(h^2) = O(h^5).$$

A.5 Exercise A

1. Find the order of the following functions:
2. a) $2x + 5x^3 - 3x^5$ b) $4x^2 - 17x^4 + 3x^8$ c) $\sin(x)$ d) x
3. Prove A.3.1) – A.3.5)

Appendix B: Boundary Conditions

B.1 Introduction

When solving a PDE using a finite difference (FD) scheme we may need to specify ghost grid points and associated ghost values for the dependent variable at these points.

e.g. in the FOU scheme to solve the 1D linear advection equation (Chapter 4) we need a ghost point to the *left* of the first grid point. This is illustrated in the following diagram:

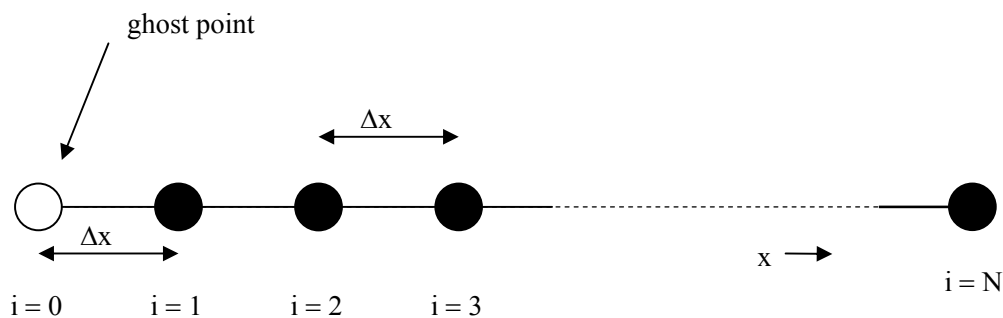


Figure B.1 Ghost (white) and grid (black) points for the FOU scheme

Notes:

1. In general for a 1D region on which grid points are indexed by $i = 1, 2, \dots, N$, we will index a left ghost point by $i = 0$ and a right ghost point by $i = N + 1$.
2. In many computer languages (e.g. Scilab and Matlab) array indices start at 1 so we must *shift indices* when using them to code up a FD scheme with a left ghost point. For example to code the FOU scheme we shift grid indices so that the left hand ghost point index is 1 and so indices 2 to $N+1$ represent the N computational grid points. If a scheme (e.g. Lax-Friedrichs) requires both left and right ghost points then the left ghost point has index 1 and the right ghost point has index $N+2$.

B.2 Boundary Conditions

In a computational region (which may be 1, 2 or 3D) ghost points and associated values occur at or adjacent to the *boundaries* of the region. Conditions leading to the prescription of ghost values are called *boundary conditions*. Boundary conditions are derived from the underlying physics of the situation. The correct treatment of boundary conditions is vital for accurate problem simulation. This is illustrated by considering the grid for a 2D region shown in Figure B.2. Grid points are indexed by $i = 1, 2, \dots, N$ in the x direction and by $j = 1, 2, \dots, M$ in the y direction. Indices 0, $N+1$ and $M+1$ indicate ghost points. To see the effect of different boundary conditions we consider the region to represent two different problems (modelled by the same PDEs whose solution is approximated by a FD scheme requiring the ghost points shown).

Please click the advert



May we offer you one
of the world's greatest
challenges?
In all humility.

We are looking for highly
educated engineers and
finance students.

Join us at dongenergy.com/job

dongenergy.com/job





B.2.1 Scenario 1

Consider the region in Figure B.2 to represent a channel of water viewed from above and flowing from left to right. The rows having indices $j = 1$ and $j = M$ correspond to the right and left sides of the channel respectively. If water cannot flow through or over the sides of the channel we must impose *solid boundary conditions* at each side by specifying the values of water depth and velocity at these locations in a special way. The columns with indices $i = 1$ and $i = N$ are at *inflow* and *outflow boundaries* respectively and require water depths and velocities to be specified according to flow type.

B.2.2 Scenario 2

Consider the region in Figure B.2 to be a near-shore area of ocean as viewed from above with waves travelling from left to right and impacting on a solid harbour wall that cannot be overtopped. The harbour wall, being solid, requires a solid boundary condition be imposed on the column with $i = N$. The column with $i = 1$ requires a *time dependent boundary condition* (for water depth and velocity) to generate the incoming waves. The rows with $j = 1$ and $j = M$ are edges of the finite computational domain and require *transmissive* boundary conditions to be imposed there so that waves can pass in and out of the computational domain.

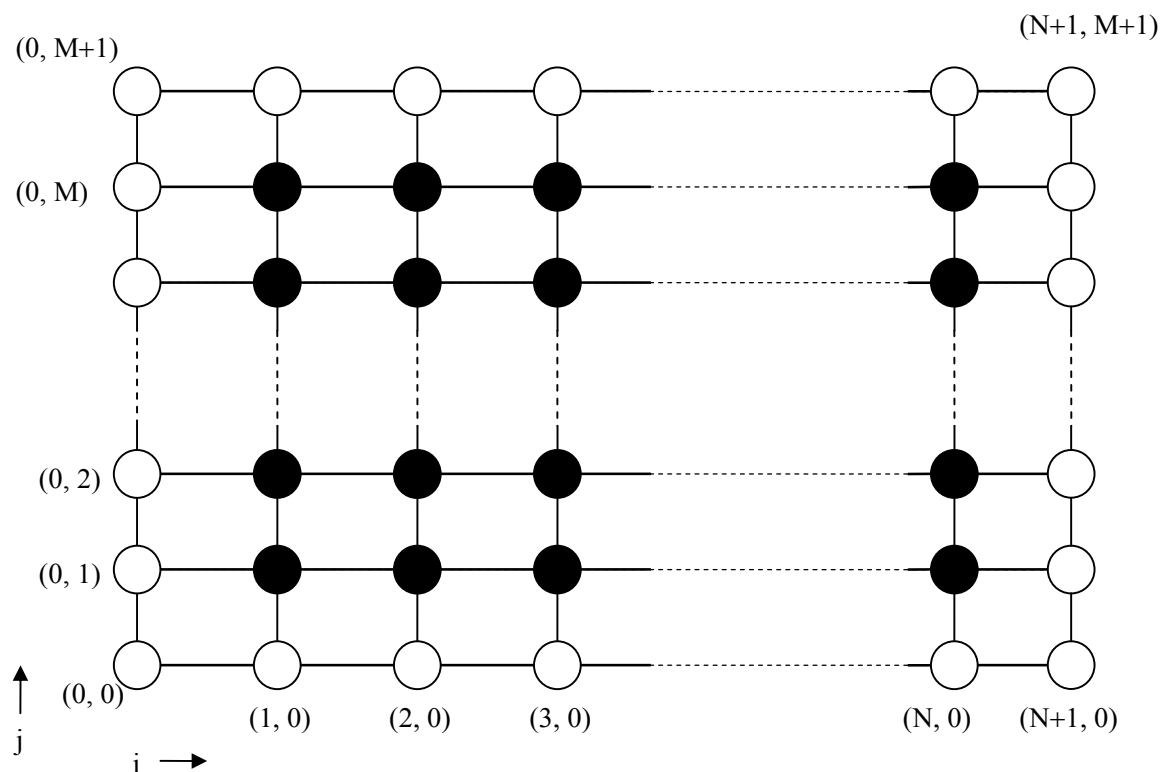


Figure B.2 Ghost (white) and grid (black) points for a 2D region

From the two previous scenarios it is clear that changing the boundary conditions changes the problem.

B.3 Specifying Ghost and Boundary Values

There are two basic ways to specify *values* of the dependent variable at ghost points. For clarity we will assume that the ghost point is at index $i = 0$ in a 1D domain and hence $i = 1$ is the index of the grid point on left hand boundary.

B.3.1 Dirichlet Boundary Conditions

In Dirichlet boundary conditions the *value*, u_0^n , of the dependent variable at a ghost point is *specified* in some way. Examples include:

- a) $u_0^n = \text{constant}$. e.g. $u_0^n = 0$ in the FOU scheme for the 1D linear advection equation (this condition indicates that there is no more pollutant entering the river, see Chapter 4).
- b) $u_0^n = f(n)$, which is a *time dependent* boundary condition (e.g. a tidal boundary as in the previous harbour example).
- c) $u_0^n = u_N^n$. This is a *periodic* boundary condition. This means that what passes out of the right boundary will pass in from the left boundary as though the two boundaries were joined together.

B.3.2 Derivative (von Neumann) Boundary Conditions

As the name suggests derivative boundary conditions specify the rate of change of the dependent variable *at the grid point adjacent to the ghost point* (i.e. at $i = 1$ in our case). This can be done in two ways:

B.3.2.1 $U_x = f(U)$.

Here the derivative of U in the x direction is specified at the boundary grid point $i = 1$. From this information u_0^n can be *calculated*. One way is to estimate U_x at $i = 1$ by a central difference giving,

$$U_x = f(U) \approx \frac{u_2^n - u_0^n}{2\Delta x} \quad (\text{B.1a})$$

which rearranges to give,

$$u_0^n = u_2^n - 2\Delta x f(U) \quad (\text{B.1b})$$

B.3.2.2 $U_{\underline{n}} = f(U)$.

Here the derivative of U in the direction of \underline{n} , the *outward* pointing normal to the boundary is given at the grid point adjacent to the ghost point. Note that this direction is opposite to the x direction at the left hand boundary ($i = 1$). From this information u_0^n can be *calculated*. As before we will use the left hand boundary and estimate $U_{\underline{n}}$ at $i = 1$ by a central difference giving,

$$U_{\underline{n}} \approx f(U) = \frac{u_0^n - u_2^n}{2\Delta x} \quad (\text{B.2a})$$

which rearranges to give,

$$u_0^n = u_2^n + 2\Delta x f(U) \quad (\text{B.2b})$$

Notes:

1. (B.1b) and (B.2b) are different because the directions of the given derivatives are opposite (the formulae would be the same at the right hand boundary). It is more usual to use (B.2b) than (B.1b).
2. In the derivative boundary condition examples central differences were used but other estimates could easily have been used (e.g. first order backward difference for the left hand ghost value).
3. The formal accuracy of a scheme may be reduced if ghost values are calculated on the basis of a method whose accuracy is less than the spatial accuracy of the scheme. e.g. a spatially third order scheme may drop to second order if a central difference (second order) is used to calculate a ghost value.
4. In many numerical simulations both Dirichlet and von Neumann boundary conditions are used. These are called mixed boundary conditions.

B.4 Common Boundary Conditions

We give some boundary conditions that are used frequently in problems.

B.4.1 Transmissive Boundary Conditions

Computational domains are finite and it may be that we need quantities simply to pass out of a boundary when they reach one that is not solid. This is often done by specifying a zero gradient normal to the boundary in the variable of interest on the boundary i.e. we use a derivative boundary condition in which the derivative is zero. As an example consider Figure B.2 with Scenario 2 in which water height h is a dependent variable and the solver requires a ghost value at $i = 3, j = 0$. Grid point $i = 3, j = 1$ is on a transmissive boundary so we want waves to pass through. To impose the transmissive boundary condition on water height, h , we set $h_y = 0$ at grid point $i = 3, j = 1$. Using a first order backward difference approximation,

$$h_x = 0 \approx \frac{h_{3,1}^n - h_{3,0}^n}{\Delta y} \quad (\text{B.3a})$$

$$\therefore h_{3,0}^n = h_{3,1}^n \quad (\text{B.3b})$$

Please click the advert



Ambitious and curious?

We are looking for young and highly educated engineers and finance students.

Join us at dongenergy.com/job

dongenergy.com/job





Download free ebooks at bookboon.com

B.4.2 Solid (reflective) Boundary Conditions

There is no flow through a solid boundary. This implies that the *normal* component of flow velocity at a solid boundary is zero. This condition is implemented by copying the normal component of velocity at the neighbouring interior grid point to the ghost cell and changing its sign. As an example consider Figure B.2 with Scenario 1. Grid point $i=3, j=1$ is on a solid boundary and we wish to find the velocity at the ghost point $i=3, j=0$. Suppose the velocity at neighbouring interior grid point $i=3, j=2$ has x and y components v_x and v_y respectively. The component of velocity normal to the solid boundary is v_y so the velocity at the ghost point $i=3, j=0$ has x and y components v_x and $-v_y$ respectively. Another way of obtaining this result is to linearly extrapolate using the normal velocities at $i=3, j=2$ (i.e. v_y) and $i=3, j=1$ (i.e. 0). Variables other than velocity at ghost points next to solid boundaries are usually found by a zero gradient approach.

B.4.3 Slip and No-Slip Boundary Conditions

At a solid boundary we have a choice of *tangential* velocity component. If there is appreciable friction at the boundary then it is said to be a no-slip boundary and the tangential velocity component is zero (along with the normal component). If friction is not present then the boundary is said to be a slip boundary and the tangential component of velocity may be *extrapolated* from interior tangential components.

B.5 Exercise B

1. A 1D PDE is to be solved numerically. The computational domain consists of N grid points numbered 1, 2, ..., N . In each case give the indices of the ghost grid points for the following FD schemes (which can be found in Chapter 4),
 - a. FTCS scheme, b) Crank-Nicolson, c) Lax-Wendroff, d) Lax-Friedrichs.
2. Run the FOU scheme for the 1D linear advection equation (Chapter 4) using a periodic boundary condition and describe how the solution evolves in time. (Remember indices start at 1 in Scilab or Matlab).
3. The computational domain $[0, 100]$ is discretised using 101 points. Initial values are given by $U(0, x) = \sin(x)$. The FOU scheme is used. Calculate the ghost point and the initial associated u value given the following boundary conditions,
 - a) Periodic, b) Dirichlet: $u_0^n = 0.1, u_{N+1}^n = -0.1$ c) Derivative: $U_x = 0$,
 - d) Derivative: $U_{\underline{n}} = 0.1$. e) Derivative: $U_x = U$.

4. Repeat Q3 for the Lax-Friedrichs scheme.
5. In Scenario 1 the flow velocity vector at grid point $i=2, j=M-1$ is $(v_x, v_y) = (4, 8)$.
 - a) Calculate the flow velocity vector at ghost point $i=2, j=M+1$.
 - b) Assuming a no-slip boundary find the flow velocity vector at grid point $i=2, j=M$.
 - c) Repeat b) assuming a slip boundary.
6. In Scenario 2 the water height at grid point $i=3, j=1$ is 8. Find the water height at ghost point $i=3, j=0$.

Please click the advert

The advertisement is enclosed in a rectangular frame. In the top left corner, there is a red rectangular logo with the text "AMBITIOUS PEOPLE" in white, bold, sans-serif font, and "MOVING ENERGY FORWARD" in a smaller, white, sans-serif font below it. In the top right corner, the URL "dongenergy.com/job" is written in a small, grey, sans-serif font. A large, light-grey speech bubble with a drop shadow is centered in the advertisement. Inside the speech bubble, the text "Everybody is talking..." is written in a large, black, sans-serif font. Below this, in a smaller, black, sans-serif font, is the text "...about future energy supply. We are not. Stop talking and make a career moving energy forward." At the bottom of the speech bubble, in an even smaller, black, sans-serif font, is the text "Ambitious engineers and finance students go to dongenergy.com/job". In the bottom right corner of the advertisement, the "DONG energy" logo is displayed, with "DONG" in a large, bold, red, sans-serif font and "energy" in a smaller, black, sans-serif font below it.

AMBITION
PEOPLE
MOVING ENERGY FORWARD

dongenergy.com/job

Everybody
is talking...

...about future energy supply. We are not.
Stop talking and make a career
moving energy forward.

Ambitious engineers and
finance students go to
dongenergy.com/job

DONG
energy

Appendix C: Consistency, Convergence and Stability

C.1 Introduction

Consistency, convergence and stability are important concepts but much of the theory has only been developed for special cases so a combination of theory and numerical experimentation is often the only way to proceed (e.g. to find the maximum stable time step). For clarity we restrict our attention to a single PDE where there are two independent variables, t and x . The dependent variable is $U = U(t, x)$. In operator notation the PDE is written,

$$L U = 0 \quad (\text{C.1})$$

L is a differential operator and U is the *analytical* (i.e. exact) solution of (C.1).

In order to approximate the solution of (C.1) the computational region is discretised into a finite set of grid points, x_i , where Δx is the (constant) grid spacing. The approximate solution is found at a set of time levels, t^n where Δt is the (variable) spacing between time levels. The analytical (i.e. exact) solution to (C.1) at (t^n, x_i) namely $U(t^n, x_i)$, is denoted by U_i^n . In operator notation the Finite Difference (FD) scheme to approximate the solution of (C.1) is written,

$$D_{\Delta t, \Delta x} u_i^n = 0 \quad (\text{C.2})$$

D is a difference operator and u_i^n is the solution to the FD scheme at (t^n, x_i) .

Notes:

1. The idea of the FD scheme (C.2) is that u_i^n approximates U_i^n and the approximation becomes better and better as Δx and Δt become smaller. Let,

$$e_i^n = u_i^n - U_i^n \quad (\text{C.3})$$

e_i^n is called the pointwise error (also called the ‘global error’).

2. Initially (i.e. time level 0) U is known at all grid points and u_i^0 is taken to be U_i^0 so $e_i^0 = 0$ at all grid points. As iterations of the FD scheme introduce errors, in general $e_i^n \neq 0$. It may be that as iterations continue errors are compounded and e_i^n grows unboundedly making the FD scheme useless.

The following concepts look at properties of the FD scheme with respect to errors.

C.2 Convergence

The FD scheme (C.2) for the PDE (C.1) is said to be *convergent* at (t, x) if the pointwise error at (t, x) tends to zero as Δx and Δt tend to zero, i.e.

$$e_i^n \rightarrow 0 \quad \text{as } (t^n, x_i) \rightarrow (t, x) \quad \text{when } \Delta t, \Delta x \rightarrow 0. \quad (\text{C.4})$$

To be useful our FD scheme must be convergent but this is very difficult to prove for many schemes. Convergence implies that a solution of the FD scheme approximates a solution of the PDE.

C.3 Consistency and Scheme Order

A measure of how well the exact solution of the PDE satisfies the FD scheme is given by the *truncation error*,

$$D_{\Delta t, \Delta x} U_i^n \quad (\text{C.5})$$

The FD scheme (C.2) is *consistent* with the PDE (C.1) if the truncation error tends to zero as Δx and Δt tend to zero, i.e.

$$D_{\Delta t, \Delta x} U_i^n \rightarrow 0 \quad \text{as } \Delta t, \Delta x \rightarrow 0 \quad \text{at } (t^n, x_i). \quad (\text{C.6})$$

Consistency is *necessary* for convergence.

The *order* of the truncation error is obtained by Taylor expansion of its terms about (t^n, x_i) . In many cases the order of the truncation error is the same as the order of the pointwise error so that the truncation error is a good (and accessible) guide to the accuracy of a FD scheme. We define the formal order of accuracy of a FD scheme by the order of its truncation error.

C.3.1 Example Calculation of Consistency and Scheme Order

The 1D linear advection equation is,

$$U_t + v U_x = 0 \quad (\text{C.7})$$

The FTCS scheme to solve (C.7) is given in Chapter 4. Writing this scheme in the form of (C.2) gives,

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + v \frac{u_{i+1}^n - u_{i-1}^n}{2 \Delta x} = 0 \quad (\text{C.8})$$

To determine scheme consistency and order we replace u in (C.8) by the exact solution, U , of (C.7) to give the truncation error,

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} + v \frac{U_{i+1}^n - U_{i-1}^n}{2 \Delta x} = 0 \quad (\text{C.9})$$

Terms in (C.9) are replaced by their Taylor expansion about the i^{th} spatial point and the n^{th} time level to give,

$$\begin{aligned} \frac{U_i^n + \Delta t U_t + O(\Delta t^2) - U_i^n}{\Delta t} + \frac{v}{2 \Delta x} \left(\left[U_i^n + \Delta x U_x + \frac{\Delta x^2}{2} U_{xx} + O(\Delta x^3) \right] \right. \\ \left. - \left[U_i^n - \Delta x U_x + \frac{\Delta x^2}{2} U_{xx} + O(\Delta x^3) \right] \right) \end{aligned} \quad (\text{C.10})$$

Simplifying (C.10) (using properties of O notation) gives,

$$U_t + v U_x + O(\Delta t) + O(\Delta x^2) \quad (\text{C.11})$$

Please click the advert



Dear highly educated engineering and finance students,

if you are driven, ambitious, open-minded and focused - we have a challenge for you. Actually, the greatest challenge in the world. Curious? Visit dongenergy.com/job

Best wishes

DONG Energy

dongenergy.com/job




and using (C.7) the truncation error is finally,

$$O(\Delta t) + O(\Delta x^2) \quad (\text{C.12})$$

Clearly the truncation error (C.12) tends to zero as $\Delta t, \Delta x \rightarrow 0$ which demonstrates that the FTCS scheme (C.8) is consistent with the original PDE (C.7). Furthermore the truncation error is first order in time and second order in space so the FTCS scheme is said to be (formally) first order in time and second order in space.

Notes:

1. In the Taylor expansions we expanded to second order for t and to third order for x . It doesn't matter if we expand too far as additive order expressions collapse to the term of lowest order.
2. Consistency does not imply that a FD scheme is any good! In fact the FTCS doesn't work for reasons we now explore.

C.4 Stability

We have seen from numerical results in Chapter 4 that care must be used when choosing the time step, Δt . A scheme that produces acceptable results for small Δt can give results that grow unboundedly as iterations continue if too large a Δt is chosen. In this case the scheme is said to have become *unstable*. A FD scheme is *stable* if and only if pointwise errors do not grow unboundedly with time which will be the case if, after some time level, N ,

$$|\underline{e}^{n+1}| \leq |\underline{e}^n|, \quad n \geq N. \quad (\text{C.13a})$$

Another way of defining stability comes from the following argument. Rewriting (C.3) gives,

$$u_i^n = U_i^n + e_i^n \quad (\text{C.13b})$$

For stability e_i^n must not grow unboundedly and since U_i^n is finite, (C.13b) implies that u_i^n must also not grow unboundedly which is true if,

$$|\underline{u}^{n+1}| \leq |\underline{u}^n|, \quad n \geq N. \quad (\text{C.13c})$$

The Lax Equivalence Theorem says that for a *linear* PDE, a consistent FD scheme to approximate its solution is convergent if and only if it is stable.

C.4.1 Heuristic Stability Analysis

The following reasoning gives a guide to choosing a stable time step and may be applied to PDEs where there is a speed of propagation. We apply our reasoning to the simple 1D linear advection equation of Chapter 4 (we will see in Chapter 7 that this reasoning extends to more complicated PDEs). Reasoning goes as follows: during time step, Δt , the concentration profile travels a distance $|v|\Delta t$ downstream. To capture the solution numerically it seems reasonable not to let the concentration profile move more than one grid interval, Δx , in a single time step, Δt . This means that, $|v|\Delta t \leq \Delta x$,

$$\therefore \Delta t \leq \frac{\Delta x}{|v|} \quad (\text{C.14a})$$

Inequality (C.14a) gives a heuristic guide to the maximum allowable time step (for the linear advection equation). It is often a good idea to multiply this maximum value by a ‘safety factor’ F where, $F < 1$. Numerical experiments can then be used to determine F . It is customary to let,

$$c = \frac{v\Delta t}{\Delta x} \quad (\text{C.14b})$$

c is called the Courant number. As we will see, schemes are often stable for c less than or equal to some number. (C.14a) implies that,

$$|c| \leq 1 \quad (\text{C.14c})$$

This is called the *CFL* condition (after *Courant, Friedrichs and Lewy*).

C.4.2 Matrix Stability Analysis

This is a more mathematical approach to stability analysis but it only applies to *linear* FD schemes. By writing out a linear FD scheme at each grid point and expressing the resulting set of linear equations as a single matrix equation, a linear FD scheme with 2 times levels can be written as,

$$A \underline{u}^{n+1} = B \underline{u}^n \quad (\text{C.15})$$

where, $\underline{u}^n = (u_1^n, u_2^n, \dots, u_N^n)^T$ and A and B are $N \times N$ matrices ($A = I$ for an explicit scheme). If the FD scheme is *consistent* it is satisfied by the exact solution of the PDE (neglecting the vanishing truncation error) and so,

$$A \underline{u}_i^{n+1} = B \underline{u}_i^n \quad (C.16)$$

Subtracting (C.16) from (C.15) and using (C.3) gives,

$$A \underline{e}^{n+1} = B \underline{e}^n \quad (C.17a)$$

$$\therefore \underline{e}^{n+1} = A^{-1} B \underline{e}^n \quad (C.17b)$$

$$\therefore |\underline{e}^{n+1}| = |A^{-1} B \underline{e}^n| \quad (C.17c)$$

$$\therefore |\underline{e}^{n+1}| \leq |A^{-1} B| |\underline{e}^n| \quad (C.17d)$$

Please click the advert

**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

May we offer you one
of the world's greatest
challenges?
In all humility.

We are looking for highly
educated engineers and
finance students.

Join us at dongenergy.com/job

dongenergy.com/job



DONG
energy

Hence by (C.13a) the FD scheme (C.15) is stable if,

$$\|A^{-1}B\| \leq 1 \quad (\text{C.18})$$

The matrix norm in (C.18) is induced by the vector norm in (C.13a). Hence the stability of a (linear) FD scheme can be investigated by finding the norm of the matrix $A^{-1}B$. This is the matrix method for stability and may be quite difficult to implement. It should be noted that there are many definitions of norms and a FD scheme may be stable in one norm but not in another.

C.4.2.1 Example of Matrix Stability Analysis

From Chapter 4 the FOU scheme to solve the 1D linear advection equation (C.7) can be written,

$$u_i^{n+1} = cu_{i-1}^n + (1-c)u_i^n \quad (\text{C.19})$$

Where c is defined in (C.14b). We use matrix stability analysis to find the maximum time step for scheme stability. Writing (C.19) out for each grid point, assuming $u_0^n = 0$ at all time levels and expressing the resulting linear equations in the form of (C.15) gives,

$$\underline{u}^{n+1} = \begin{pmatrix} (1-c) & 0 & 0 & \dots & 0 \\ c & (1-c) & 0 & \dots & 0 \\ 0 & c & (1-c) & 0 & \dots & 0 \\ \vdots & & & & & \\ 0 & & \dots & c & (1-c) \end{pmatrix} \underline{u}^n \quad (\text{C.20})$$

(C.19) is stable if the norm of the above matrix is less than or equal to 1. If we use the infinity norm for vectors (which is the maximum absolute value of components) the induced matrix norm is the maximum of the sum of all absolute values in each row. For our matrix this is $|c| + |1-c|$. For $0 \leq c \leq 1$ (which implies that $v > 0$), $|c| + |1-c| = c + (1-c) = 1$. Hence the FOU scheme is stable for $0 \leq c \leq 1$ and hence the maximum

allowable time step for stability is $\frac{\Delta x}{v}$ which agrees with the previous heuristically derived result. The

FOU scheme is said to be *conditionally* stable.

C.4.3 Von Neumann Stability Analysis

This analysis of stability is due to von Neumann and is based on (C.13c). We assume that the FD scheme is linear and that boundary conditions are *periodic*. Using the complex version of Fourier's Theorem and rescaling so that u has period 2π , we may write,

$$u_i^n = \sum_{k=-\infty}^{\infty} g_k^n e^{jkx_i} \quad (C.21)$$

where,

$j = \sqrt{-1}$, and $|g_k^n|$ is the amplitude of the k^{th} Fourier component.

By linearity it is enough to examine the behaviour of a single Fourier component so we replace u_i^n by $g_k^n e^{jkx_i}$ in (C.13c) and rearrange to get,

$$G = \left| \frac{g_k^{n+1}}{g_k^n} \right| \leq 1 \quad (C.22)$$

This is the von Neumann condition for stability. G is called the amplification factor. Von Neumann stability is carried out by the following steps:

Step 1. Replace each instance of u_i^n in the FD scheme by its corresponding single Fourier component.

Step 2. Rearrange to get G .

Step 3. Use the constraint (C.22) to obtain the condition for Δt (this step could be algebraically tricky). If (C.22) can never be satisfied for $\Delta t > 0$ the scheme is *unconditionally unstable*.

C.4.3.1 Von Neumann Stability Analysis: Example 1

The FTCS scheme for the 1D linear advection equation is,

$$u_i^{n+1} = u_i^n - \frac{v\Delta t}{2\Delta x} (u_{i+1}^n - u_{i-1}^n), \quad (C.23a)$$

which can be written compactly as,

$$u_i^{n+1} = u_i^n - \frac{c}{2} (u_{i+1}^n - u_{i-1}^n), \quad (C.23b)$$

where c is the usual Courant number defined by (C.14b).

Step 1: The FD scheme is linear and we assume periodic boundary conditions. Replacing each term in (C.23b) by its k^{th} Fourier component gives,

$$g_k^{n+1} e^{jkx_i} = g_k^n e^{jkx_i} - \frac{c}{2} (g_k^n e^{jkx_{i+1}} - g_k^n e^{jkx_{i-1}}) \quad (C.24)$$

Step 2: Noting that $x_{i+1} = x_i + \Delta x$, $x_{i-1} = x_i - \Delta x$, gives,

$$g_k^{n+1} e^{jkx_i} = g_k^n e^{jkx_i} - \frac{c}{2} (g_k^n e^{jk(x_i + \Delta x)} - g_k^n e^{jk(x_i - \Delta x)}) \quad (C.25)$$

Dividing through by $g_k^n e^{jkx_i}$ gives,

$$\frac{g_k^{n+1}}{g_k^n} = 1 - \frac{c}{2} (e^{jk\Delta x} - e^{-jk\Delta x}) \quad (C.26)$$

Please click the advert

dongenergy.com/job

**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

Ambitious and curious?

We are looking for young and highly educated engineers and finance students.

Join us at dongenergy.com/job

Step 3: From (C.22) for stability we must have,

$$G = \left| 1 - \frac{c}{2}(e^{jk\Delta x} - e^{-jk\Delta x}) \right| \leq 1 \quad (C.27)$$

Using the well known identity, $\sin(\theta) = \left(\frac{e^{j\theta} - e^{-j\theta}}{2j} \right)$

(C.27) gives,

$$|1 - jc\sin(k\Delta x)| \leq 1 \quad (C.28)$$

Squaring each side and evaluating the squared modulus gives,

$$1 + c^2 \sin^2(k\Delta x) \leq 1 \quad (C.29)$$

This inequality can only be satisfied (for all k) if $c = 0$ which implies that $\Delta t = 0$. i.e. there is no feasible value for the time step that makes the scheme stable. i.e. the FTCS scheme for the 1D linear advection equation is *unconditionally unstable*. The FTCS scheme is therefore useless even though we have shown that it is consistent!

C.4.3.2 Von Neumann Stability Analysis: Example 2

We apply von Neumann stability analysis to the FOU scheme (C.19).

Step 1: Replacing each term in (C.19) by its k^{th} Fourier component gives,

$$g_k^{n+1} e^{jkx_i} = (1-c)g_k^n e^{jkx_i} + cg_k^n e^{jkx_{i-1}} \quad (C.30)$$

Noting that $x_{i-1} = x_i - \Delta x$, gives,

$$g_k^{n+1} e^{jkx_i} = (1-c)g_k^n e^{jkx_i} + cg_k^n e^{jk(x_i - \Delta x)} \quad (C.31)$$

Step 2: Dividing through by $g_k^n e^{jkx_i}$ gives,

$$\frac{g_k^{n+1}}{g_k^n} = (1-c) + ce^{-jk\Delta x} \quad (C.32)$$

Step 3: For stability we must have,

$$G = |(1-c) + ce^{-jk\Delta x}| \leq 1 \quad (C.33)$$

The well-known triangle inequality states that, $|a+b| \leq |a| + |b|$, hence,

$$|(1-c) + ce^{-jk\Delta x}| \leq |(1-c)| + |ce^{-jk\Delta x}| = |1-c| + |c| \quad (C.34)$$

When $0 \leq c \leq 1$, $|1-c| = 1-c$ and $|c| = c$, therefore (C.34) gives,

$$|(1-c) + ce^{-jk\Delta x}| \leq |1-c| + |c| = (1-c) + c = 1 \quad (C.35)$$

Hence the FOU scheme for the 1D linear advection equation is stable when $0 \leq c \leq 1$ which means that $\Delta t \leq \frac{\Delta x}{v}$. The FOU scheme is said to be conditionally stable.

Notes:


1. Stability analysis hasn't been worked out for most non-linear schemes (PDEs).
2. Strictly speaking, the von Neumann stability analysis requires periodic boundary conditions but seems to work even when this is not the case.

C.5 Exercise C

1. Show that the FOU scheme for the 1D linear advection equation is consistent and find its formal order.
2. Repeat Q1 for a) Lax-Friedrichs b) Lax-Wendroff c) Crank-Nicolson

3. Show that the 5-point scheme of Chapter 3 for the 2D Laplace's Equation is consistent and find its formal order.
4. Using heuristic analysis, estimate the maximum allowable time step for an explicit scheme to solve the 1D linear advection equation for pollution in a river 0.5 km long flowing at 4m/s using 200 grid points.
5. Use von Neumann stability analysis to show that the FD scheme to solve the linear advection equation for $v > 0$ using first order forward differences in both space and time is unconditionally unstable.
6. Use von Neumann stability analysis to show that the Crank-Nicolson scheme to solve the 1D linear advection equation is unconditionally stable.
7. Use von Neumann stability analysis to investigate the stability of the Lax-Friedrichs scheme to solve the 1D linear advection equation.
8. Repeat Q7 for the Leap-Frog scheme.
9. Repeat Q5-Q7 using matrix stability analysis.

Please click the advert



AMBITION
PEOPLE
MOVING ENERGY FORWARD

dongenergy.com/job

Everybody
is talking...

...about future energy supply. We are not.
Stop talking and make a career
moving energy forward.

Ambitious engineers and
finance students go to
dongenergy.com/job

DONG
energy

Appendix D: Convergence Analysis for Iterative Methods

D.1 Introduction

Iterative schemes for matrix inversion do not necessarily converge so we need to determine conditions for convergence. We would also like to know how fast they converge. In the following we assume that the reader is familiar with some standard results from Linear Algebra. We are trying to solve the system of linear equations (refer back to Equations (3.6), (3.9)),

$$A \underline{u} = \underline{b} \quad (D.1)$$

where A is an $N \times N$ matrix, \underline{u} is a column vector of N unknowns and \underline{b} is a column vector of N known constants. It is always possible to scale each equation so that every entry on the main diagonal of A is 1. A can then be written as,

$$A = -L + I - U \quad (D.2)$$

where I is the $N \times N$ identity matrix and L and U are $N \times N$ lower and upper triangular matrices respectively. Substituting (D.2) into (D.1) and rearranging gives,

$$\underline{u} = (L + U) \underline{u} + \underline{b} \quad (D.3)$$

Equation (D.3) is the basis for the following analysis.

Notes:

1. Research into computationally efficient ways to invert a matrix continues. Computational efficiency is not simply a matter of reducing the number of calculations for a particular class of problem; it also depends on computer architecture. It may be that some less sophisticated method is faster than a more modern method when run on a parallel computer.
2. There are many efficient freely downloadable matrix inversion programs so it is almost never worth writing your own.
3. There is probably no best solution to computational efficiency. Where speed is an issue it will pay to experiment with several methods and tune them (if necessary) by numerical experiments on small problems.

D.2 Jacobi Iteration

Equation (D.3) suggests the iterative scheme,

$$\underline{u}^{m+1} = (L + U) \underline{u}^m + \underline{b} \quad (\text{D.4})$$

which is the Jacobi iteration scheme in matrix form. $(L + U)$ is called the *iteration matrix*. In order to analyse convergence of the Jacobi iteration scheme we need to look at how the error behaves between iterations. Clearly we want the error to decrease to zero as iterations continue. The exact solution to (D.1) is \underline{u} . Let the error after the m^{th} iteration be \underline{e}^m , so,

$$\underline{e}^m = \underline{u}^m - \underline{u} \quad (\text{D.5})$$

Note that \underline{e}^m is a *vector* in \mathbb{R}^N whose components are the errors at each grid point after the m^{th} iteration. Subtracting (D.3) from (D.4) gives,

$$\underline{u}^{m+1} - \underline{u} = (L + U) \underline{u}^m + \underline{b} - (L + U) \underline{u} - \underline{b} \quad (\text{D.6})$$

$$\therefore \underline{e}^{m+1} = (L + U) \underline{e}^m \quad (\text{D.7})$$

writing \underline{e}^0 for the initial error in the initial (guessed) values for \underline{u}^0 , after 1 iteration (D.7) gives,

$$\underline{e}^1 = (L + U) \underline{e}^0.$$

A second iteration gives,

$$\begin{aligned} \underline{e}^2 &= (L + U) \underline{e}^1 \\ &= (L + U)^2 \underline{e}^0. \end{aligned}$$

So after n iterations we have

$$\underline{e}^n = (L + U)^n \underline{e}^0 \quad (\text{D.8})$$

For convergence of the iterative scheme all components of \underline{e}^n must approach zero in the limit,

$$\text{i.e. } \lim_{n \rightarrow \infty} \underline{e}^n = \underline{0}$$

$$\therefore \lim_{n \rightarrow \infty} (L + U)^n \underline{e}^0 = \underline{0}$$

Since \underline{e}^0 is a non-zero constant vector we must focus our attention on the behaviour of $(L + U)^n$. We assume that $(L + U)$ has n linearly independent eigenvectors, $\underline{v}_1, \underline{v}_2, \dots, \underline{v}_N$ with corresponding eigenvalues, $\lambda_1, \lambda_2, \dots, \lambda_N$. By a standard result from Linear Algebra $(L + U)^n$ has the same eigenvectors with corresponding eigenvalues, $\lambda_1^n, \lambda_2^n, \dots, \lambda_N^n$. The set of eigenvectors form a basis for \mathbb{R}^N so for some constants a_i we may write,

$$\underline{e}^0 = a_1 \underline{v}_1 + a_2 \underline{v}_2 + \dots + a_N \underline{v}_N.$$

$$\therefore (L + U)^n \underline{e}^0 = (L + U)^n (a_1 \underline{v}_1 + a_2 \underline{v}_2 + \dots + a_N \underline{v}_N)$$

$$= (L + U)^n a_1 \underline{v}_1 + (L + U)^n a_2 \underline{v}_2 + \dots + (L + U)^n a_N \underline{v}_N$$

$$= a_1 (L + U)^n \underline{v}_1 + a_2 (L + U)^n \underline{v}_2 + \dots + a_N (L + U)^n \underline{v}_N$$

$$= a_1 \lambda_1^n \underline{v}_1 + a_2 \lambda_2^n \underline{v}_2 + \dots + a_N \lambda_N^n \underline{v}_N$$

which will clearly tend to the zero vector if and only if,

$$|\lambda_i| < 1, \text{ for } i = 1, 2, \dots, N.$$

Definition: The *dominant eigenvalue* of a matrix is the eigenvalue with the largest modulus.

Hence we can say that the Jacobi iterative scheme converges if and only if the dominant eigenvalue of its iteration matrix has modulus less than 1. We denote the dominant eigenvalue of the Jacobi iteration matrix by λ .

D.3 Gauss-Seidel Iteration

Using the previous notation it can be shown that the Gauss-Seidel iterative method can be expressed as,

$$\underline{u}^{m+1} = U \underline{u}^m + L \underline{u}^{m+1} + \underline{b} \quad (\text{D.9})$$

and a similar analysis to the Jacobi iterative scheme shows that,

$$\underline{e}^{m+1} = U \underline{e}^m + L \underline{e}^{m+1} \quad (\text{D.10a})$$

$$\therefore \underline{e}^{m+1} = (I - L)^{-1} U \underline{e}^m \quad (\text{D.10b})$$

$$\therefore \underline{e}^{m+1} = ((I - L)^{-1} U)^m \underline{e}^0 \quad (\text{D.10c})$$

$(I - L)^{-1} U$ is called the Gauss-Seidel iteration matrix. By an exactly similar analysis to that for Jacobi iteration, the Gauss-Seidel scheme converges if and only if the dominant eigenvalue of its iteration matrix has modulus less than 1. It can be shown that the dominant eigenvalue $= \lambda^2$.

Please click the advert

dongenergy.com/job

**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

Dear highly educated engineering and finance students,

if you are driven, ambitious, open-minded and focused - we have a challenge for you. Actually, the greatest challenge in the world. Curious? Visit dongenergy.com/job

Best wishes
DONG Energy

DONG
 energy

D.4 SoR Iterative Scheme

A similar analysis to the above shows that the SoR iteration matrix is,

$$(I - w U)^{-1} ((1 - w) I + w L) \quad (D.11a)$$

It can be shown that its dominant eigenvalue, is,

$$\frac{2}{1 + \sqrt{1 - \lambda^2}} - 1 \quad (D.11b)$$

As before the scheme converges if and only if this has modulus less than 1.

D.4.1 A Special Case for SoR

For a rectangular $p\Delta x$ by $q\Delta y$ computational region the optimal value of the SoR relaxation parameter can be shown to be,

$$w_o = \frac{2}{1 + \sqrt{1 - \mu}} \quad (D.12a)$$

where the dominant eigenvalue, μ , of the corresponding Gauss-Seidel scheme is,

$$\mu = \frac{\left(\cos\left(\frac{\pi}{p}\right) + \cos\left(\frac{\pi}{q}\right) \right)^2}{4} \quad (D.12b)$$

D.5 Theory for Dominant Eigenvalues

Convergence of iterative schemes depends on the dominant eigenvalue of the associated iteration matrix having a modulus less than 1. In general it is difficult and/or computationally expensive to find eigenvalues. The following theorem is a quick way to find an *upper bound* for the modulus of the dominant eigenvalue of a matrix.

D.5.1 Gershgorin's Theorem

The modulus of the dominant eigenvalue of a matrix is less than or equal to the sum of the moduli of the entries in any row or column.

e.g. Let $A = \begin{pmatrix} 0.3 & -0.1 & 0.1 \\ 0.2 & 0.2 & 0.3 \\ 0.4 & 0.1 & -0.3 \end{pmatrix}$

The sums of the moduli are:

Row 1: $|0.3|+|-0.1|+|0.1| = 0.5$, Row 2: $|0.2|+|0.2|+|0.3| = 0.7$

Row 3: $|0.4|+|0.1|+|-0.3| = 0.8$, Col 1: $|0.3|+|0.2|+|0.4| = 0.9$

Col 2: $|-0.1|+|0.2|+|0.1| = 0.4$, Col 3: $|0.1|+|0.3|+|-0.3| = 0.7$

Hence the dominant eigenvalue of A is less than or equal to 0.9. If A were an iteration matrix then the iteration scheme would converge. If the maximum value of the sum of the moduli of the row or column elements is greater than 1 then the theorem is no use for determining whether the scheme converges. In this case we need a way of estimating the dominant eigenvalue.

D.5.2 Power Method for Estimating Dominant Eigenvalues

This is an efficient way of estimating the dominant eigenvalue (and associated eigenvector) of a matrix A. Start with an arbitrary non-zero vector \underline{v}^0 and define the iterative scheme,

$$\underline{v}^{i+1} = A \underline{v}^i \quad (\text{D.13})$$

It can be shown that as the iteration index i tends to infinity, \underline{v}^{i+1} tends to $\lambda \underline{v}^i$ where λ is the dominant eigenvalue of A with associated eigenvector \underline{v}^i and where after each iteration the resulting vector, \underline{v}^{i+1} , is scaled by a constant k_i (which is the reciprocal of its first entry) so that its first component becomes 1. The distance between the scaled versions of \underline{v}^{i+1} and \underline{v}^i is found and the iteration stops when this distance is less than some predefined tolerance. The resulting estimate for the dominant eigenvalue of A is k_i .

D.5.2.1 Example Power Method Calculation

e.g. $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$. We use the power method with a tolerance of $\text{tol} = 0.01$ to find the dominant eigenvalue of A.

Let, $\underline{v}^0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. Using the iteration scheme (D.13),

$$\begin{aligned}\underline{v}^1 &= \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \\ &= 2 \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}.\end{aligned}$$

so $k_1 = 2$ and scaled $\underline{v}^1 = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}$.

$|\underline{v}^1 - \underline{v}^0|_\infty = 0.5 > \text{tol}$, so the iterations continue,

$$\begin{aligned}\underline{v}^2 &= \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 2.5 \\ 2 \end{pmatrix} \\ &= 2.5 \begin{pmatrix} 1 \\ 0.8 \end{pmatrix},\end{aligned}$$

so $k_2 = 2.5$ and scaled $\underline{v}^2 = \begin{pmatrix} 1 \\ 0.8 \end{pmatrix}$

Comparing the scaled vectors gives,

$|\underline{v}^2 - \underline{v}^1|_\infty = 0.3 > \text{tol}$ so the iterations continue and after 6 iterations we have,

$$\underline{v}^6 = 2.9918 \begin{pmatrix} 1 \\ 0.9973 \end{pmatrix},$$

so $k_6 = 2.9918$ and scaled $\underline{v}^6 = \begin{pmatrix} 1 \\ 0.9973 \end{pmatrix}$

Comparing scaled vectors gives, $|\underline{v}^6 - \underline{v}^5|_\infty = 0.0055 < \text{tol}$, so the iterations stop. An estimate for the dominant eigenvalue of A is $k_6 = 2.9918$ (with \underline{v}^6 being the corresponding estimated eigenvector). The exact answer is 3. Computer code for this method can be downloaded from the website.

D.6 Rates of Convergence of Iterative Schemes

The Rate of Convergence (RoC) of an iterative scheme is a measure of the number of iterations needed to converge to some given tolerance. It turns out that the RoC of an iterative scheme can be defined as,

$$-\log_e \lambda \quad (\text{D.14})$$

where λ is the dominant eigenvalue of its iteration matrix.

The relative RoC of two schemes with dominant eigenvalues λ_1 and λ_2 is,

$$\frac{-\log_e \lambda_1}{-\log_e \lambda_2} \quad (\text{D.15})$$

Please click the advert



May we offer you one
of the world's greatest
challenges?
In all humility.

We are looking for highly
educated engineers and
finance students.

Join us at dongenergy.com/job

dongenergy.com/job





We can now compare (convergent) iterative schemes.

e.g. Given that the point-Jacobi scheme is convergent with dominant eigenvalue λ we know that the point-Gauss-Seidel scheme has dominant eigenvalue λ^2 and so is also convergent. By (D.15) the relative RoC of the Gauss-Seidel scheme to the Jacobi scheme is,

$$\frac{-\log_e \lambda^2}{-\log_e \lambda} = 2$$

i.e. the number of iterations to achieve the same level of accuracy using the Gauss-Seidel scheme is approximately half that of the Jacobi scheme.

The relative RoC isn't the whole story when comparing iterative schemes. It could be that an iterative scheme needs many iterations to converge but that each iteration is computationally fast. This could make it a faster than a quick converging but computationally slow scheme.

D.7 Exercise D

1. 1. $A = \begin{pmatrix} -0.2 & -0.3 & -0.1 \\ -0.3 & 0.1 & 0.5 \\ -0.1 & 0.5 & 0.5 \end{pmatrix}$. Use Gershgorin's theorem to provide an upper bound for the dominant eigenvalue of A. If A was an iteration matrix would the iteration converge?
2. Find the exact dominant eigenvalue for $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ and compare with the results from the power method code which you should run with a tolerance of 0.001.
3. Adapt the power method code to estimate the dominant eigenvalue for the matrix in Q1. Check your answer by using Scilab's (or Matlab's) built-in function for finding eigenvalues.
4. Given that the Jacobi iterative scheme converges show that the Gauss-Seidel scheme also converges. Show that the SoR scheme also converges.
5. Find the RoC for an iterative scheme with dominant eigenvalue 0.3.
6. Repeat Q5 for a dominant eigenvalue of 0.6.

7. Given that a Gauss-Seidel iterative scheme has dominant eigenvalue 0.5 find the relative RoC of SoR to Gauss-Seidel.
8. Iterative scheme 1 has dominant eigenvalue 0.3 and iterative scheme 2 has dominant eigenvalue 0.6. Which scheme converges fastest? If scheme 1 takes 20 iterations to converge approximately in how many iterations does scheme 2 to converge (with the same tolerance)?

Please click the advert



**AMBITIOUS
PEOPLE**
MOVING ENERGY FORWARD

dongenergy.com/job

Ambitious and curious?

We are looking for young and highly educated engineers and finance students.

Join us at dongenergy.com/job





DONG
energy