

NUMERICAL METHODS IN COMPUTATIONAL FINANCE

A Partial Differential Equation
(PDE/FDM) Approach

DANIEL J. DUFFY



WILEY

Numerical Methods in Computational Finance

Founded in 1807, John Wiley & Sons is the oldest independent publishing company in the United States. With offices in North America, Europe, Australia and Asia, Wiley is globally committed to developing and marketing print and electronic products and services for our customers' professional and personal knowledge and understanding.

The Wiley Finance series contains books written specifically for finance and investment professionals as well as sophisticated individual investors and their financial advisors. Book topics range from portfolio management to e-commerce, risk management, financial engineering, valuation and financial instrument analysis, as well as much more.

For a list of available titles, visit our Web site at www.WileyFinance.com.

Numerical Methods in Computational Finance

**A Partial Differential Equation
(PDE/FDM) Approach**

DANIEL J. DUFFY

WILEY

This edition first published 2022
Copyright © 2022 by John Wiley & Sons, Ltd.

Registered office

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by the UK Copyright, Designs and Patents Act 1988, without the prior permission of the publisher. Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The publisher is not associated with any product or vendor mentioned in this book.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. It is sold on the understanding that the publisher is not engaged in rendering professional services and neither the publisher nor the author shall be liable for damages arising here from. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Library of Congress Cataloging-in-Publication Data is Available:

ISBN 9781119719670 (Hardback)
ISBN 9781119719724 (ePub)
ISBN 9781119719694 (ePDF)

Cover Design: Wiley

Cover Image: © windesign/Shutterstock

Set in 10/12pt STIXTwoText by Straive, Chennai, India

Contents

Preface	xix	
Who Should Read this Book?	xxiii	
<hr/>		
PART A		
Mathematical Foundation for One-Factor Problems		
<hr/>		
CHAPTER 1		
Real Analysis Foundations for this Book	3	
1.1	Introduction and Objectives	3
1.2	Continuous Functions	4
1.2.1	<i>Formal Definition of Continuity</i>	5
1.2.2	<i>An Example</i>	6
1.2.3	<i>Uniform Continuity</i>	6
1.2.4	<i>Classes of Discontinuous Functions</i>	7
1.3	Differential Calculus	8
1.3.1	<i>Taylor's Theorem</i>	9
1.3.2	<i>Big O and Little o Notation</i>	10
1.4	Partial Derivatives	11
1.5	Functions and Implicit Forms	13
1.6	Metric Spaces and Cauchy Sequences	14
1.6.1	<i>Metric Spaces</i>	15
1.6.2	<i>Cauchy Sequences</i>	16
1.6.3	<i>Lipschitz Continuous Functions</i>	17
1.7	Summary and Conclusions	19
<hr/>		
CHAPTER 2		
Ordinary Differential Equations (ODEs), Part 1	21	
2.1	Introduction and Objectives	21
2.2	Background and Problem Statement	22
2.2.1	<i>Qualitative Properties of the Solution and Maximum Principle</i>	22
2.2.2	<i>Rationale and Generalisations</i>	24

2.3	Discretisation of Initial Value Problems: Fundamentals	25
2.3.1	<i>Common Schemes</i>	26
2.3.2	<i>Discrete Maximum Principle</i>	28
2.4	Special Schemes	29
2.4.1	<i>Exponential Fitting</i>	29
2.4.2	<i>Scalar Non-Linear Problems and Predictor-Corrector Method</i>	31
2.4.3	<i>Extrapolation</i>	31
2.5	Foundations of Discrete Time Approximations	32
2.6	Stiff ODEs	37
2.7	Intermezzo: Explicit Solutions	39
2.8	Summary and Conclusions	41
CHAPTER 3		
Ordinary Differential Equations (ODEs), Part 2		43
3.1	Introduction and Objectives	43
3.2	Existence and Uniqueness Results	43
3.2.1	<i>An Example</i>	45
3.3	Other Model Examples	45
3.3.1	<i>Bernoulli ODE</i>	45
3.3.2	<i>Riccati ODE</i>	46
3.3.3	<i>Predator-Prey Models</i>	47
3.3.4	<i>Logistic Function</i>	48
3.4	Existence Theorems for Stochastic Differential Equations (SDEs)	48
3.4.1	<i>Stochastic Differential Equations (SDEs)</i>	49
3.5	Numerical Methods for ODEs	51
3.5.1	<i>Code Samples in Python</i>	52
3.6	The Riccati Equation	55
3.6.1	<i>Finite Difference Schemes</i>	57
3.7	Matrix Differential Equations	59
3.7.1	<i>Transition Rate Matrices and Continuous Time Markov Chains</i>	61
3.8	Summary and Conclusions	62
CHAPTER 4		
An Introduction to Finite Dimensional Vector Spaces		63
4.1	Short Introduction and Objectives	63
4.1.1	<i>Notation</i>	64
4.2	What Is a Vector Space?	65
4.3	Subspaces	67
4.4	Linear Independence and Bases	68

4.5	Linear Transformations	69
4.5.1	<i>Invariant Subspaces</i>	70
4.5.2	<i>Rank and Nullity</i>	71
4.6	Summary and Conclusions	72

CHAPTER 5**Guide to Matrix Theory and Numerical Linear Algebra****73**

5.1	Introduction and Objectives	73
5.2	From Vector Spaces to Matrices	73
5.2.1	<i>Sums and Scalar Products of Linear Transformations</i>	73
5.3	Inner Product Spaces	74
5.3.1	<i>Orthonormal Basis</i>	75
5.4	From Vector Spaces to Matrices	76
5.4.1	<i>Some Examples</i>	76
5.5	Fundamental Matrix Properties	77
5.6	Essential Matrix Types	80
5.6.1	<i>Nilpotent and Related Matrices</i>	80
5.6.2	<i>Normal Matrices</i>	81
5.6.3	<i>Unitary and Orthogonal Matrices</i>	82
5.6.4	<i>Positive Definite Matrices</i>	82
5.6.5	<i>Non-Negative Matrices</i>	83
5.6.6	<i>Irreducible Matrices</i>	83
5.6.7	<i>Other Kinds of Matrices</i>	84
5.7	The Cayley Transform	84
5.8	Summary and Conclusions	86

CHAPTER 6**Numerical Solutions of Boundary Value Problems****87**

6.1	Introduction and Objectives	87
6.2	An Introduction to Numerical Linear Algebra	87
6.2.1	<i>BLAS (Basic Linear Algebra Subprograms)</i>	90
6.3	Direct Methods for Linear Systems	92
6.3.1	<i>LU Decomposition</i>	92
6.3.2	<i>Cholesky Decomposition</i>	94
6.4	Solving Tridiagonal Systems	94
6.4.1	<i>Double Sweep Method</i>	94
6.4.2	<i>Thomas Algorithm</i>	96
6.4.3	<i>Block Tridiagonal Systems</i>	97
6.5	Two-Point Boundary Value Problems	99
6.5.1	<i>Finite Difference Approximation</i>	100
6.5.2	<i>Approximation of Boundary Conditions</i>	102
6.6	Iterative Matrix Solvers	103
6.6.1	<i>Iterative Methods</i>	103
6.6.2	<i>Jacobi Method</i>	104

6.6.3	<i>Gauss–Seidel Method</i>	104
6.6.4	<i>Successive Over-Relaxation (SOR)</i>	105
6.6.5	<i>Other Methods</i>	105
6.7	Example: Iterative Solvers for Elliptic PDEs	106
6.8	Summary and Conclusions	107

CHAPTER 7**Black–Scholes Finite Differences for the Impatient****109**

7.1	Introduction and Objectives	109
7.2	The Black–Scholes Equation: Fully Implicit and Crank–Nicolson Methods	110
7.2.1	<i>Fully Implicit Method</i>	110
7.2.2	<i>Crank–Nicolson Method</i>	111
7.2.3	<i>Final Remarks</i>	114
7.3	The Black–Scholes Equation: Trinomial Method	115
7.3.1	<i>Comparison with Other Methods</i>	115
7.4	The Heat Equation and Alternating Direction Explicit (ADE) Method	120
7.4.1	<i>Background and Motivation</i>	120
7.5	ADE for Black–Scholes: Some Test Results	121
7.6	Summary and Conclusions	126

PART B**Mathematical Foundation for Two-Factor Problems****CHAPTER 8****Classifying and Transforming Partial Differential Equations****129**

8.1	Introduction and Objectives	129
8.2	Background and Problem Statement	129
8.3	Introduction to Elliptic Equations	130
8.3.1	<i>What is an Elliptic Operator?</i>	130
8.3.2	<i>Total and Principal Symbols</i>	131
8.3.3	<i>The Adjoint Equation</i>	132
8.3.4	<i>Self-Adjoint Operators and Equations</i>	133
8.3.5	<i>Numerical Approximation of PDEs in Adjoint Form</i>	134
8.3.6	<i>Elliptic Equations with Non-Negative Characteristic Form</i>	135
8.4	Classification of Second-Order Equations	135
8.4.1	<i>Characteristics</i>	136
8.4.2	<i>Model Example</i>	137
8.4.3	<i>Test your Knowledge</i>	138
8.5	Examples of Two-Factor Models from Computational Finance	139
8.5.1	<i>Multi-Asset Options</i>	139
8.5.2	<i>Stochastic Dividend PDE</i>	140
8.6	Summary and Conclusions	141

CHAPTER 9		
Transforming Partial Differential Equations to a Bounded Domain		143
9.1 Introduction and Objectives		143
9.2 The Domain in Which a PDE Is Defined: Preamble		143
9.2.1 <i>Background and Specific Mappings</i>		144
9.2.2 <i>Initial Examples</i>		146
9.3 Other Examples		147
9.4 Hotspots		148
9.5 What Happened to Domain Truncation?		148
9.6 Another Way to Remove Mixed Derivative Terms		149
9.7 Summary and Conclusions		151
CHAPTER 10		
Boundary Value Problems for Elliptic and Parabolic Partial Differential Equations		153
10.1 Introduction and Objectives		153
10.2 Notation and Prerequisites		154
10.3 The Laplace Equation		154
10.3.1 <i>Harmonic Functions and the Cauchy–Riemann Equations</i>		154
10.4 Properties of The Laplace Equation		156
10.4.1 <i>Maximum-Minimum Principle for Laplace's Equation</i>		158
10.5 Some Elliptic Boundary Value Problems		159
10.5.1 <i>Some Motivating Examples</i>		159
10.6 Extended Maximum-Minimum Principles		159
10.6.1 <i>An Example</i>		161
10.7 Summary and Conclusions		162
CHAPTER 11		
Fichera Theory, Energy Inequalities and Integral Relations		163
11.1 Introduction and Objectives		163
11.2 Background and Problem Statement		163
11.2.1 <i>The 'Big Bang': Cauchy–Euler Equation</i>		163
11.3 Well-Posed Problems and Energy Estimates		165
11.3.1 <i>Time to Reflect: What Have We Achieved and What's Next?</i>		167
11.4 The Fichera Theory: Overview		168
11.5 The Fichera Theory: The Core Business		168
11.6 The Fichera Theory: Further Examples and Applications		171
11.6.1 <i>Cox–Ingersoll–Ross (CIR)</i>		171
11.6.2 <i>Heston Model Fundamentals</i>		172
11.6.3 <i>Heston Model by Fichera Theory</i>		176
11.6.4 <i>First-Order Hyperbolic PDE in One and Two Space Variables</i>		177

11.7 Some Useful Theorems	178
11.7.1 <i>Divergence (Gauss–Ostrogradsky) Theorem</i>	179
11.7.2 <i>Green’s Theorem/Formula</i>	180
11.7.3 <i>Green’s First and Second Identities</i>	180
11.8 Summary and Conclusions	180
CHAPTER 12	
An Introduction to Time-Dependent Partial Differential Equations	181
12.1 Introduction and Objectives	181
12.2 Notation and Prerequisites	181
12.3 Preamble: Separation of Variables for the Heat Equation	182
12.4 Well-Posed Problems	184
12.4.1 <i>Examples of an ill-posed Problem</i>	185
12.4.2 <i>The Importance of Proving that Problems Are Well-Posed</i>	187
12.5 Variations on Initial Boundary Value Problem for the Heat Equation	188
12.5.1 <i>Smoothness and Compatibility Conditions</i>	188
12.6 Maximum-Minimum Principles for Parabolic PDEs	189
12.7 Parabolic Equations with Time-Dependent Boundaries	190
12.8 Uniqueness Theorems for Boundary Value Problems in Two Dimensions	192
12.8.1 <i>Laplace Equation</i>	192
12.8.2 <i>Heat Equation</i>	193
12.9 Summary and Conclusions	193
CHAPTER 13	
Stochastics Representations of PDEs and Applications	195
13.1 Introduction and Objectives	195
13.2 Background, Requirements and Problem Statement	196
13.3 An Overview of Stochastic Differential Equations (SDEs)	196
13.4 An Introduction to One-Dimensional Random Processes	196
13.5 An Introduction to the Numerical Approximation of SDEs	199
13.5.1 <i>Euler–Maruyama Method</i>	199
13.5.2 <i>Milstein Method</i>	201
13.5.3 <i>Predictor-Corrector Method</i>	201
13.5.4 <i>Drift-Adjusted Predictor-Corrector Method</i>	202
13.6 Path Evolution and Monte Carlo Option Pricing	203
13.6.1 <i>Monte Carlo Option Pricing</i>	204
13.6.2 <i>Some C++ Code</i>	205
13.7 Two-Factor Problems	209
13.7.1 <i>Spread Options with Stochastic Volatility</i>	209
13.7.2 <i>Heston Stochastic Volatility Model</i>	211
13.8 The Ito Formula	215
13.9 Stochastics Meets PDEs	215
13.9.1 <i>A Statistics Refresher</i>	215

13.9.2 <i>The Feynman–Kac Formula</i>	217
13.9.3 <i>Kolmogorov Equations</i>	218
13.9.4 <i>Kolmogorov Forward (Fokker–Planck (FPE)) Equation</i>	218
13.9.5 <i>Multi-Dimensional Problems and Boundary Conditions</i>	219
13.9.6 <i>Kolmogorov Backward Equation (KBE)</i>	220
13.10 First Exit-Time Problems	221
13.11 Summary and Conclusions	222

PART C**The Foundations of the Finite Difference Method (FDM)****CHAPTER 14****Mathematical and Numerical Foundations of the Finite Difference Method, Part I****225**

14.1 Introduction and Objectives	225
14.2 Notation and Prerequisites	226
14.3 What Is the Finite Difference Method, Really?	227
14.4 Fourier Analysis of Linear PDEs	227
14.4.1 <i>Fourier Transform for Advection Equation</i>	229
14.4.2 <i>Fourier Transform for Diffusion Equation</i>	230
14.5 Discrete Fourier Transform	232
14.5.1 <i>Finite and Infinite Dimensional Sequences and Their Norms</i>	232
14.5.2 <i>Discrete Fourier Transform (DFT)</i>	233
14.5.3 <i>Discrete von Neumann Stability Criterion</i>	235
14.5.4 <i>Some More Examples</i>	235
14.6 Theoretical Considerations	237
14.6.1 <i>Consistency</i>	237
14.6.2 <i>Stability</i>	238
14.6.3 <i>Convergence</i>	239
14.7 First-Order Partial Differential Equations	239
14.7.1 <i>Why First-Order Equations are Different: Essential Difficulties</i>	242
14.7.2 <i>A Simple Explicit Scheme</i>	243
14.7.3 <i>Some Common Schemes for Initial Value Problems</i>	245
14.7.4 <i>Some Other Schemes</i>	246
14.7.5 <i>General Linear Problems</i>	248
14.8 Summary and Conclusions	248

CHAPTER 15**Mathematical and Numerical Foundations of the Finite Difference Method, Part II****249**

15.1 Introduction and Objectives	249
15.2 A Short History of Numerical Methods for CDR Equations	250

15.2.1	<i>Temporal and Spatial Stability</i>	251
15.2.2	<i>Motivating Exponential Fitting Methods</i>	253
15.2.3	<i>Eliminating Temporal and Spatial Stability Problems</i>	254
15.3	Exponential Fitting and Time-Dependent Convection-Diffusion	257
15.4	Stability and Convergence Analysis	258
15.5	Special Limiting Cases	260
15.6	Stability for Initial Boundary Value Problems	260
15.6.1	<i>Gershgorin's Circle Theorem</i>	261
15.7	Semi-Discretisation for Convection-Diffusion Problems	264
15.7.1	<i>Essentially Positive Matrices</i>	265
15.7.2	<i>Fully Discrete Schemes</i>	267
15.8	Padé Matrix Approximation	269
15.8.1	<i>Padé Matrix Approximations</i>	270
15.9	Time-Dependent Convection-Diffusion Equations	275
15.9.1	<i>Fully Discrete Schemes</i>	275
15.10	Summary and Conclusions	276

CHAPTER 16

Sensitivity Analysis, Option Greeks and Parameter Optimisation, Part I		277
16.1	Introduction and Objectives	277
16.2	Helicopter View of Sensitivity Analysis	278
16.3	Black–Scholes–Merton Greeks	279
16.3.1	<i>Higher-Order and Mixed Greeks</i>	282
16.4	Divided Differences	282
16.4.1	<i>Approximation to First and Second Derivatives</i>	282
16.4.2	<i>Black–Scholes Numeric Greeks and Divided Differences</i>	285
16.5	Cubic Spline Interpolation	286
16.5.1	<i>Caveat: Cubic Splines with Sparse Input Data</i>	289
16.5.2	<i>Cubic Splines for Option Greeks</i>	290
16.5.3	<i>Boundary Conditions</i>	291
16.6	Some Complex Function Theory	292
16.6.1	<i>Curves and Regions</i>	293
16.6.2	<i>Taylor's Theorem and Series</i>	294
16.6.3	<i>Laurent's Theorem and Series</i>	295
16.6.4	<i>Cauchy–Goursat Theorem</i>	296
16.6.5	<i>Cauchy's Integral Formula</i>	297
16.6.6	<i>Cauchy's Residue Theorem</i>	298
16.6.7	<i>Gauss's Mean Value Theorem</i>	299
16.7	The Complex Step Method (CSM)	299
16.7.1	<i>Caveats</i>	302
16.8	Summary and Conclusions	302

CHAPTER 17**Advanced Topics in Sensitivity Analysis** **305**

17.1	Introduction and Objectives	305
17.2	Examples of CSE	305
17.2.1	<i>Simple Initial Value Problem</i>	306
17.2.2	<i>Population Dynamics</i>	307
17.2.3	<i>Comparing CSE and Complex Step Method (CSM)</i>	310
17.3	CSE and Black-Scholes PDE	310
17.3.1	<i>Black-Scholes Greeks: Algorithms and Design</i>	311
17.3.2	<i>Some Specific Black-Scholes Greeks</i>	312
17.4	Using Operator Calculus to Compute Greeks	313
17.5	An Introduction to Automatic Differentiation (AD) for the Impatient	314
17.5.1	<i>What Is Automatic Differentiation: The Details</i>	316
17.6	Dual Numbers	317
17.7	Automatic Differentiation in C++	318
17.8	Summary and Conclusions	319

PART D**Advanced Finite Difference Schemes for Two-Factor Problems****CHAPTER 18****Splitting Methods, Part I** **323**

18.1	Introduction and Objectives	323
18.2	Background and History	324
18.3	Notation, Prerequisites and Model Problems	325
18.4	Motivation: Two-Dimensional Heat Equation	328
18.4.1	<i>Alternating Direction Implicit (ADI) Method</i>	328
18.4.2	<i>Soviet (Operator) Splitting</i>	330
18.4.3	<i>Mixed Derivative and Yanenko Scheme</i>	331
18.5	Other Related Schemes for the Heat Equation	333
18.5.1	<i>D'Yakonov Method</i>	333
18.5.2	<i>Approximate Factorisation of Operators</i>	334
18.5.3	<i>Predictor-Corrector Methods</i>	337
18.5.4	<i>Partial Integro Differential Equations (PIDEs)</i>	338
18.6	Boundary Conditions	339
18.7	Two-Dimensional Convection PDEs	341
18.8	Three-Dimensional Problems	343
18.9	The Hopscotch Method	344
18.10	Software Design and Implementation Guidelines	346
18.11	The Future: Convection-Diffusion Equations	346
18.12	Summary and Conclusions	347

CHAPTER 19

The Alternating Direction Explicit (ADE) Method	349
19.1 Introduction and Objectives	349
19.2 Background and Problem Statement	351
19.3 Global Overview and Applicability of ADE	351
19.4 Motivating Examples: One-Dimensional and Two-Dimensional Diffusion Equations	352
19.4.1 Barakat and Clark (B&C) Method	353
19.4.2 Saul'yev Method	354
19.4.3 Larkin Method	355
19.4.4 Two-Dimensional Diffusion Problems	355
19.5 ADE for Convection (Advection) Equation	356
19.6 Convection-Diffusion PDEs	358
19.6.1 Example: Black–Scholes PDE	359
19.6.2 Boundary Conditions	360
19.6.3 Spatial Amplification Errors	361
19.7 Attention Points with ADE	362
The Consequences of Conditional Consistency	362
Call Pay-Off Behaviour at the Far Field	362
19.7.1 General Formulation of the ADE Method	362
19.8 Summary and Conclusions	364

CHAPTER 20

The Method of Lines (MOL), Splitting and the Matrix Exponential	365
20.1 Introduction and Objectives	365
20.2 Notation and Prerequisites: The Exponential Function	366
20.2.1 Initial Results	367
20.2.2 The Exponential of a Matrix	367
20.3 The Exponential of a Matrix: Advanced Topics	368
20.3.1 Fundamental Theorem for Linear Systems	368
Proof of Theorem 20.1.	369
20.3.2 An Example	369
20.4 Motivation: One-Dimensional Heat Equation	370
20.5 Semi-Linear Problems	373
20.6 Test Case: Double-Barrier Options	375
20.6.1 PDE Formulation	376
20.6.2 Using Exponential Fitting of Barrier Options	377
20.6.3 Performing MOL with Boost C++ odeint	378
20.6.4 Computing Sensitivities	381
20.6.5 American Options	384
20.7 Summary and Conclusions	384

CHAPTER 21

Free and Moving Boundary Value Problems	387
21.1 Introduction and Objectives	387
21.2 Background, Problem Statement and Formulations	388

21.3	Notation and Prerequisites	388
21.4	Some Initial Examples of Free and Moving Boundary Value Problems	389
21.4.1	<i>Single-Phase Melting Ice</i>	389
21.4.2	<i>Oxygen Diffusion</i>	390
21.4.3	<i>American Option Pricing</i>	391
21.4.4	<i>Two-Phase Melting Ice</i>	392
21.5	An Introduction to Parabolic Variational Inequalities	392
21.5.1	<i>Formulation of Problem: Test Case</i>	392
21.5.2	<i>Examples of Initial Boundary Value Problems</i>	395
21.6	An Introduction to Front-Fixing	399
21.6.1	<i>Front-Fixing for the Heat Equation</i>	399
21.7	Python Code Example: ADE for American Option Pricing	400
21.8	Summary and Conclusions	405

CHAPTER 22

Splitting Methods, Part II

407

22.1	Introduction and Objectives	407
22.2	Background and Problem Statement: The Essence of Sequential Splitting	408
22.3	Notation and Mathematical Formulation	408
22.3.1	<i>C_0 Semigroups</i>	408
22.3.2	<i>Abstract Cauchy Problem</i>	409
22.3.3	<i>Examples</i>	410
22.4	Mathematical Foundations of Splitting Methods	411
22.4.1	<i>Lie (Trotter) Product Formula</i>	411
22.4.2	<i>Splitting Error</i>	411
22.4.3	<i>Component Splitting and Operator Splitting</i>	413
22.4.4	<i>Splitting as a Discretisation Method</i>	413
22.5	Some Popular Splitting Methods	414
22.5.1	<i>First-Order (Lie-Trotter) Splitting</i>	415
22.5.2	<i>Predictor-Corrector Splitting</i>	415
22.5.3	<i>Marchuk's Two-Cycle (1-2-2-1) Method</i>	416
22.5.4	<i>Strang Splitting</i>	417
22.6	Applications and Relationships to Computational Finance	417
22.7	Software Design and Implementation Guidelines	418
22.8	Experience Report: Comparing ADI and Splitting	419
22.9	Summary and Conclusions	421

PART E**Test Cases in Computational Finance****CHAPTER 23****Multi-Asset Options**

425		
23.1	Introduction and Objectives	425
23.2	Background and Goals	426
23.3	The Bivariate Normal Distribution (BVN) and its Applications	427
23.3.1	<i>Computing BVN by Solving a Hyperbolic PDE</i>	430
23.3.2	<i>Analytical Solutions of Multi-Asset and Basket Options</i>	433
23.4	PDE Models for Multi-Asset Option Problems: Requirements and Design	435
23.4.1	<i>Domain Transformation</i>	435
23.4.2	<i>Numerical Boundary Conditions</i>	435
23.5	An Overview of Finite Difference Schemes for Multi-Asset Option Problems	436
23.5.1	<i>Common Design Principles</i>	436
23.5.2	<i>Detailed Design</i>	438
23.5.3	<i>Testing the Software</i>	440
23.6	American Spread Options	440
23.7	Appendices	442
23.7.1	<i>Traditional Approach to Numerical Boundary Conditions</i>	442
23.7.2	<i>Top-Down Design of Monte Carlo Applications</i>	443
23.8	Summary and Conclusions	444

CHAPTER 24**Asian (Average Value) Options**

447		
24.1	Introduction and Objectives	447
24.2	Background and Problem Statement	448
24.2.1	<i>Challenges</i>	449
24.3	Prototype PDE Model	450
24.3.1	<i>Similarity Reduction</i>	451
24.4	The Many Ways to Handle the Convective Term	452
24.4.1	<i>Method of Lines (MOL)</i>	452
24.4.2	<i>Other Schemes</i>	454
24.4.3	<i>A Stable Monotone Upwind Scheme</i>	455
24.5	ADE for Asian Options	455
24.6	ADI for Asian Options	456
24.6.1	<i>Modern ADI Variations</i>	458
24.7	Summary and Conclusions	459

CHAPTER 25**Interest Rate Models**

461		
25.1	Introduction and Objectives	461
25.2	Main Use Cases	462

25.3	The CIR Model	462
25.3.1	<i>Analytic Solutions</i>	463
25.3.2	<i>Initial Boundary Value Problem</i>	466
25.4	Well-Posedness of the CIRPDE Model	466
25.4.1	<i>Gronwall's Inequalities</i>	467
25.4.2	<i>Energy Inequalities</i>	468
25.5	Finite Difference Methods for the CIR Model	469
25.5.1	<i>Numerical Boundary Conditions</i>	470
25.6	Heston Model and the Feller Condition	471
25.7	Summary and Conclusion	475
CHAPTER 26		
Epilogue Models Follow-Up Chapters 1 to 25		477
26.1	Introduction and Objectives	477
26.2	Mixed Derivatives and Monotone Schemes	478
26.2.1	<i>The Maximum Principle and Mixed Derivatives</i>	478
26.2.2	<i>Some Examples</i>	480
26.2.3	<i>Code Sample Method of Lines (MOL) for Two-Factor Hull–White Model</i>	481
26.3	The Complex Step Method (CSM) Revisited	483
26.3.1	<i>Black–Scholes Greeks Using CSM and the Faddeeva Function</i>	483
26.3.2	<i>CSM and Functions of Several Complex Variables</i>	487
26.3.3	<i>C++ Code for Extended CSM</i>	488
26.3.4	<i>CSM for Non-Linear Solvers</i>	492
26.4	Extending the Hull–White: Possible Projects	493
26.5	Summary and Conclusions	495
Bibliography		497
Index		505

Preface

This book is a detailed introduction to the mathematical theory and foundations of ordinary and partial differential equations, their approximation by the finite difference method and applications to computational finance.

Major benefits of the book are:

- Step-by-step, incremental build-up of the material.
- Examples and algorithms worked out in detail. Opportunity to modify the algorithms and extend them to your own applications.
- Modern, state-of-the art numerical schemes for PDEs in finance.
- Guidelines on C++ coding (C++11 to C++20); the book is the ideal companion to the author's book *Financial Instrument Pricing Using C++* (second edition, 2018).
- The book is structured so that the material can be applied to a range of existing and new application areas.
- We resolve a number of outstanding issues and improve several less-than-optimal numerical methods in finance.

We have divided the book into five parts, with each part addressing a single major issue.

Part A (Chapters 1 to 7) introduces the mathematical and numerical analysis concepts that are needed to understand the finite difference method and its application to computational finance. The main reason for writing the chapters is to make the book as self-contained as possible and to introduce and define standardised notation and results that we use in later chapters. Furthermore, the presented material can also be used as a standalone reference.

We realise that some readers will not be familiar with all of the *building blocks* that are needed to write finite difference schemes for the Black–Scholes PDE; for this reason, Part A was written to resolve this issue. We identify and discuss all the steps to design and implement a finite difference solver for one-factor finance PDEs. To this end, we take an incremental and *single responsibility* approach by focusing on one major topic in each chapter:

- Initial value problems and boundary value problems and their numerical approximation.
- Vector spaces, matrix theory and numerical linear algebra.
- My first Crank–Nicolson and Alternating Direction Explicit (ADE) methods for the one-factor Black–Scholes PDE.

Finally, Chapter 1 is devoted to major concepts (such as continuity and differentiability) in *real analysis* that permeate the book, and for this reason it is important to understand them.

Part A can be used by readers with no prior knowledge of partial differential equations or the finite difference method. It can be used as a mini-course or mini-project to learn the material.

Part B (Chapters 8 to 13) discusses a number of rigorous mathematical techniques relating to boundary value problems and initial boundary value problems for elliptic and parabolic partial differential equations in two-space variables. The goal is to identify and elaborate the underlying theory to unambiguously specify these problems before mapping them to a numerical solution, thereby filling some ‘mathematical gaps’ in current practice. In particular, we develop strategies to preprocess and modify a PDE before we approximate it by the finite difference method, thus removing ad hoc and heuristic tricks that are often used to arrive at (hopefully) robust numerical schemes.

The chapters in this part fill a major gap in the application of PDE/FDM to finance. In general, most of the finance literature glosses over the niceties of analysing PDEs mathematically before approximating them using the finite difference method. The new approach resolves many of issues and heuristic approaches. Some new improvements for two-factor PDEs are:

- Transform a PDE with a mixed derivative term to one in which this term has been removed (the *canonical PDE*).
- Why domain transformation is better than domain truncation in general.
- A rigorous set of mathematical techniques (Fichera theory, energy estimates) to discover the correct boundary conditions for finance problems.
- The deep relationship between PDEs and stochastic differential equations (SDEs). We discuss formulations and results that are important in calibration applications.

Part B prepares the way for a seamless route from PDEs to robust and understandable finite difference schemes that approximate them. It eliminates much trial-and-error experimentation. In a sense the topics in Part B serve as a reference for the more hands-on topics in later chapters. At the very least, it is important to be aware of the main results.

Part C (Chapters 14 to 17) introduces the mathematical background to the finite difference method for initial boundary value problems for parabolic partial differential equations. It encapsulates in one place all the background information that is needed to construct stable and accurate finite difference schemes for time-dependent problems. The schemes will be applied to one-factor and two-factor finance PDEs in later chapters. The advantage is that the chapters discuss finite difference schemes for generic PDEs that can then be applied to finance PDEs. We also devote two dedicated chapters to *Sensitivity Analysis*, and we propose at least five methods to compute the derivatives of solutions of initial boundary value problems with respect to underlying parameters. In finance, these are sometimes called *option greeks*.

The chapters in this part discuss both new as well as established methods to gain a deep understanding of the foundations of the finite difference method and its applications to finance, and beyond:

- Defining stability for initial value problems, consistency and the Lax Equivalence Theorem.

- Modern stability analysis for initial boundary value problems : discrete maximum principle, convergence.
- Six ways to compute sensitivities.
- A compact introduction to complex analysis: the Complex Step Method (CSM).

A good working knowledge of the topics in Part C is essential in order to proceed in an effective manner.

Part D (Chapters 18 to 22) introduces a number of modern and popular finite difference methods (approximately six) to approximate the solution of initial boundary value problems for two-factor partial differential equations. To our knowledge, this is the only book that discusses these methods as well as their comparative strengths together with their applications to option pricing and hedging.

In this part we offer a number of robust and accurate schemes for a range of PDEs:

- Soviet Splitting (Marchuk, Yanenko, Strang).
- Alternating Direction Explicit (ADE) (Saul'yev)
- Method of Lines (MOL).
- Front-fixing and variational methods for free boundary value problems.

We lay the mathematical foundations of all splitting methods by introducing semi-group theory and generalised exponential functions. In short, these schemes in combination with the PDE preprocessing techniques in Part B open up a world beyond the perennial Crank–Nicolson and Alternating Direction Implicit (ADI) methods.

A good working knowledge of the algorithms in Part D is essential. In particular, the ability to program these algorithms (in C++ for obvious reasons) is ideal. Algorithms should work on paper and in the computer.

Part E (Chapters 23 to 26) is concerned with applications of the techniques from the first twenty-two chapters. We discuss finite difference schemes for one-factor and two-factor, stochastic volatility, and interest problems. We also revisit some earlier chapters with a view to examining them in even more detail or from new perspectives. Finally, we offer tips and guidelines on extending the results in this book to other finance problems.

This part uses most of the mathematical and numerical techniques of the first twenty-two chapters, and we apply them to a range of linear and nonlinear PDEs. For each problem, we can preprocess it using the techniques in Part B before deciding which numerical schemes from Part D to use. In particular, we scope the chapters by addressing the following problems and solutions:

- Spread options with domain transformation with Yanenko, Predictor-Corrector and Marchuk two-cycle methods.
- Asian option pricing using ADE. Clarification of some issues relating to boundary conditions. Modern ADI methods.
- Interest rate models (CIR), Feller condition and energy inequalities. ADE method. Relationship with Heston model.
- Monotone schemes (viscosity solution) for two-factor problems with mixed derivatives and applications to Uncertain Volatility Models (UVM).
- One-factor and two-factor Hull–White models using ADE and MOL.

Who Should Read this Book?

This book has universal appeal because it is a focused and detailed introduction to the mathematical theory and foundations of ordinary and partial differential equations, their approximation by the finite difference method and subsequent applications to computational finance. It is suitable both as an entry-level introduction as well as a detailed treatment of modern methods as used by industry quants and MSc/MFE students in finance. The topics in the book have major applications to numerical analysis, science and engineering. In fact, most of the PDE/FDM methods have their origins in these fields.

For more information relating to computational finance, including links to resources and the author's online courses, please visit www.datasim.nl.

PART **A**

Mathematical Foundation for One-Factor Problems

Real Analysis Foundations for this Book

The beginner should not be discouraged if he finds he does not have the prerequisites for reading the prerequisites.

Paul Halmos

1.1 INTRODUCTION AND OBJECTIVES

In this chapter we introduce a number of mathematical concepts and methods that underlie many of the topics in this book. The most urgent attention points revolve around functions of real variables, their properties and the ways they are used in applications. We discuss the most important topics from *real analysis* to help us in our understanding of partial differential equations (PDEs). A definition of real analysis is:

In mathematics, real analysis is the branch of mathematical analysis that studies the behavior of real numbers, sequences and series of real numbers, and real functions. Some particular properties of real-valued sequences and functions that real analysis studies include convergence, limits, continuity, smoothness, differentiability and integrability.

Real analysis is distinguished from complex analysis, which deals with the study of complex numbers and their functions.

(*Wikipedia*)

A related branch of mathematics is *calculus*, which we learn at school:

Calculus, originally called infinitesimal calculus or ‘the calculus of infinitesimals’, is the mathematical study of continuous change, in the same way that geometry is the study of shape and algebra is the study of generalizations of arithmetic operations.

It has two major branches, differential calculus and integral calculus; the former concerns instantaneous rates of change, and the slopes of curves, while integral calculus concerns accumulation of quantities, and areas under or between curves. These two branches are related to each other by the fundamental theorem of calculus, and they make use of the fundamental notions of convergence of infinite sequences and infinite series to a well-defined limit.

(Wikipedia)

In practice, there is a distinction between calculus and real analysis. Calculus entails techniques (and tricks) to differentiate and integrate functions. It does not discuss the conditions under which a function is continuous or differentiable. It assumes that it is allowed to carry out these operations on functions. Real analysis, on the other hand, does discuss these issues and more; for example:

- Continuous functions: How do we recognise them and prove that a function is continuous?
- The different kinds of discontinuous functions.
- Differential calculus from a real-analysis viewpoint.
- Taylor's theorem.
- An introduction to metric spaces and Cauchy sequences.

In our opinion, these topics are necessary prerequisites for the rest of this book.

Knowledge of vector (linear) analysis and numerical linear algebra is also a prerequisite for computational finance. To this end, we devote Chapters 4 and 5 to these topics. Finally, complex variables and complex functions (which are at the heart of complex analysis) are introduced in Chapter 16. We use the notation \forall to mean ‘for all’ and \exists to mean ‘there exists’.

1.2 CONTINUOUS FUNCTIONS

In this section we are mainly concerned with real-valued functions of a real variable, that is $f : \mathbb{R} \rightarrow \mathbb{R}$. In rough terms, a continuous function is one that can be drawn by hand without taking the pen from paper. In other words, a continuous function does not have jumps or breaks, but it is allowed to have sharp bends and kinks. Examples of continuous functions are:

$$\begin{aligned} f : \mathbb{R} &\rightarrow \mathbb{R}, \quad f(x) = x^2 \\ f : \mathbb{R} &\rightarrow \mathbb{R}, \quad f(x) = \max(0, x). \end{aligned} \tag{1.1}$$

We can see that these functions are continuous just by drawing them. The first function is ‘smoother’ than the second function, the latter being similar to a one-factor call or put payoff on the one hand and a *Rectified Linear Unit* (ReLU) activation function

on the other hand (Goodfellow, Bengio and Courville (2016)). Intuitively, a function f is continuous if $f(x) \rightarrow f(p)$ when $x \rightarrow p$, no matter how x approaches p . Alternatively, small changes in x lead to small changes in $f(x)$.

If we formally differentiate the above ReLU function (1.1), we get the famous discontinuous *Heaviside function*:

$$H(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0. \end{cases} \quad (1.2)$$

A *discontinuous function* is one that is not continuous. Another discontinuous function is:

$x \in \mathbb{R}$, $[x] \equiv$ largest integer n s.t. $n \leq x \leq n + 1$.

Define $f(x) = [x]$; let $p \in \mathbb{Z}$ (integer).

Then taking left and right limits gives different answers, showing that the function is not continuous.

1. $x < p \Rightarrow f(x) = p - 1$
2. $x > p \Rightarrow f(x) = p$

Thus $\lim_{x \rightarrow p^-} f(x) = p - 1$, $\lim_{x \rightarrow p^+} f(x) = p$.

1.2.1 Formal Definition of Continuity

The following definition is based on the fact that small changes in x lead to small changes in $f(x)$.

Definition 1.1

$\lim_{x \rightarrow p} f(x) = A$ means $\forall \varepsilon > 0 \ \exists \delta > 0$ s.t.

$|f(x) - A| < \varepsilon$ when $0 < |x - p| < \delta$.

Some properties of continuous functions $f(x)$ and $g(x)$ are:

$$\lim_{x \rightarrow p} (f(x) \pm g(x)) = \lim_{x \rightarrow p} f(x) \pm \lim_{x \rightarrow p} g(x)$$

$$\lim_{x \rightarrow p} (f(x)g(x)) = \lim_{x \rightarrow p} f(x) \lim_{x \rightarrow p} g(x)$$

$$\lim_{x \rightarrow p} \frac{f(x)}{g(x)} = \frac{\lim_{x \rightarrow p} f(x)}{\lim_{x \rightarrow p} g(x)}, \quad g(x) \neq 0.$$

1.2.2 An Example

It can be a mathematical challenge to prove that a function is continuous using the above ‘*epsilon-delta*’ approach in Definition 1.1. One approach is to use the well-known technique of splitting the problem into several mutually exclusive cases, solving each case separately and then merging the corresponding partial solutions to form the desired solution. To this end, let us examine the square root function:

$$f : \mathbb{R}^+ \rightarrow \mathbb{R}^+, f(x) = \sqrt{x}. \quad (1.3)$$

We show that there exists $\delta > 0$ such that for $x \geq 0$:

$$|x - y| < \delta \Rightarrow |\sqrt{x} - \sqrt{y}| < \epsilon \quad \forall y \in \mathbb{R}^+.$$

Then:

$$\sqrt{x} - \sqrt{y} = \frac{x - y}{\sqrt{x} + \sqrt{y}}.$$

We now consider two cases:

Case 1 : $x > 0$. Then:

$$|x - y| < \delta \Rightarrow |\sqrt{x} - \sqrt{y}| \leq \frac{|x - y|}{\sqrt{x}} = \frac{\delta}{\sqrt{x}} = \epsilon.$$

Choose $\delta = \epsilon\sqrt{x}$.

Case 2: $x = 0$. Then:

$$|x - y| < \delta \Rightarrow |\sqrt{x} - \sqrt{y}| = \frac{|x - y|}{\sqrt{x} + \sqrt{y}} = \frac{y}{\sqrt{y}} = \sqrt{y} = \epsilon.$$

Hence:

$$|-y| = |y| < \delta \Rightarrow \sqrt{y} = \epsilon \Rightarrow \sqrt{\delta} < \epsilon \Rightarrow \delta < \epsilon^2.$$

Choose $\delta = \epsilon^2$.

We have thus proved that the square root function is continuous.

1.2.3 Uniform Continuity

In general terms, *uniform continuity* guarantees that $f(x)$ and $f(y)$ can be made as close to each other as we please by requiring that x and y be sufficiently close to each other. This is in contrast to ordinary continuity, where the distance between $f(x)$ and $f(y)$ may depend on x and y themselves. In other words, in Definition 1.1 δ depends only on ϵ and not on the points in the domain. Continuity itself is a *local property* because a function f is or is not continuous at a particular point and continuity can be determined by looking at the values of the function in an arbitrary small neighbourhood of that point. Uniform continuity, on the other hand, is a *global property* of f because the definition

refers to pairs of points rather than individual points. The new definition in this case for a function f defined in an interval I is:

$$\forall \epsilon > 0 \exists \delta > 0 \text{ s.t. } \forall x, y \in I : |x - y| < \delta \Rightarrow |f(x) - f(y)| < \epsilon.$$

Let us take an example of a uniformly continuous function:

$$f : \mathbb{R} \rightarrow \mathbb{R}, \quad f(x) = 3x + 7. \quad (1.4)$$

Then $|f(x) - f(y)| = |3x + 7 - (3y + 7)| = 3|x - y| < 3\delta < \epsilon, \quad (x, y \in \mathbb{R}).$

Choose $\delta = \epsilon/3$.

In general, a continuous function on a closed interval is uniformly continuous. An example is:

$$f(x) = x^2 \text{ on } I = [0, 2]. \quad (1.5)$$

Let $x, y \in I$. Then:

$$|f(x) - f(y)| = (x + y)|x - y| < (2 + 2)\delta = \epsilon.$$

Choose $\delta = \epsilon/4$.

An example of a function that is continuous and nowhere differentiable is the *Weierstrass function* that we can write as a *Fourier series*:

$$f(x) = \sum_{n=0}^{\infty} a^n \cos(b^n \pi x), \quad 0 < a < 1, \quad (1.6)$$

b is a positive odd integer and $ab > 1 + \frac{3}{2}\pi$.

This is a jagged function that appears in models of Brownian motion. Each partial sum is continuous, and hence by the *uniform limit theorem* (which states that the uniform limit of any sequence of continuous functions is continuous), the series (1.6) is continuous.

1.2.4 Classes of Discontinuous Functions

A function that is not continuous at some point is said to be *discontinuous* at that point. For example, the Heaviside function (1.2) is not continuous at $x = 0$. In order to determine if a function is continuous at a point x in an interval (a, b) we apply the test:

$$f(x+) = q \text{ if } f(t_n) \rightarrow q, n \rightarrow \infty \text{ for all sequences } \{t_n\} \text{ in } (x, b) \text{ s.t. } t_n \rightarrow x$$

$$f(x-) = q \text{ if } f(t_n) \rightarrow q, n \rightarrow \infty \text{ for all sequences } \{t_n\} \text{ in } (a, x) \text{ s.t. } t_n \rightarrow x$$

$$\exists \lim_{t \rightarrow x} f(t) \iff f(x+) = f(x-) = \lim_{t \rightarrow x} f(t) = f(x).$$

There are two (simple discontinuity) main categories of discontinuous functions:

- *First kind:* $f(x+) = \lim_{t \rightarrow x+} f(t)$ and $f(x-) = \lim_{t \rightarrow x-} f(t)$ exists. Then either we have $f(x+) \neq f(x-)$ or $f(x+) = f(x-) \neq f(x)$.
- *Second kind:* a discontinuity that is not of the first kind.

Examples are:

$$f(x) = \begin{cases} 1, & x \text{ rational } (x \in \mathbb{Q}) \\ 0, & x \text{ not rational, } x \notin \mathbb{Q} \\ \text{2nd kind: Neither } f(x+) \text{ nor } f(x-) \text{ exists.} \end{cases}$$

$$f(x) = \begin{cases} x + 2, & -3 < x < -2 \\ -x - 2, & -2 \leq x < 0 \\ x + 2, & 0 \leq x < 1 \\ \text{Simple discontinuity at } x = 0. \end{cases}$$

You can check that this latter function has a discontinuity of the first kind at $x = 0$.

1.3 DIFFERENTIAL CALCULUS

The *derivative* of a function is one of its fundamental properties. It represents the rate of change of the slope of the function: in other words, how fast the function changes with respect to changes in the independent variable. We focus on real-valued functions of a real variable.

Let $f : \mathbb{R} \rightarrow \mathbb{R}$. Then the derivative of f at x (if it exists) is defined by the limit for $x \in [a, b]$:

$$\varphi(t) = \frac{f(t) - f(x)}{t - x} \quad (t \neq x),$$

$$f'(x) = \lim_{t \rightarrow x} \varphi(t) \quad (1.7)$$

or

$$\frac{df(x)}{dx} = f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}.$$

This limit may not exist at certain points, and it is possible to define right-hand and left-hand limits, that is, one-sided derivatives.

Some results that we learn in high school are:

$$(f + g)'(x) = f'(x) + g'(x)$$

$$(fg)'(x) = f'(x)g(x) + f(x)g'(x) \quad (1.8)$$

$$\left(\frac{f}{g}\right)'(x) = \frac{g(x)f'(x) - g'(x)f(x)}{g^2(x)} \quad (g(x) \neq 0).$$

A *composite function* is a function that we can differentiate using the *chain rule* that we state as follows:

$$x \in [a, b], \exists f'(x) \text{ with } g \text{ differentiable at } f(x).$$

Then:

$$\begin{aligned} h(t) &\equiv g(f(t)), \quad a \leq t \leq b \text{ has derivative} \\ h'(x) &= g'(f(x))f'(x). \end{aligned} \tag{1.9}$$

A simple example of use is:

$$\begin{aligned} f(x) &= x^2, \quad g(y) = 2y + 1 \\ h(x) &= g(f(x)) = g(x^2) = 2x^2 + 1 \\ h'(x) &= g'(f(x))f'(x) = 4x (= 2 * 2x). \end{aligned}$$

More challenging examples of composite functions are:

$$\begin{aligned} f(x) &= \begin{cases} x \sin \frac{1}{x}, & x \neq 0 \\ 0, & x = 0 \end{cases} \\ f'(x) &= \sin \frac{1}{x} - \frac{1}{x} \cos \frac{1}{x}, \quad x \neq 0 \\ f'(0) &\text{ does not exist.} \\ f(x) &= \begin{cases} x^2 \sin \frac{1}{x}, & x \neq 0 \\ 0, & x = 0 \end{cases} \\ f'(x) &= 2x \sin \frac{1}{x} - \cos \frac{1}{x}, \quad x \neq 0 \\ f'(0) &= \lim_{t \rightarrow 0} \frac{f(t) - f(0)}{t - 0} = 0. \end{aligned}$$

1.3.1 Taylor's Theorem

Taylor's theorem allows us to expand a function as a series involving higher-order derivatives of a function. We take the *Cauchy form* (with exact remainder):

f is n times differentiable

$$f(b) = \sum_{k=0}^{n-1} \frac{(b-a)^k}{k!} f^{(k)}(a) + R_n \tag{1.10}$$

where:

$$R_n = \frac{(b-\xi)^n f^{(n)}(\xi)}{n!}, \quad a < \xi < b$$

and:

$$f' = f^{(1)} = \frac{df}{dx}, f^{(2)} = \frac{d^2f}{dx^2}$$

$$f^{(n)}(x) = (f^{(n-1)}(x))' = \frac{d}{dx} (f^{(n-1)}(x)).$$

We conclude with a discussion of the *exponential function*. It is the only function that is the same as its derivative. To see this, we use the formal definition (1.7) of a derivative (and noting that $e^x e^y = e^{x+y}, x, y \in \mathbb{R}$):

$$\frac{d}{dx} e^x = \lim_{h \rightarrow 0} \left(\frac{e^{x+h} - e^x}{h} \right) = e^x \lim_{h \rightarrow 0} \frac{e^h - 1}{h} = e^x, x \in \mathbb{R}.$$

We summarise some useful properties of the exponential function:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

$$e^x = \lim_{n \rightarrow \infty} \left(1 + \frac{x}{n} \right)^n$$

$$\frac{d}{dx} e^x = e^x$$

$$e^{x+y} = e^x e^y$$

$$y = \log x \iff x = e^y$$

$$\log(ab) = \log a + \log b.$$

$$\frac{d^n}{dx^n} e^x = e^x \quad \forall n \geq 1$$

$$e^x = \sum_{k=0}^{n-1} \frac{x^k}{k!} + \mathbb{R}_n \text{ where } \mathbb{R}_n = \frac{x^n}{n!} e^\xi, \quad \xi < x.$$

1.3.2 Big O and Little o Notation

For many applications we need a definition of the *asymptotic behaviour* of quantities such as functions and series; in particular we wish to find bounds on mathematical expressions and applications in computer science. To this end, we introduce the *Landau symbols* O and o.

Definition 1.2 (O-Notation).

$$f(x) = O(g(x)) \text{ as } x \rightarrow \infty \text{ if } \exists M > 0, \exists x_0 \text{ s.t.}$$

$$|f(x)| \leq M|g(x)| \text{ for } x > x_0$$

$$f(x) = O(g(x)) \text{ as } x \rightarrow a \text{ if } |f(x)| \leq M|g(x)| \text{ for } |x - a| < \delta$$

$$\text{Unified definition: } \lim_{x \rightarrow a} \frac{f(x)}{g(x)} < \infty.$$

An example is:

$$\begin{aligned}f(x) &\equiv 6x^4 - 7x^2 + 2 \\g(x) &\equiv x^4 \\f(x) &= O(g(x)) \text{ as } x \rightarrow \infty \\f_n &\equiv 2n^3 + 6n^2 + 5(\log n)^3 \\f_n &= O(n^3) \text{ as } n \rightarrow \infty.\end{aligned}$$

Definition 1.3 (o-Notation).

$$\begin{aligned}f(x) &= o(g(x)) \text{ as } x \rightarrow \infty \\&\text{if } \lim_{x \rightarrow \infty} \frac{f(x)}{g(x)} = 0.\end{aligned}$$

An example is:

$$\begin{aligned}2x &= o(x^2) \\2x^2 &\neq o(x^2) \\1/x &= o(1).\end{aligned}$$

We note that *complexity analysis* applies to both continuous and discrete functions.

1.4 PARTIAL DERIVATIVES

In general, we are interested in functions of two (or more) variables. We consider a function of the form:

$$z = f(x, y).$$

The variables x and y can take values in a given bounded or unbounded interval. First, we say that $f(x, y)$ is continuous at (a, b) if the limit:

$$\lim_{\substack{x \rightarrow a \\ y \rightarrow b}} f(x, y)$$

exists and is equal to $f(a, b)$. We now need definitions for the derivatives of f in the x and y directions.

In general, we calculate the *partial derivatives* by keeping one variable fixed and differentiating with respect to the other variable; for example:

$$\begin{aligned}z &= f(x, y) = e^{kx} \cos my \\ \frac{\partial z}{\partial x} &= ke^{kx} \cos my \\ \frac{\partial z}{\partial y} &= -me^{kx} \sin my.\end{aligned}$$

We now discuss the situation when we introduce a change of variables into some problem and then wish to calculate the new partial derivatives. To this end, we start with the variables (x, y) , and we define new variables (u, v) . We can think of these as ‘original’ and ‘transformed’ coordinate axes, respectively. Now define the function $z(u, v)$ as follows:

$$z = z(u, v), \quad u = u(x, y), \quad v = v(x, y).$$

This can be seen as *a function of a function*. The result that we are interested in is the following: if z is a differentiable function of (u, v) and u, v are themselves continuous functions of x, y , with partial derivatives, then the following rule holds:

$$\begin{aligned}\frac{\partial z}{\partial x} &= \frac{\partial z}{\partial u} \frac{\partial u}{\partial x} + \frac{\partial z}{\partial v} \frac{\partial v}{\partial x} \\ \frac{\partial z}{\partial y} &= \frac{\partial z}{\partial u} \frac{\partial u}{\partial y} + \frac{\partial z}{\partial v} \frac{\partial v}{\partial y}.\end{aligned}\tag{1.11}$$

This is a fundamental result that we shall apply in this chapter. We take a simple example of Equation (1.11) to show how things work. To this end, consider the *Laplace equation* in Cartesian geometry:

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

We now wish to transform this equation into an equation in a circular region defined by the polar coordinates:

$$x = r \cos \theta, \quad y = r \sin \theta.$$

The derivative in r is given by:

$$\frac{\partial u}{\partial r} = \frac{\partial u}{\partial x} \frac{\partial x}{\partial r} + \frac{\partial u}{\partial y} \frac{\partial y}{\partial r} = \cos \theta \frac{\partial u}{\partial x} + \sin \theta \frac{\partial u}{\partial y}$$

and you can check that the derivative with respect to θ is:

$$\frac{\partial u}{\partial \theta} = -r \sin \theta \frac{\partial u}{\partial x} + r \cos \theta \frac{\partial u}{\partial y}$$

hence:

$$\frac{\partial u}{\partial x} = \cos \theta \frac{\partial u}{\partial r} - \frac{1}{r} \sin \theta \frac{\partial u}{\partial \theta}$$

$$\frac{\partial u}{\partial y} = \sin \theta \frac{\partial u}{\partial r} + \frac{1}{r} \cos \theta \frac{\partial u}{\partial \theta}$$

and:

$$\begin{aligned}\frac{\partial^2 u}{\partial x^2} &= \cos\theta \frac{\partial}{\partial r} \left(\frac{\partial u}{\partial x} \right) - \frac{1}{r} \sin\theta \frac{\partial}{\partial \theta} \left(\frac{\partial u}{\partial x} \right) \\ \frac{\partial^2 u}{\partial y^2} &= \sin\theta \frac{\partial}{\partial r} \left(\frac{\partial u}{\partial y} \right) + \frac{1}{r} \cos\theta \frac{\partial}{\partial \theta} \left(\frac{\partial u}{\partial y} \right).\end{aligned}$$

Combining these results allows us to write Laplace's equation in polar coordinates as follows:

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r} \frac{\partial u}{\partial r} + \frac{1}{r^2} \frac{\partial^2 u}{\partial \theta^2} = 0.$$

Thus, the original heat equation in Cartesian coordinates is transformed to a PDE of convection-diffusion type in polar coordinates.

We can find a solution to this problem using the *Separation of Variables method*, for example.

1.5 FUNCTIONS AND IMPLICIT FORMS

Some problems use functions of two variables that are written in the *implicit form*:

$$f(x, y) = 0.$$

In this case we have an implicit relationship between the variables x and y . We assume that y is a function of x . The basic result for the differentiation of this *implicit function* is:

$$df \equiv \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy = 0 \quad (1.12a)$$

or:

$$\frac{dy}{dx} = -\frac{\partial f / \partial x}{\partial f / \partial y} \quad (1.12b)$$

We now use this result by posing the following problem. Consider the transformation:

$$\left. \begin{array}{l} u = u(x, y) \\ v = v(x, y) \end{array} \right\} \text{original equations}$$

and suppose we wish to transform back:

$$\left. \begin{array}{l} x = x(u, v) \\ y = y(u, v) \end{array} \right\} \text{find } x, y \text{ (inverse functions).}$$

To this end, we examine the following *differentials*:

$$\begin{aligned} du &= \frac{\partial u}{\partial x} dx + \frac{\partial u}{\partial y} dy \\ dv &= \frac{\partial v}{\partial x} dx + \frac{\partial v}{\partial y} dy.\end{aligned} \quad (1.13)$$

Let us assume that we wish to find dx and dy , given that all other quantities are known. Some arithmetic applied to Equation (1.13) (two equations in two unknowns!) results in:

$$dx = \left(\frac{\partial v}{\partial y} du - \frac{\partial u}{\partial y} dv \right) / J$$

$$dy = \left(-\frac{\partial v}{\partial x} du + \frac{\partial u}{\partial x} dv \right) / J$$

where J is the *Jacobian determinant* defined by:

$$J = \begin{vmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{vmatrix} = \frac{\partial(u, v)}{\partial(x, y)}.$$

We can thus conclude the following result.

Theorem 1.1 The functions $x = F(u, v)$ and $y = G(u, v)$ exist if:

$$\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial v}{\partial x}, \frac{\partial v}{\partial y}$$

are continuous at (a, b) and if the Jacobian determinant is non-zero at (a, b) .

Let us take the example:

$$u = x^2/y, v = y^2/x.$$

You can check that the Jacobian is given by:

$$\frac{\partial(u, v)}{\partial(x, y)} = \begin{vmatrix} 2x/y & -x^2/y^2 \\ -y^2/x^2 & 2y/x \end{vmatrix} = 3 \neq 0$$

Solving for x and y gives:

$$x = u^{2/3}v^{1/3}, y = u^{1/3}v^{2/3}.$$

You need to be comfortable with partial derivatives. A good reference is Widder (1989).

1.6 METRIC SPACES AND CAUCHY SEQUENCES

Section 1.6 may be skipped on a first reading without loss of continuity.

1.6.1 Metric Spaces

We work with sets and other mathematical structures in which it is possible to assign a so-called *distance function* or *metric* between any two of their elements. Let us suppose that X is a set, and let x, y and z be elements of X . Then a *metric* d on X is a non-negative real-valued function of two variables having the following properties:

$$D1: d(x, y) \geq 0; \quad d(x, y) = 0 \text{ if and only if } x = y$$

$$D2: d(x, y) = d(y, x)$$

$$D3: d(x, y) \leq d(x, z) + d(z, y) \text{ where } x, y, z \in X.$$

The concept of distance is a generalisation of the difference between two real numbers or the distance between two points in n -dimensional Euclidean space, for example.

Having defined a metric d on a set X , we then say that the pair (X, d) is a *metric space*. We give some examples of metrics and metric spaces:

1. We define the set X of all continuous real-valued functions of one variable on the interval $[a, b]$ (we denote this space by $C[a, b]$), and we define the metric:

$$d(f, g) = \max \{|f(t) - g(t)|; t \in [a, b]\}.$$

Then (X, d) is a metric space.

2. n -dimensional Euclidean space, consisting of vectors of real or complex numbers of the form:

$$x = (x_1, \dots, x_n), \quad y = (y_1, \dots, y_n)$$

with metric:

$$d(x, y) = \max\{|x_j - y_j|; j = 1, \dots, n\} \text{ or using the notation for a norm } d(x, y) = \|x - y\|_\infty.$$

3. Let $L^2[a, b]$ be the space of all square-integrable functions on the interval $[a, b]$:

$$\int_a^b |f(x)|^2 dx < \infty.$$

We can then define the distance between two functions f and g in this space by the metric:

$$d(f, g) = \|f - g\|_2 \equiv \left\{ \int_a^b |f(x) - g(x)|^2 \right\}^{1/2}.$$

This metric space is important in many branches of mathematics, including probability theory and stochastic calculus.

4. Let X be a non-empty set and let the metric d be defined by:

$$d(x, y) = \begin{cases} 0, & \text{if } x = y \\ 1, & \text{if } x \neq y. \end{cases}$$

Then (X, d) is a metric space.

Many of the results and theorems in mathematics are valid for metric spaces, and this fact means that the same results are valid for all specialisations of these spaces.

1.6.2 Cauchy Sequences

We define the concept of convergence of a sequence of elements of a metric space X to some element that may or may not be in X . We introduce some definitions that we state for the set of real numbers, but they are valid for any *ordered field*, which is basically a set of numbers for which every non-zero element has a multiplicative inverse and there is a certain ordering between the numbers in the field.

Definition 1.4 A sequence (a_n) of elements on the real line \mathbb{R} is said to be *convergent* if there exists an element $a \in \mathbb{R}$ such that for each positive element ε in \mathbb{R} there exists a positive integer n_0 such that:

$$|a_n - a| < \varepsilon \text{ whenever } n \geq n_0.$$

A simple example is to show that the sequence $\left\{\frac{1}{n}\right\}$, $n \geq 1$ converges to 0. To this end, let ε be a positive real number. Then there exists a positive integer $n_0 > 1/\varepsilon$ such that $|\frac{1}{n} - 0| = \frac{1}{n} < \varepsilon$ whenever $n \geq n_0$.

Definition 1.5 A sequence (a_n) of elements of an ordered field F is called a *Cauchy sequence* if for each $\varepsilon > 0$ in F there exists a positive integer n_0 such that:

$$|a_n - a_m| < \varepsilon \text{ whenever } m, n \geq n_0.$$

In other words, the terms in a Cauchy sequence get close to each other while the terms of a convergent sequence get close to some fixed element. A convergent sequence is always a Cauchy sequence, but a Cauchy sequence whose elements belong to a field F does not necessarily converge to an element in F . To give an example, let us suppose that F is the set of rational numbers; consider the sequence of integers defined by the *Fibonacci recurrence relation*:

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}, \quad n \geq 2.$$

It can be shown that:

$$F_n = \frac{1}{\sqrt{5}}[\alpha^n - \beta^n] \quad (1.14)$$

where $\alpha = \frac{1 + \sqrt{5}}{2}$, $\beta = \frac{1 - \sqrt{5}}{2}$.

Now define the sequence of rational numbers by:

$$x_n = F_n/F_{n-1}, \quad n \geq 1.$$

We can show that:

$$\lim_{n \rightarrow \infty} x_n = \alpha = \frac{1 + \sqrt{5}}{2} \text{ (the Golden Ratio)}$$

and this limit is not a rational number. The Fibonacci numbers are useful in many kinds of applications, such as optimisation (finding the minimum or maximum of a function) and random number generation.

We define a *complete metric space* X as one in which every Cauchy sequence converges to an element in X . Examples of complete metric spaces are:

- Euclidean space \mathbb{R}^n .
- The metric space $C[a, b]$ of continuous functions on the interval $[a, b]$.
- By definition, *Banach spaces* are complete normed linear spaces. A *normed linear space* has a norm based on a metric, as follows $d(x, y) = \|x - y\|$.
- $L^p(0, 1)$ is the Banach space of functions $f : [0, 1] \rightarrow \mathbb{R}$ defined by the norm $\|f\|_p = \left(\int_0^1 |f(x)|^p dx\right)^{1/p} < \infty$ for $1 \leq p < \infty$.

Definition 1.6 An *open cover* of a set E in a metric space X is a collection $\{G_j\}$ of open subsets of X such that $E \subset \bigcup_j G_j$.

Finally, we say that a subset K of a metric space X is *compact* if every open cover of K contains a *finite subcover*, that is $K \subset \bigcup_{j=1}^N G_j$ for some finite N .

1.6.3 Lipschitz Continuous Functions

We now examine functions that map one metric space into another one. In particular, we discuss the *concepts of continuity* and *Lipschitz continuity*.

It is convenient to discuss these concepts in the context of metric spaces.

Definition 1.7 Let (X, d_1) and (Y, d_2) be two metric spaces. A function f from X into Y is said to be *continuous at the point* $a \in X$ if for each $\varepsilon > 0$ there exists a $\delta > 0$ such that:

$$d_2(f(x), f(a)) < \varepsilon \text{ whenever } d_1(x, a) < \delta.$$

This is a generalisation of the concept of continuity in Section 1.2 (Definition 1.1). We should note that this definition refers to the continuity of a function at a single point. Thus, a function can be continuous at some points and discontinuous at other points.

Definition 1.8 A function f from a metric space (X, d_1) into a metric space (Y, d_2) is said to be a *uniformly continuous* on a set $E \subset X$ if for each $\varepsilon > 0$ there exists a $\delta > 0$ such that:

$$d_2(f(x), f(y)) < \varepsilon \text{ whenever } x, y \in E \text{ and } d_1(x, y) < \delta.$$

If the function f is uniformly continuous, then it is continuous, but the converse is not necessarily true. Uniform continuity holds for all points in the set E , whereas normal continuity is only defined at a single point.

Definition 1.9 Let $f: [a, b] \rightarrow \mathbb{R}$ be a real-valued function and suppose that we can find two constants M and α such that $|f(x) - f(y)| \leq M|x - y|^\alpha, \forall x, y \in [a, b]$. Then we say that f satisfies a *Lipschitz condition* of order α , and we write $f \in Lip(\alpha)$.

We take an example. Let $f(x) = x^2$ on the interval $[a, b]$.

Then:

$$\begin{aligned} |f(x) - f(y)| &= |x^2 - y^2| = |(x + y)(x - y)| \leq (|x| + |y|)|x - y| \\ &\leq M|x - y|, \text{ where } M = 2\max(|a|, |b|). \end{aligned}$$

Hence $f \in Lip(1)$.

A concept related to Lipschitz continuity is called a *contraction*.

Definition 1.10 Let (X, d_1) and (Y, d_2) be metric spaces. A transformation T from X into Y is called a *contraction* if there exists a number $\lambda \in (0, 1)$ such that:

$$d_2(T(x), T(y)) \leq \lambda d_1(x, y) \text{ for all } x, y \in X.$$

In general, a contraction maps a pair of points into another pair of points that are closer together. A contraction is always continuous.

The ability to discover and apply contraction mappings has considerable theoretical and numerical value. For example, it is possible to prove that stochastic differential equations (SDEs) have unique solutions by the application of *fixed point theorems*:

- Brouwer's fixed point theorem
- Kakutani's fixed point theorem
- Banach's fixed point theorem
- Schauder's fixed point theorem

Our interest here lies in the following fixed point theorem.

Theorem 1.2 (Banach Fixed Point Theorem) Let T be a contraction of a complete metric space (X, d) into itself:

$$d(T(x), T(y)) \leq \lambda d(x, y), \quad \lambda \in (0, 1).$$

Then T has a unique fixed-point \bar{x} . Moreover, if x_0 is any point in X and the sequence (x_n) is defined recursively by the formula $x_n = T(x_{n-1})$, $n = 1, 2, \dots$, then $\lim x_n = \bar{x}$ and:

$$d(\bar{x}, x_n) \leq \frac{\lambda}{1 - \lambda} d(x_{n-1}, x_n) \leq \frac{\lambda^n}{1 - \lambda} d(x_0, x_1).$$

In general, we assume that X is a Banach space and that T is a linear or non-linear mapping of X into itself. We then say that x is a *fixed point* of T if $Tx = x$.

1.7 SUMMARY AND CONCLUSIONS

In this chapter we gave an introduction to a number of relevant mathematical concepts from real analysis that are used throughout this book, directly or indirectly. We also have introduced other relevant topics in other chapters. To summarise:

- Chapter 1: Real analysis
- Chapter 4: Finite dimensional vector spaces
- Chapter 5: Numerical linear algebra
- Chapter 16: Complex analysis

In this way we hope that this book becomes more self-contained than otherwise.

Ordinary Differential Equations (ODEs), Part 1

It is better to solve one problem five different ways, than to solve five problems one way.

George Pólya.

2.1 INTRODUCTION AND OBJECTIVES

In this chapter we introduce a class of differential equations in which the highest order derivative is one. Furthermore, these equations have a single independent variable (which in nearly all applications plays the role of time). In short, these are termed *ordinary differential equations* (ODEs) precisely because of the dependence on a single variable.

ODEs crop up in many application areas, such as mechanics, biology, engineering, dynamical systems, economics and finance, to name just a few. It is for this reason that we devote two dedicated chapters to them.

The following topics are discussed in this chapter:

- Motivational examples of ODEs
- Qualitative properties of ODEs
- Common finite difference schemes for initial value problems for ODEs
- Some theoretical foundations.

In Chapter 3 we continue with our discussion of ODEs, including code examples in C++ and Python.

2.2 BACKGROUND AND PROBLEM STATEMENT

In this section we introduce the very first differential equation of this book. It is a *scalar first-order linear ordinary differential equation* (ODE), and we shall analyse it from several qualitative and quantitative viewpoints.

Consider a bounded interval $[0, T]$ where $T > 0$. This interval could represent time or distance, for example. In most cases we shall view this interval as representing time values. In the interval we define the *initial value problem* (IVP) for an ODE:

$$\begin{aligned} Lu = u'(t) + a(t)u(t) &= f(t), t \in [0, T] \text{ with } a(t) \geq \alpha > 0, \forall t \in [0, T] \\ u(0) &= A \end{aligned} \tag{2.1}$$

where L is a first-order linear differential operator involving the derivative with respect to the time variable and $a = a(t)$ is a strictly positive function in $[0, T]$. The term $f(t)$ is called the *inhomogeneous forcing term*, and it is independent of u . Finally, the solution to the IVP must be specified at $t = 0$; this is the so-called *initial condition*.

In general, the problem (2.1) has a unique solution given by:

$$\begin{aligned} u(t) &= I_1(t) + I_2(t) \\ I_1(t) &= A \exp\left(-\int_0^t a(s)ds\right) \\ I_2(t) &= \exp\left(-\int_0^t a(s)ds\right) \int_0^t \exp\left(\int_0^x a(s)ds\right) f(x)dx. \end{aligned} \tag{2.2}$$

(See Hochstadt (1964), where the so-called *integration factor* is used to determine a solution.)

A special case of (2.1) is when the right-hand term $f(t)$ is zero and $a(t)$ is constant; in this case the solution becomes a simple exponential term without any integrals, and this will be used later when we examine difference schemes to determine their feasibility. In particular, a scheme that behaves badly for the above special case will be unsuitable for more general or more complex problems unless some modifications are introduced.

2.2.1 Qualitative Properties of the Solution and Maximum Principle

Before we introduce difference schemes for (2.1), we discuss a number of results that allow us to describe how the solution u behaves. First, we wish to conclude that if the initial value A and inhomogeneous term $f(t)$ are positive, then the solution $u(t)$ should also be positive for any value t in $[0, T]$. This so-called *positivity* or *monotonicity result* should be reflected in our difference schemes (not all schemes possess this property). Second, we wish to know how the solution $u(t)$ grows or decreases as a function of time. The following two results deal with these issues.

Lemma 2.1 (Positivity). Let the operator L be defined in Equation (2.1), and let w be a well-behaved function satisfying the inequalities:

$$Lw(t) \geq 0 \quad \forall t \in [0, T]$$

$$w(0) \geq 0.$$

Then the following result holds true:

$$w(t) \geq 0 \quad \forall t \in [0, T].$$

Roughly speaking, this lemma states that you cannot get a negative solution from positive input.

You can verify it by examining Equation (2.2) because all terms are positive.

The following result gives bounds on the growth of $u(t)$.

Theorem 2.1 Let $u(t)$ be the solution of Equation (2.1). Then:

$$|u(t)| \leq \frac{N}{\alpha} + |A| \quad \forall t \in [0, T]$$

where

$$|f(t)| \leq N \quad \forall t \in [0, T].$$

This result states that the value of the solution is bounded by the input data. In other words, it is a *well-posed problem*.

We wish to replicate these properties in our difference schemes for Equation (2.1).

For completeness, we show the steps to be executed in order to produce the result in Equation (2.2).

$$\text{Let } I(t) = \exp\left(\int_0^t a(s)ds\right), \quad I^{-1}(t) = \exp\left(-\int_0^t a(s)ds\right). \quad (2.3)$$

Then from Equation (2.1) we see:

$$I(t)\left(\frac{du}{dt} + au\right) = I(t)f(t)$$

or:

$$\frac{d}{dt}(I(t)u) = I(t)f(t).$$

Integrating this equation between $t = 0$ and $t = \xi$ gives:

$$\begin{aligned} \int_0^\xi \frac{d}{dt}(I(t)u)dt &= \int_0^\xi I(t)f(t)dt \quad (\text{and using the fact that } I(0) = 1) \\ I(\xi)u(\xi) &= u(0) + \int_0^\xi I(t)f(t)dt \\ u(\xi) &= u(0)I^{-1}(\xi) + I^{-1}(\xi) \int_0^\xi I(t)f(t)dt \\ &= \exp\left(-\int_0^\xi a(s)ds\right)u(0) + \exp\left(-\int_0^\xi a(s)ds\right) \int_0^\xi I(t)f(t)dt. \end{aligned}$$

This style of mathematical analysis will be used in other contexts in this book, for example when transforming convection-diffusion-reaction equations (in particular, the Black–Scholes equation) to *adjoint form*.

2.2.2 Rationale and Generalisations

The IVP Equation (2.1) is a model for all the linear time-dependent differential equations that we encounter in this book. We no longer think in terms of scalar problems in which the functions in Equation (2.1) are scalar-valued, but we can view an ODE at different levels of abstraction. To this end, we focus on the generic *homogeneous ODE* with solution $u(t)$:

$$\frac{du}{dt} = Au, \quad t > 0. \quad (2.4)$$

This equation subsumes several special cases:

1. The variable A is a square matrix, and then Equation (2.4) represents a system of ODEs. This is a very important area of research having many applications in science, engineering, and finance.
2. The variable A is an ordinary or partial differential operator, and then Equation (2.4) represents an ODE in a Hilbert or Banach space.
3. The variable A is a tridiagonal or block tridiagonal matrix that originates from a semi-discretisation in space of a time-dependent partial differential equation (PDE) using the Method of Lines (MOL) as discussed in Chapter 20.
4. The formal solution of (2.4) is:

$$u(t) = u(0)e^{At}, \quad t > 0. \quad (2.5)$$

In other words, we express the solution in terms of the exponential function of a matrix or of a differential operator. In the former case, there are many ways to compute the exponential of a matrix (see Moler and Van Loan (2003)).

5. The solution of Equation (2.4) can be simplified by *matrix* or *operator splitting* of the operator A :

$$\begin{aligned} A &= A_1 + A_2 \\ \frac{du}{dt} &= A_1 u \\ \frac{du}{dt} &= A_2 u. \end{aligned} \quad (2.6)$$

For example, we can split a matrix A into two simpler matrices, or we can split an operator A into its convection and diffusion components. In other words, we solve Equation (2.4) as a sequence of simpler problems in (2.6). These topics will be discussed in Chapters 18, 22, and 23.

6. The initial value problem (2.1) was originally used as a model test of finite difference methods in (Dahlquist (1956)). The resulting results and insights are helpful when dealing more complex IVPs.

Finally, this chapter and Chapter 3 are recommended for readers who may not be familiar with ODE theory and ODE numerics. It is a prerequisite before moving to partial differential equations.

2.3 DISCRETISATION OF INITIAL VALUE PROBLEMS: FUNDAMENTALS

We now discuss finding an approximate solution to Equation (2.1) using the *finite difference method*. We introduce several popular schemes as well as defining standardised notation.

The interval or range where the solution of Equation (2.1) is defined is $[0, T]$. When approximating the solution using finite difference equations, we use a discrete set of points in $[0, T]$ where the discrete solution will be calculated. To this end, we divide $[0, T]$ into N equal intervals of length k , where k is a positive number called the *step size*. (We also use the symbol Δt to denote the step size in many cases.) We number these discrete points as shown in Figure 2.1. In general all coefficients and discrete functions will be defined at these *mesh points* only. We adopt the following notation:

$$\begin{aligned} a^n &= a(t_n), f^n = f(t_n) \\ a^{n,\theta} &= a(\theta t_n + (1 - \theta)t_{n+1}), 0 \leq \theta \leq 1, 0 \leq n \leq N - 1 \\ u^{n,\theta} &= \theta u^n + (1 - \theta)u^{n+1}, 0 \leq n \leq N - 1 \text{ (function to be calculated).} \end{aligned} \quad (2.7)$$

Not only do we have to approximate functions at mesh points, but we also have to come up with a scheme to approximate the derivative appearing in Equation (2.1). There are several possibilities, and they are based on *divided differences*. For example, the following divided differences approximate the first derivative of u at the *mesh point* $t_n = n * k$:

$$\left. \begin{aligned} D_+ u^n &\equiv \frac{u^{n+1} - u^n}{k} \\ D_- u^n &\equiv \frac{u^n - u^{n-1}}{k} \\ D_0 u^n &\equiv \frac{u^{n+1} - u^{n-1}}{2k} \end{aligned} \right\} \quad (2.8)$$

The first two divided differences are called *one-sided differences* and give first-order accuracy to the derivative, while the last divided difference is called a *centred approximation* to the derivative. In fact, by using a Taylor's expansion (assuming sufficient

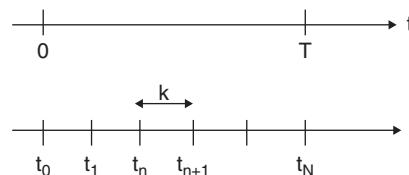


FIGURE 2.1 Continuous and discrete spaces.

smoothness of u), we can prove the following:

$$\begin{cases} |D_{\pm}u(t_n) - u'(t_n)| \leq Mk, & n = 0, 1, \dots \\ |D_0u(t_n) - u'(t_n)| \leq Mk^2, & n = 0, 1, \dots \end{cases} \quad (2.9)$$

Note that the first two approximations use two consecutive mesh points while the last formula uses three consecutive mesh points.

We now decide on how to approximate Equation (2.1) using finite differences. To this end, we need to introduce two new concepts:

- One-step and multistep methods
- Explicit and implicit schemes.

A *one-step method* is a finite difference scheme that calculates the solution at time-level $n + 1$ in terms of the solution at time-level n . No information at levels $n - 1$, $n - 2$, or previous levels is needed in order to calculate the solution at level $n + 1$. A *multistep method*, on the other hand, is a difference scheme where the solution at level $n + 1$ is determined by values at levels n , $n - 1$ and possibly previous time levels. Multistep methods are more complicated than one-step methods, and we concentrate solely on the latter methods in this book.

An *explicit difference scheme* is one where the solution at time $n + 1$ can be calculated from the information at level n directly. No extra arithmetic is needed: for example, using division or matrix inversion. An *implicit finite difference scheme* is one in which the terms involving the approximate solution at level $n + 1$ are grouped together and only then can the solution at this level be found. Obviously, implicit methods are more difficult to program than explicit methods because we must solve a system of equations at each time step.

2.3.1 Common Schemes

We now introduce a number of important and useful difference schemes that approximate the solution of Equation (2.1). These schemes will pop up all over the place in later chapters. Understanding how the schemes work in a simpler context will help you appreciate them when we tackle partial differential equations based on the Black–Scholes model. They also help in our understanding of notation, jargon, and syntax.

The main schemes are:

- Explicit Euler
- Implicit Euler
- Crank–Nicolson (or Box scheme)
- The trapezoidal method.

The *explicit Euler* method is given by:

$$\frac{u^{n+1} - u^n}{k} + a^n u^n = f^n, \quad n = 0, \dots, N - 1 \quad (2.10)$$

$$u^0 = A$$

whereas the *implicit Euler* method is given by:

$$\begin{aligned} \frac{u^{n+1} - u^n}{k} + a^{n+1}u^{n+1} &= f^{n+1}, \quad n = 0, \dots, N-1 \\ u^0 &= A. \end{aligned} \tag{2.11}$$

Notice the difference: in Equation (2.10) the solution at level $n+1$ can be directly calculated in terms of the solution at level n , while in Equation (2.11) we must rearrange terms in order to calculate the solution at level $n+1$.

The next scheme is called the *Crank–Nicolson* or *box scheme*, and it can be seen as an average of explicit and implicit Euler schemes. It is given as (see notation in Equation (2.7)):

$$\begin{aligned} \frac{u^{n+1} - u^n}{k} + a^{n,\frac{1}{2}}u^{n,\frac{1}{2}} &= f^{n,\frac{1}{2}}, \quad n = 0, \dots, N-1 \\ u^0 &= A \\ \text{where} \\ u^{n,\frac{1}{2}} &\equiv \frac{1}{2}(u^n + u^{n+1}). \end{aligned} \tag{2.12}$$

It is useful to know that the three schemes can be merged into one generic scheme as it were by introducing a parameter θ (the scheme is sometimes called the *Theta method*):

$$\begin{aligned} L(k)u^n &\equiv \frac{u^{n+1} - u^n}{k} + a^{n,\theta}u^{n,\theta} = f^{n,\theta} \\ u^{n,\theta} &\equiv \theta u^n + (1 - \theta)u^{n+1}, \quad 0 \leq \theta \leq 1 \\ f^{n,\theta} &\equiv f(\theta t_n + (1 - \theta)t_{n+1}) \end{aligned} \tag{2.13}$$

and the special cases are given by:

$$\begin{aligned} \theta &= 1, \text{ explicit Euler} \\ \theta &= 0, \text{ implicit Euler} \\ \theta &= \frac{1}{2}, \text{ Crank–Nicolson.} \end{aligned} \tag{2.14}$$

The solution of Equation (2.13) is given by:

$$u^{n+1,\theta} \equiv u^{n+1} = \frac{(1 - k\theta a^{n,\theta})u^n + kf^{n,\theta}}{1 + k(1 - \theta)a^{n,\theta}}. \tag{2.15}$$

This equation is useful because it can be mapped to C++ code and will be used by other schemes by defining the appropriate value of the parameter θ .

Finally, the *trapezoidal method* is similar to Crank–Nicolson, but it takes a slightly different averaging mechanism:

$$\frac{u^{n+1} - u^n}{k} + \frac{1}{2}(a^n u_n + a^{n+1} u_{n+1}) = \frac{1}{2}(f^n + f^{n+1}), \quad n = 0, \dots, N-1 \quad (2.16)$$

$$u^0 = A.$$

2.3.2 Discrete Maximum Principle

Having developed some difference schemes, we would like to have a way of determining if the discrete solution is a good approximation to the exact solution in some sense. Although we do not deal with this issue in great detail, we do look at stability and convergence issues.

Definition 2.1 The one-step difference scheme $L(k)$ of the form (2.13) is said to be *positive* if:

$$L(k)w^n \geq 0, \quad n = 0, \dots, N-1, w^0 \geq 0 \quad (2.17)$$

implies that $w^n \geq 0 \forall n = 0, \dots, N$. Here, w^n is a *mesh function* defined at the mesh points t_n .

Based on this definition, we see that the implicit Euler scheme is always positive while the explicit Euler scheme is positive if the term:

$$1 - ka^n \geq 0 \text{ or } k \leq \frac{1}{a^n}, \quad n \geq 0 \quad (2.18)$$

is positive. Thus, if the function $a(t)$ achieves large values (and this happens in practice), we will have to make k very small in order to produce good results. Even worse, if k does not satisfy the constraint in (2.18) then the discrete solution looks nothing like the exact solution, and so-called *spurious oscillations* occur. This phenomenon occurs in other finite difference schemes, and we propose a number of remedies later in this book.

Definition 2.2 A difference scheme is *stable* if its solution is based in much the same way as the solution of the continuous problem (2.1) (see Theorem 2.1), that is:

$$|u^n| \leq \frac{N}{\alpha} + |A|, \quad n \geq 0 \quad (2.19)$$

where:

$$a(t_n) \geq \alpha, \quad n \geq 0, \quad |f(t_n)| \leq N, \quad n \geq 0$$

and:

$$u^0 = A.$$

Based on the fact that a scheme is stable and consistent (see Dahlquist and Björck (1974)), we can state in general that the error between the exact and discrete solutions is bounded by some polynomial power of the step-size k :

$$|u^n - u(t_n)| \leq Mk^p, \quad p = 1, 2, \dots, \quad n \geq 0 \quad (2.20)$$

where M is a constant that is *independent* of k . For example, in the case of schemes 2.10, 2.11 and 2.12 we have:

$$\text{Implicit Euler: } |u^n - u(t_n)| \leq Mk, \quad n = 0, \dots, N$$

$$\text{Crank–Nicolson (Box): } |u^n - u(t_n)| \leq Mk^2, \quad n = 0, \dots, N \quad (2.21)$$

$$\text{Explicit Euler: } |u^n - u(t_n)| \leq Mk, \quad n = 0, \dots, N \quad \text{if } 1 - a^n k > 0.$$

Thus, we see that the Box method is second-order accurate and is better than the implicit Euler scheme, which is only first-order accurate.

2.4 SPECIAL SCHEMES

We introduce *exponentially fitted schemes* that are used for boundary layer problems (for example, convection-dominated PDEs) and the extrapolation method to increase the accuracy of finite difference schemes. We shall see how to apply these techniques to more complex problems in later chapters. We also discuss predictor-corrector methods.

2.4.1 Exponential Fitting

We now introduce a special class of schemes with desirable properties. These are schemes that are suitable for problems with rapidly increasing or decreasing solutions. In the literature these are called *stiff* or *singular perturbation* problems (see Duffy (1980)). We can motivate these schemes in the present context. Let us take the problem (2.1) when $a(t)$ is constant and $f(t)$ is zero. The solution $u(t)$ is given by a special case of (2.2), namely:

$$u(t) = Ae^{-at}. \quad (2.22)$$

If a is large then the derivatives of $u(t)$ tend to increase; in fact, at $t = 0$, the derivatives are given by:

$$\frac{d^k u(0)}{dt^k} = A(-a)^k, \quad k = 0, 1, 2, \dots \quad (2.23)$$

The physical interpretation of this fact is that a boundary layer exists near $t = 0$ where u is changing rapidly, and it has been shown that classical finite difference schemes fail to give acceptable answers when a is large (typically values between 1000 and 10000). We get so-called *spurious oscillations*, and this problem is also encountered when solving one-factor and multifactor Black–Scholes equations using finite difference methods. We have resolved this problem using so-called *exponentially fitted schemes*. We motivate the scheme in the present context, and later chapters describe how to apply it to more complicated cases.

In order to motivate the fitted scheme, consider the case of constant $a(t)$ and $f(t) = 0$. We wish to produce a difference scheme in such a way that the discrete solution is equal to the exact solution at the mesh points for this constant-coefficient case. We introduce a so-called *fitting factor* σ in the new scheme:

$$\begin{cases} \sigma \left(\frac{u^{n+1} - u^n}{k} \right) + a^{n,\theta} u^{n,\theta} = f^{n,\theta}, & n = 0, \dots, N-1, \quad 0 \leq \theta \leq 1 \\ u^0 = A. \end{cases} \quad (2.24)$$

The motivation for finding the fitting factor is to demand that the exact solution of (2.1) (which is known) has the same values as the discrete solution of (2.24) at the mesh points.

Plugging the exact solution (2.22) into (2.24) and doing some simple arithmetic, we get the following representation for the *fitting factor* σ :

$$\sigma = \frac{ak(\theta + (1-\theta)e^{-ak})}{1 - e^{-ak}}. \quad (2.25)$$

Having found the fitting factor for the constant coefficient case, we generalise to a scheme for the case (2.1) as follows:

$$\begin{aligned} \sigma^{n,\theta} \frac{u^{n+1} - u^n}{k} + a^{n,\theta} u^{n,\theta} &= f^{n,\theta}, & n = 0, \dots, N-1, \quad 0 \leq \theta \leq 1 \\ u^0 &= A \\ \sigma^{n,\theta} &= \frac{a^{n,\theta} (\theta + (1-\theta)e^{-a^{n,\theta} k})}{1 - e^{-a^{n,\theta} k}} k. \end{aligned} \quad (2.26)$$

In practice we work with a number of special cases:

$$\left. \begin{aligned} \theta &= 0 && \text{(implicit)} \\ \sigma^{n,0} &= a^{n+1} k / (e^{a^{n+1} k} - 1) \\ \theta &= \frac{1}{2} && \text{(fitted Box)} \\ \theta^{n,\frac{1}{2}} &= \frac{a^{n,\frac{1}{2}} k}{2} \left(\frac{1 + e^{-a^{n,\frac{1}{2}} k}}{1 - e^{-a^{n,\frac{1}{2}} k}} \right) \\ \theta^{n,\frac{1}{2}} &= \frac{a^{n,\frac{1}{2}} k}{2} \coth \frac{a^{n,\frac{1}{2}} k}{2} && \text{(Il'in).} \end{aligned} \right\} \quad (2.27)$$

In the final case $\coth(x)$ is the hyperbolic cotangent function.

In later chapters we shall apply the fitting scheme to the one-factor and multifactor Black–Scholes equations, and we shall show that we get good approximations to the

option price and its delta in all regions of (S, t) space where S is the underlying asset and t is time (up to maturity T). This is in contrast to the Crank–Nicolson scheme where the *spurious oscillations* are seen, especially when the underlying S is near the strike price K or when the payoff function is discontinuous.

2.4.2 Scalar Non-Linear Problems and Predictor-Corrector Method

Real-life problems are very seldom linear. In general, we model applications using non-linear IVPs:

$$\begin{cases} u' \equiv \frac{du}{dt} = f(t, u), & t \in (0, T] \\ u(0) = A. \end{cases} \quad (2.28)$$

Here $f(t, u)$ is a non-linear function in u in general. Of course, Equation (2.28) contains Equation (2.1) as a special case. However, it is not possible to come up with an exact solution for (2.28) in general, and we must resort to some numerical techniques. Approximating (2.28) poses challenges because the resulting difference schemes may also be non-linear, thus forcing us to solve the discrete system at each time level by Newton's method or some other non-linear solver. For example, consider applying the trapezoidal method to (2.28):

$$u_{n+1} = u_n + \frac{k}{2}[f(t_n, u_n) + f(t_{n+1}, u_{n+1})] \quad n = 0, \dots, N - 1 \quad (2.29)$$

where $f(t, u)$ is non-linear. Here see that the unknown term u is on both the left- and right-hand sides of the equation, and hence it is not possible to solve the problem explicitly in the way that we did for the linear case. However, not all is lost, and to this end we introduce the *predictor-corrector method* that consists of a set consisting of two difference schemes; the first equation uses the explicit Euler method to produce an intermediate solution called a predictor that is then used in what could be called a *modified trapezoidal rule*:

$$\begin{aligned} \text{Predictor: } & \bar{u}_{n+1} = u_n + k f(t_n, u_n) \\ \text{Corrector: } & u_{n+1} = u_n + \frac{k}{2}[f(t_n, u_n) + f(t_{n+1}, \bar{u}_{n+1})] \\ \text{or: } & u_{n+1} = u_n + \frac{k}{2}\{f(t_n, u_n) + f(t_{n+1}, u_n + k f(t_n, u_n))\}. \end{aligned} \quad (2.30)$$

The predictor-corrector is used in practice; it can be used with non-linear systems and stochastic differential equations (SDE). We discuss this topic in Chapter 13.

2.4.3 Extrapolation

We give an introduction to a technique that allows us to improve the accuracy of finite difference schemes. This is called *Richardson extrapolation* in general. We take a specific case to show the essence of the method, namely the implicit Euler method (2.11).

We know that it is first-order accurate and that it has good stability properties. We now apply the method on meshes of size k and $k/2$, and we can show that the approximate solutions can be represented as follows:

$$v^k = u + mk + O(k^2)$$

$$v^{k/2} = u + m \frac{k}{2} + O(k^2).$$

Then:

$$w^{k/2} \equiv 2v^{k/2} - v^k = u + O(k^2).$$

Thus, $w^{k/2}$ is a second-order approximation to the solution of (2.1).

The constant m is independent of k , and this is why we can eliminate it in the first equations to get a scheme that is second-order accurate. The same trick can be employed with the second-order Crank–Nicolson scheme to get a fourth-order accurate scheme as follows:

$$v^k = u + mk^2 + O(k^4)$$

$$v^{k/2} = u + m \left(\frac{k}{2} \right)^2 + O(k^4).$$

Then:

$$w^{k/2} \equiv \frac{4}{3}v^{k/2} - \frac{1}{3}v^k = u + O(k^4).$$

In general, with extrapolation methods we state what accuracy we desire, and the algorithm divides the interval $[0, T]$ into smaller subintervals until the difference between the solutions on consecutive meshes is less than a given tolerance.

A thorough introduction to extrapolation techniques for ordinary and partial differential equations (including one-factor and multifactor parabolic equations) can be found in Marchuk and Shaidurov (1983).

2.5 FOUNDATIONS OF DISCRETE TIME APPROXIMATIONS

We discuss the following properties of a finite difference approximation to an ODE:

- Consistency
- Stability
- Convergence.

These topics are also relevant when we discuss numerical methods for partial differential equations.

In order to reduce the scope of the problem (for the moment), we examine the simple scalar non-linear *initial value problem* (IVP) defined by:

$$\begin{cases} \frac{dX}{dt} = \mu(t, X), & 0 < t \leq T \\ X(0) = X_0 & \text{given.} \end{cases} \quad (2.31)$$

We assume that this system has a unique solution in the interval $[0, T]$. In general it is impossible to find an exact solution of Equation (2.31), and we resort to some kind of numerical scheme. To this end, we can write a generic *k-step method* in the form (Henrici (1962), Lambert (1991)):

$$\sum_{j=0}^k (\alpha_j X_{n-j} - \Delta t \beta_j \mu(t_{n-j}, X_{n-j})) = 0, \quad k \leq n \leq N \quad (2.32)$$

where α_j and β_j are constants, $j = 0, \dots, k$, and Δt is the constant step-size.

Since this is a *k*-step method, we need to give *k initial conditions*:

$$X_0; X_1, \dots, X_{k-1}. \quad (2.33)$$

We note that the first initial condition is known from the continuous problem (2.31) while the determination of the other $k - 1$ numerical initial conditions is a part of the numerical problem. These $k - 1$ numerical initial conditions must be chosen with care if we wish to avoid producing unstable schemes. In general, we compute these values by using Taylor's series expansions or by one-step methods.

We discuss *consistency* of scheme (2.32). This is a measure of how well the exact solution of (2.31) satisfies (2.32). Consistency states that the difference equation (2.32) formally converges to the differential equation in (2.31) when Δt tends to zero. In order to determine if a finite difference scheme is consistent, we define the *generating polynomials*:

$$\begin{aligned} \rho(\zeta) &= \sum_{j=0}^k \alpha_j \zeta^{k-j} \\ \sigma(\zeta) &= \sum_{j=0}^k \beta_j \zeta^{k-j}. \end{aligned} \quad (2.34)$$

It can be shown that *consistency* (see Henrici (1962), Dahlquist and Björck (1974)) is equivalent to the following conditions:

$$\rho(1) = 0, \quad \frac{d\rho}{d\zeta}(1) = \sigma(1). \quad (2.35)$$

Let us take the explicit Euler method applied to IVP (2.31):

$$X_n - X_{n-1} = \Delta t \mu(t_n, X_{n-1}), \quad n = 1, \dots, N.$$

The reader can check the following:

$$\begin{aligned} \rho(\zeta) &= \alpha_0 \zeta + \alpha_1 = \zeta - 1 \\ \sigma(\zeta) &= 1 \end{aligned} \quad (2.36)$$

from which we deduce that the explicit Euler scheme is consistent with the IVP (2.31) by checking with Equation (2.35).

The class of difference schemes (2.32) subsumes well-known specific schemes, for example:

- The one-step ($k = 1$) explicit and implicit Euler schemes.
- The two-step ($k = 2$) leapfrog scheme.
- The three-step ($k = 3$) Adams–Bashforth scheme.
- The one-step trapezoidal ($k = 1$) scheme.

Each of these schemes is consistent with the IVP (2.31), as can be checked by calculating their generating polynomials.

We now discuss what is meant by the *stability* of a finite difference scheme. To take a simple counterexample, a scheme whose solution is exponentially increasing or oscillating in time while the exact solution is decreasing in time cannot be stable. In order to define stability, it is common practice to examine *model problems* (whose solutions are known) and apply various finite difference schemes to them. We then examine the stability properties of the schemes. The model problem in this case is the constant-coefficient scalar IVP in which the coefficient μ is a complex number:

$$\begin{cases} \frac{dX}{dt} = \mu X, & 0 < t \leq T \\ X(0) = 1 \end{cases} \quad (2.37)$$

and whose solution is given by:

$$X(t) = e^{\mu t}. \quad (2.38)$$

Thus, the solution is increasing when μ is positive and real and decreasing when μ is negative and real. The corresponding finite difference schemes should have similar properties. We take an example of the *one-step trapezoidal method*:

$$\begin{cases} X_{n+1} - X_n = \frac{\mu \Delta t}{2}(X_{n+1} + X_n), & 0 \leq n \leq N-1 \\ X_0 = 1. \end{cases} \quad (2.39)$$

The solution of Equation (2.39) is given by:

$$X_n = \left[\frac{1+\alpha}{1-\alpha} \right]^n, \quad \alpha \equiv \frac{\mu \Delta t}{2}.$$

We see that X_n is bounded if and only if $|1+\alpha| \leq |1-\alpha|$ and this implies $\operatorname{Re}\left(\frac{\mu \Delta t}{2}\right) \leq 0$ (recall that μ is a complex number) where $\operatorname{Re}(z)$ means ‘real part of z ’.

This example leads us to our first encounter with the concept of stability of finite difference schemes. In particular, we define the *region of absolute stability* of a numerical method for an IVP as the set of complex values of $\mu \Delta t$ for which all discrete approximations to the test problem (2.37) remain bounded when n tends to infinity. For example, for the trapezoidal method (2.39) the left-half plane is the stability region.

Theorem 2.1 (The Root Condition). A necessary and sufficient condition for the stability of a linear multistep method (2.32) is that all the roots of the polynomial ρ

defined by equation (2.34) are located inside or on the unit circle and that the roots of modulus 1 are simple.

We now discuss convergence issues. We say that a difference scheme has *order of accuracy p* if:

$$\max|X_n - X(t_n)| \leq M\Delta t^p, \quad \text{for } 0 \leq n \leq N$$

where X_n = approximate solution of (2.32), $X(t_n)$ = exact solution of (2.31), and M is independent of Δt .

We conclude this section by stating a convergence result that allows us to estimate the error between the exact solution of an initial value problem and the solution of a multistep scheme that approximates it. To this end, we consider the n -dimensional *autonomous* initial value problem:

$$(IVPI) \left\{ \begin{array}{l} y' = \frac{dy}{dt} = f(y), \text{ in the interval } [a, b], \quad y(a) = c \\ \text{where:} \\ y = (y_1, \dots, y_n)^\top \\ f(y) = (f_1(y), \dots, f_n(y))^\top, \quad c = (c_1, \dots, c_n)^\top. \end{array} \right.$$

By *autonomous* we mean that $f(y)$ is a function of the dependent variable y only and is thus not of the form $f(y, t)$. The latter form is called *non-autonomous*.

We approximate this IVP using the multistep method (2.32). We recall:

$$\sum_{j=0}^k (\alpha_j X_{n-j} - \Delta t \beta_j f(X_{n-j})) = 0.$$

Theorem 2.2 Assume that the solution y of the IVPI is $p + 1$ times differentiable with $\|y^{(p+1)}(x)\| \leq K_0$, $p \geq 1$ and assume that f is differentiable for all y .

Suppose furthermore that the sequence $\{X_n\}$ is defined by the equations:

$$X_n = y(t_n) + \epsilon_n, \quad n = 0, \dots, k - 1$$

$$\sum_{j=0}^k (\alpha_j X_{n-j} - \Delta t \beta_j f(X_{n-j})) = \epsilon_n, \quad k \leq n \leq \frac{b-a}{\Delta t}.$$

If the multistep method is stable and satisfies:

$$\sum_{j=1}^k (\alpha_j y(t - j\Delta t) - \Delta t \beta_j y'(t - jh)) \sim C \Delta t^{p+1} y^{p+1}(t) \quad \forall t \in [0, T]$$

where C is a positive constant independent of Δt , then there exist constants K_0, K_1, K_2 and Δt_0 such that for all $t_n \in [a, b]$, $\Delta t \leq \Delta t_0$:

$$\|X_n - y(t_n)\| \leq K_0 \Delta t^p (t_n - a) K_0 + \sum_{j=0}^n \|\epsilon_j\| K_1 e^{K_2(t_n - a)}.$$

In all cases we define the norm $\|.\|$ for a vector $z \in \mathbb{R}^n$ as:

$$\|z\| = \left(\sum_{j=1}^n |z_j|^2 \right)^{1/2} \text{ or } \|z\| = \max_{j=1,\dots,n} |z_j|.$$

We state this theorem in more general terms: consistency and stability of a multistep scheme are sufficient for convergence.

Finally, the discussion in this section is also applicable to systems of ODEs. For more discussions, we recommend Henrici (1962) and Lambert (1991).

Finally, we present four finite difference schemes for the IVP (2.31) and their generating polynomials as defined by Equations (2.34):

Leapfrog Scheme:

$$\frac{X_{n+1} - X_{n-1}}{2\Delta t} = \mu(t_n, X_n), \quad 1 \leq n \leq N-1$$

$$\rho(\zeta) = \zeta^2 - 1, \quad \sigma(\zeta) = 2\zeta.$$

Trapezoidal Scheme:

$$\frac{X_{n+1} - X_n}{\Delta t} = \frac{1}{2}[\mu(t_n, X_n) + \mu(t_{n+1}, X_{n+1})], \quad 0 \leq n \leq N-1$$

$$\rho(\zeta) = \zeta - 1, \quad \sigma(\zeta) = \frac{1}{2}(\zeta + 1).$$

Implicit Euler Scheme:

$$\frac{X_{n+1} - X_n}{\Delta t} = \mu(t_{n+1}, X_{n+1}), \quad 0 \leq n \leq N-1$$

$$\rho(\zeta) = \zeta - 1, \quad \sigma(\zeta) = \zeta.$$

Adams–Bashforth Scheme:

$$\frac{X_{n+1} - X_n}{\Delta t} = \frac{1}{12}[23\mu(t_n, X_n) - 16\mu(t_{n-1}, X_{n-1}) + 5\mu(t_{n-2}, X_{n-2})], \quad 2 \leq n \leq N$$

$$\rho(\zeta) = \zeta - 1$$

$$\sigma(\zeta) = \frac{23}{12}\zeta^2 - \frac{16}{12}\zeta + \frac{5}{12}.$$

We recommend that you verify the results using the forms of the generating polynomials for one-step and two-step methods, respectively. The general forms are:

$$\rho(\zeta) = \alpha_0\zeta + \alpha_1$$

$$\sigma(\zeta) = \beta_0\zeta + \beta_1$$

$$\rho(\zeta) = \alpha_0\zeta^2 + \alpha_1\zeta + \alpha_2$$

$$\sigma(\zeta) = \beta_0\zeta^2 + \beta_1\zeta + \beta_2.$$

2.6 STIFF ODEs

We now discuss special classes of ODEs that arise in practice and whose numerical solution demands special attention. These are called *stiff systems* whose solutions consist of two components; first, the *transient solution* that decays quickly in time, and second, the *steady-state solution* that decays slowly. We speak of *fast transient* and *slow transient*, respectively. As a first example, let us examine the scalar linear initial value problem:

$$\begin{cases} \frac{dy}{dt} + ay = 1, & t \in (0, T], \quad a > 0 \text{ is a constant} \\ y(0) = A \end{cases} \quad (2.40)$$

whose exact solution is given by:

$$y(t) = Ae^{-at} + \frac{1}{a}[1 - e^{-at}] = \left(A - \frac{1}{a}\right)e^{-at} + \frac{1}{a}.$$

In this case the transient solution is the exponential term, and this decays very fast (especially when the constant a is large) for increasing t . The steady-state solution is a constant, and this is the value of the solution when t is infinity. The transient solution is called the *complementary function*, and the steady-state solution is called the *particular integral* (when $\frac{dy}{dt} = 0$), the latter including no arbitrary constant. The stiffness in the above example is caused when the value a is large; in this case traditional finite difference schemes can produce unstable and highly oscillating solutions. One remedy is to define very small time steps. Special finite difference techniques have been developed that remain stable even when the parameter a is large. These are the *exponentially fitted schemes*, and they have a number of variants. The variant described in Liniger and Willoughby (1970) is motivated by finding a fitting factor for a general initial value problem and is chosen in such a way that it produces an exact solution for a certain model problem. To this end, let us examine the scalar ODE:

$$\frac{dy}{dt} = f(t, y(t)), \quad t \in (0, T] \quad (2.41)$$

and let us approximate it using the *Theta method*:

$$y_{n+1} - y_n = \Delta t[(1 - \theta)f_{n+1} + \theta f_n], \quad f_n = f(t_n, y_n) \quad (2.42)$$

where the parameter θ has not yet been specified. We determine it using the heuristic that this so-called *Theta method* should be exact for the *linear constant-coefficient model problem*:

$$\frac{dy}{dt} = \lambda y \quad (\text{exact solution } y(t) = e^{\lambda t}). \quad (2.43)$$

Based on this heuristic and by using the exact solution from (2.43) in scheme (2.42) ($f(t, y) = \lambda y$), we get the value (you should check that this formula is correct; it is a bit

of algebra). We get:

$$y_{n+1} = \frac{1 + \Delta t \lambda}{1 - (1 - \theta) \Delta t \lambda} y_n \quad \text{and} \quad (2.44)$$

$$\theta = -\frac{1}{\Delta t \lambda} - \frac{\exp(\Delta t \lambda)}{1 - \exp(\Delta t \lambda)}.$$

Note: this is a different kind of exponential fitting.

We need to determine if this scheme is stable (in some sense). To answer this question, we introduce some concepts.

Definition 2.3 The *region of (absolute) stability* of a numerical method for an initial value problem is the set of complex values $\Delta t \lambda$ for which all discrete solutions of the model problem (2.43) remain bounded when n approaches infinity.

Definition 2.4 A numerical method is said to be *A-stable* if its region of stability is the left-half plane, that is:

$$R = \{\Delta t \lambda \in \mathbb{C}; \operatorname{Re} \Delta t \lambda < 0\}.$$

Returning to the exponentially fitted method, we can check that it is A-stable because for all $\Delta t \lambda < 0$ we have $\theta \leq \frac{1}{2}$, and this condition can be checked using the scheme (2.42) for the model problem (2.43).

We can generalise the exponential fitting technique to linear and non-linear systems of equations. In the case of a linear system, stiffness is caused by an isolated real negative eigenvalue of the matrix A in the equation:

$$\frac{dy}{dt} = Ay + f(t) \quad (2.45)$$

where $y, f \in \mathbb{R}^n$ and A is a constant $n \times n$ matrix with eigenvalues $\lambda_j \in \mathbb{C}$ and eigenvectors $x_j \in \mathbb{C}^n$, $j = 1, \dots, n$.

The solution of Equation (2.45) is given by:

$$y(t) = \sum_{j=1}^n c_j \exp(\lambda_j t) x_j + g(t)$$

where $c_j, j = 1, \dots, n$ are arbitrary constants and $g(t)$ is a particular integral. In this case we can employ exponential fitting by fitting the dominant eigenvalues which can be computed by the *Power method*, for example.

If we assume that the real parts of the eigenvalues are less than zero, we can conclude that the solution tends to the steady-state. Even though this solution is well-behaved the cause of numerical instabilities is the presence of quickly decaying transient components of the solution caused by the dominant eigenvalues of the matrix A in (2.45).

Let us take an example whose matrix has already been given in diagonal form:

$$\frac{d}{dt} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} -\alpha_1 & 0 \\ 0 & -\alpha_2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

$$y_1(0) = y_2(0) = 1, \quad 0 \leq \alpha_2 \ll \alpha_1.$$

The solution of this system is given by:

$$y_1(t) = e^{-\alpha_1 t}$$

$$y_2(t) = e^{-\alpha_2 t}.$$

The solutions decay at different rates, and in the case of the explicit Euler method the inequality:

$$\Delta t \leq \frac{2}{\alpha_1}$$

must be satisfied if the first component of the solution is to remain within the region of absolute stability. Unfortunately, choosing a time step of these proportions will be too small to allow for control over the round-off error in the second component.

In this case we fit the dominant eigenvalue. For variable coefficient systems and non-linear systems, we periodically compute the *Jacobian matrix* and carry out fitting on it.

The presence of different time scales in ODEs leads to a number of challenges when approximating them using the standard finite difference schemes. In particular, schemes such as explicit and implicit Euler, Crank–Nicolson, and predictor–corrector do not approximate these systems well, unless a prohibitively small time step is used. Let us take the example (Dahlquist and Björck (1974)):

$$\frac{dy}{dt} = 100(\sin t - y), \quad y(0) = 0$$

with exact solution:

$$y(t) = \frac{\sin t - 0.01 \cos t + 0.01e^{-100t}}{1.0001}.$$

This is a stiff problem because of the different time scales in the solution. We carried out an experiment using the explicit Euler method, and we had to divide the interval $[0, 3]$ into 1.2 million subintervals in order to achieve accuracy to three decimal places. The implicit Euler and Crank–Nicolson methods are not much better.

Robust ODE solvers for stiff system using the Boost C++ library *odeint* are discussed in Duffy (2018).

2.7 INTERMEZZO: EXPLICIT SOLUTIONS

A special case of an initial value problem is when the number of dimensions n in an initial value problem is equal to 1. In this case we speak of a scalar problem, and it is

useful to study these problems if one wishes to get some insights into how finite difference methods work. In this section we discuss some numerical properties of one-step finite difference schemes for the linear scalar problem:

$$\begin{aligned} Lu &\equiv \frac{du}{dt} + a(t)u = f(t), \quad 0 < t < T \\ u(0) &= u_0 \end{aligned}$$

where $a(t) \geq \alpha > 0, \forall t \in [0, T]$.

The reader can check that the one-step methods (Equations (2.10), (2.11) and (2.12)) can all be cast as the general form *recurrence relation*:

$$U^{n+1} = A_n U^n + B_n, \quad n \geq 0,$$

where $A_n = A(t_n)$, $B_n = B(t_n)$. Then, using this formula and mathematical induction we can give an explicit solution at any time level as follows:

$$U^n = \left(\prod_{j=0}^{n-1} A_j \right) U_0 + \sum_{v=0}^{n-1} B_v \prod_{j=v+1}^{n-1} A_j, \quad n \geq 1$$

with:

$$\prod_{j=I}^{j=J} g_j \equiv 1 \text{ if } I > J$$

for a mesh function g_j . A special case is when the coefficients A_n and B_n are constant ($A_n = A, B_n = B$), that is:

$$U^{n+1} = AU^n + B, \quad n \geq 0.$$

Then the general solution is given by:

$$U^n = A^n U_0 + B \frac{1 - A^n}{1 - A}, \quad n \geq 0$$

where we note that $A^n \equiv n^{\text{th}}$ power of constant A and $A \neq 1$.

In order to prove this, we need the formula for the sum of a series:

$$1 + A + \dots + A^n = \frac{1 - A^{n+1}}{1 - A}, \quad A \neq 1.$$

For a readable introduction to difference schemes, we refer the reader to Goldberg (1986).

2.8 SUMMARY AND CONCLUSIONS

In this chapter we gave an introduction to scalar ODEs and systems of ODEs. The main goal was to help the reader become acquainted with their mathematical and numerical foundations as well as become familiar with the associated notation. We recommend learning the main concepts in this chapter because many of them will be used and needed when we model one-factor and two-factor convection-diffusion-reaction PDEs such as the Black–Scholes equation, for example.

Contrary to popular thinking, there is more to ODEs than trying to find analytical solutions for them. Very few ODEs have analytical solutions, and we must resort to ODE solvers.

Ordinary Differential Equations (ODEs), Part 2

The source of all great mathematics is the special case, the concrete example. It is frequent in mathematics that every instance of a concept of seemingly great generality is in essence the same as a small and concrete special case.

Paul Halmos

3.1 INTRODUCTION AND OBJECTIVES

In Chapter 2 we discussed both systems of ODEs and scalar ODEs. The focus was mainly concerned with notation, the structure of ODEs and finite difference schemes to approximate them. We implicitly assumed that the solution of the corresponding initial value problem existed in an otherwise unspecified time interval and that the solution was unique. These assumptions constitute a huge leap of faith. In this chapter we discuss existence and uniqueness results for ODEs and stochastic differential equation (SDEs). We also introduce several important numerical schemes and code in C++ and Python.

3.2 EXISTENCE AND UNIQUENESS RESULTS

We turn our attention to a more general *initial value problem* for a *non-linear system* of ODEs:

$$\begin{cases} y' = f(t, y), & t \in \mathbb{R} \\ y(0) = A \end{cases} \quad (3.1)$$

where:

$$y : \mathbb{R} \rightarrow \mathbb{R}^n, \quad A \in \mathbb{R}^n, \quad f : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$$

and:

$$f(t, y) = (f_1(t, y), \dots, f_n(t, y))^\top \text{ where } f_j : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}, \quad j = 1, \dots, n.$$

Some of the important questions to be answered are:

- Does System (3.1) have a unique solution?
- In which interval (t_0, t_1) , $t_0 < t_1$ does this solution exist?
- What is the asymptotic behaviour of the solution as $t \rightarrow \infty$?

To this end, let B be a region of $n + 1$ dimensional space, and let f be continuously differentiable with respect to t and with respect to all the components of y at all points of B . We assume that the following inequalities hold:

$$\left| \frac{\partial f}{\partial y_j}(t, y) \right| \leq K \text{ for some } K > 0 \quad j = 1, \dots, n \quad (3.2)$$

and:

$$|f(t, y)| \leq M \text{ for some } M > 0. \quad (3.3)$$

Theorem 3.1 Let f and $\frac{\partial f}{\partial y_j}$ ($j = 1, \dots, n$) be continuous in the box $B = \{(t, y) : |t - t_0| \leq a, |y - \eta| \leq b\}$ where a and b are positive numbers and satisfying the bounds (3.2) and (3.3) for (t, y) in B . Let α be the smaller of the numbers a and b/M and define the successive approximations:

$$\begin{aligned} \phi_0(t) &= \eta \\ \phi_n(t) &= \eta + \int_{t_0}^t f(s, \phi_{n-1}(s)) ds, \quad n \geq 1. \end{aligned} \quad (3.4)$$

Then the sequence $\{\phi_n\}$ of successive approximations ($n \geq 0$) converges (uniformly) in the interval $|t - t_0| \leq \alpha$ to a solution $\phi(t)$ of (3.1) that satisfies the initial condition $\phi(t_0) = \eta$.

Method (3.4) is called the *Picard iterative method* and it is used to prove the existence of the solution of systems of ODE (3.1). It is mainly of theoretical value, as it should not necessarily be seen as a practical way to construct a numerical solution. However, it does give us insights into the qualitative properties of the solution. On the other hand, it is a useful exercise to construct the sequence of iterates in Equation (3.4) for some simple cases.

We note that the IVP (3.1) can be written as an *integral equation* as follows:

$$y(t) = y_0 + \int_{t_0}^t f(s, y(s)) ds \quad (3.5)$$

where $y_0 = A = y(t_0)$.

It can be proved that the solution of (3.1) is also the solution of (3.5) and vice versa. We see then that Picard iteration is based on (3.5) and that we wish to have the iterates converging to a solution of (3.5).

We note that the Picard method is used primarily to prove the existence of solutions and it is not a numerical method as such.

3.2.1 An Example

We take a simple autonomous non-linear scalar ODE to show how to calculate Picard iterates:

$$y' = f(y) = y^2, \quad y(t_0) = a \quad (3.6)$$

whose solution is given by:

$$y(t) = \frac{a}{1 - a(t - t_0)}.$$

We now compute the *Picard iterates* (3.4) for this ODE in order to determine the values of t for which the ODE has a solution. For convenience, let us take $a = 1, t_0 = 0$. Some simple integration shows that:

$$\begin{aligned} \phi_1(t) &= 1 \\ \phi_1(t) &= 1 + \int_0^t f(\phi_0)dt = 1 + t \\ \phi_2(t) &= 1 + \int_0^t f(\phi_1)dt = 1 + t + t^2 + t^3/3 \\ \phi_3(t) &= 1 + t + t^2 + t^3 + \frac{2t^4}{3} + \frac{t^5}{3} + \frac{t^6}{9} + \frac{t^7}{63}. \end{aligned} \quad (3.7)$$

We can see that the series is beginning to look like $\frac{1}{1-t} = \sum_{j=0}^{\infty} t^j$. We know that this series is convergent for $|t| < 1$. A nice exercise is to compute the Picard iterates in the most general case (that is, $a \neq 1, t_0 \neq 0$) and to determine under which circumstances the ODE (3.6) has a solution. In this case we have represented the solution of an ODE as a series, and we then analysed this series for which there are many convergence results, such as the *root test* and the *ratio test*.

3.3 OTHER MODEL EXAMPLES

We take some model ODEs for motivation.

3.3.1 Bernoulli ODE

The Bernoulli ODE is named after Jacob Bernoulli. It is special in the sense that it is a non-linear equation having an exact solution:

$$y' + P(x)y = Q(x)y^n \quad (n \neq 0, n \neq 1). \quad (3.8)$$

In the cases $n = 0$ and $n = 1$ and when P and Q are constants, this equation is linear, and an exact solution can then be found. We reduce Equation (3.8) to a linear one by the substitution $v = y^{1-n}$ ($v' = (1 - n)y^{-n}y'$) to produce the following linear ODE:

$$v' + (1 - n)Pv = (1 - n)Q. \quad (3.9)$$

3.3.2 Riccati ODE

The Riccati ODE is a non-linear ODE of the form:

$$y' = P(x) + Q(x)y + R(x)y^2 + N(x, y). \quad (3.10)$$

This ODE has many applications, for example to interest-rate models (Duffie and Kan (1996)). In some cases a closed-form solution to Equation (3.10) is possible, but in this book our focus is on approximating it using the finite difference method.

We now discuss the relationship between the Riccati equation and the pricing of a *zero-coupon bond* $P(t, T)$, which is a contract that offers one dollar at maturity T . By definition, an *affine term structure* model assumes that $P(t, T)$ has the form:

$$P(t, T) = \exp[A(t, T) - B(t, T)r(t)].$$

Let us assume that the short-term interest rate is described by the following stochastic differential equation (SDE):

$$dr = \mu(t, r)dt + \sigma(t, r)dW_t$$

where W_t is a standard Brownian motion under the risk-neutral equivalent measure and μ and σ are given functions.

Duffie and Kan proved that $P(t, T)$ is *exponential-affine* if and only if the drift μ and volatility σ have the form:

$$\mu(t, r) = \alpha(t)r + \beta(t), \quad \sigma(t, r) = \sqrt{\gamma(t)r + \delta(t)}$$

where $\alpha(t), \beta(t), \gamma(t)$ and $\delta(t)$ are given functions of t .

The coefficients $A(t, T)$ and $B(t, T)$ in this case are determined by the following ordinary differential equations:

$$\frac{dB}{dt} = \frac{\gamma(t)}{2}B(t, T)^2 - \alpha(t)B(t, T) - 1, \quad B(T, T) = 0 \quad (3.11)$$

and:

$$\frac{dA}{dt} = \beta(t)B(t, T) - \frac{\delta(t)}{2}B(t, T)^2, \quad A(T, T) = 0. \quad (3.12)$$

The first Equation (3.11) for $B(t, T)$ is the Riccati equation and the second one (3.12) is solved easily from the first one by integration.

3.3.3 Predator-Prey Models

ODEs can be used as simple models of population growth, for example, by assuming that the rate of reproduction of a population of size P is proportional to the existing population and to the amount of available resources. The ODE is:

$$\frac{dP}{dt} = rP \left(1 - \frac{P}{K}\right), \quad P(0) = P_0$$

where r is the *growth rate* and K is the *carrying capacity*. The initial population is P_0 . It is easy to check the following identities:

$$P(t) = \frac{KP_0 e^{rt}}{K + P_0(e^{rt} - 1)} \text{ and } \lim_{t \rightarrow \infty} P(t) = K.$$

Transformation of this equation leads to the logistic ODE:

$$\frac{dn}{d\tau} = n(1 - n) \tag{3.13}$$

where n is the population in units of carrying capacity ($n = P/K$) and τ measures time in units of $1/r$.

For systems, we can consider the *predator-prey* model in an environment consisting of foxes and rabbits:

$$\begin{aligned} \frac{dr(t)}{dt} &= -ar(t)f(t) + br(t) \\ \frac{df(t)}{dt} &= -pf(t) + qf(t)r(t) \end{aligned} \tag{3.14}$$

where:

- $r(t)$ = number of rabbits at time t
- $f(t)$ = number of foxes at time t
- $br(t)$ = birth rate of rabbits
- $-ar(t)f(t)$ = death rate of rabbits
- b = unit birth rate of rabbits
- $-pf(t)$ = death rate of foxes
- $qf(t)r(t)$ = birth rate of foxes
- q = unit birth rate of foxes.

The ODE system (3.14) is a model of a closed ecological environment in which foxes and rabbits are the only kinds of animals. Rabbits eat grass (of which there is a constant supply), procreate and are eaten by foxes. All foxes eat rabbits, procreate and die of geriatric diseases.

System (3.14) is sometimes called the *Lotka–Volterra* equations, which are an example of a more general *Kolmogorov model* to model the dynamics of ecological systems with predator-prey interactions, competition, disease and mutualism (Lotka (1956)).

3.3.4 Logistic Function

A *logistic function* (or *logistic curve*) is an S-shaped sigmoid curve defined by the equation:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}, \quad (3.15)$$

where

x_0 = x -value of sigmoid's midpoint

L = curve's maximum value

k = steepness of the curve.

A special case is when $k = 1, x_0 = 0, L = 1$, resulting in the *standard logistic function* defined by the equation:

$$f(x) = \frac{1}{1 + e^{-x}}.$$

We can verify from this equation that the logistic function satisfies the *non-linear initial* value problem:

$$\frac{df}{dx} = f(1 - f), \quad f(0) = \frac{1}{2}. \quad (3.16)$$

The logistic function models processes in a range of fields such as artificial neural networks (learning algorithms, where it is called an *activation function*), economics, probability and statistics, to name a few.

3.4 EXISTENCE THEOREMS FOR STOCHASTIC DIFFERENTIAL EQUATIONS (SDEs)

A *random process* is a family of random variables defined on some probability space and indexed by the parameter t where t belongs to some index set. A random process is a function of two variables:

$$\{\xi(t, x) : t \in T, x \in S\}$$

where T is the index set and S is the *sample space*. For a fixed value of t , the random process becomes a random variable, while for a fixed sample point x in S the random process is a real-valued function of t called a *sample function* or a *realisation* of the process. It is also sometimes called a *path*.

The index set T is called the *parameter set*, and the values assumed by $\xi(t, \omega)$ are called the *states*; finally, the set of all possible values is called the *state space* of the random process.

The index set T can be discrete or continuous; if T is discrete, then the process is called a *discrete-parameter* or *discrete-time process* (also known as a *random sequence*). If T is continuous, then we say that the random process is called *continuous-parameter* or *continuous-time*. We can also consider the situation where the state is discrete or continuous. We then say that the random process is called *discrete-state (chain)* or *continuous-state*, respectively.

3.4.1 Stochastic Differential Equations (SDEs)

We give a short introduction to stochastic differential equations (SDEs) as they are closely related to ODEs. We discuss SDEs in more detail in Chapter 13.

We introduce the scalar random processes described by SDEs of the form:

$$d\xi(t) = \mu(t, \xi(t))dt + \sigma(t, \xi(t)) dW(t) \quad (3.17)$$

where:

$\xi(t) \equiv$ random process

$\mu(t, \xi(t)) \equiv$ transition (drift) coefficient

$\sigma(t, \xi(t)) \equiv$ diffusion coefficient

$W(t) \equiv$ Brownian process

$\xi(0) \equiv$ given initial condition

defined in the interval $[0, T]$. We assume for the moment that the process takes values on the real line. We know that this SDE can be written in the equivalent integral form:

$$\xi(t) = \xi(0) + \int_0^t \mu(s, \xi(s))ds + \int_0^t \sigma(s, \xi(s))dW(s). \quad (3.18)$$

This is a non-linear equation, because the unknown random process appears on both sides of the equation and it cannot be expressed in a closed form. We know that the second integral:

$$\int_0^t \sigma(s, \xi(s))dW(s)$$

is a continuous process (with probability 1) provided $\sigma(s, \xi(s))$ is a bounded process. In particular, we restrict the scope to those functions for which:

$$\sup_{|x| \leq C} (|\mu(s, x)| + |\sigma(s, x)|) < \infty, \quad t \in (0, T] \text{ and for every } C > 0.$$

Using this fact, we shall see that the solution of Equation (3.17) is bounded and continuous with probability 1.

We now discuss existence and uniqueness theorems. First, we define some conditions on the coefficients in Equation (3.17):

- C1: $\exists K > 0$ such that $\mu(s, x)^2 + \sigma(s, x)^2 \leq K(1 + x^2)$.
- C2: $\forall s > 0, \exists L$ such that $|\mu(s, x) - \mu(s, y)| + |\sigma(s, x) - \sigma(s, y)| \leq L|x - y|$.
- C3: $\mu(s, x)$ and $\sigma(s, x)$ are defined and measurable with respect to their variables where $s \in (0, T]$, $x \in (-\infty, \infty)$.
- C4: $\mu(s, x)$ and $\sigma(s, x)$ are continuous with respect to their variables for $t \in (0, T]$, $x \in (-\infty, \infty)$.

Condition C2 is called a *Lipschitz condition* in the second variable, while condition C1 constrains the growth of the coefficients in Equation (3.17). We assume throughout that the random variable $\xi(0)$ is independent of $W(t)$.

Theorem 3.2 Assume conditions C1, C2 and C3 hold. Then Equation (3.17) has a unique continuous solution with probability 1 for any initial condition $\xi(0)$.

Theorem 3.3 Assume that conditions C1 and C4 hold. Then the Equation (3.17) has a continuous solution with probability 1 for any initial condition $\xi(0)$.

We note the difference between the two theorems: the condition C2 is what makes the solution unique. Finally, both theorems assume that $\xi(0)$ is independent of the Brownian motion $W(t)$.

We now define another condition on the diffusion coefficient in Equation (3.17).

C5: $\sigma(t, x) > 0$ and for every $C > 0$ there exists an $L > 0$ and $\alpha > \frac{1}{2}$ such that:

$$|\sigma(t, x) - \sigma(t, y)| \leq L|x - y|^\alpha \text{ for } |x| \leq C, |y| \leq C.$$

Theorem 3.4 Assume conditions C4, C1 and C5 hold. Then the Equation (3.17) has a continuous solution with probability 1 for any initial condition $\xi(0)$.

For proofs of these theorems, see Skorokhod (1982), for example.

In some cases it is possible to find a closed-form solution of Equation (3.17) (or equivalently, Equation (3.18)). When the drift and diffusion coefficients are constant, we see that the exact solution is given by the formula:

$$\xi(t) = \xi(0) \exp((\mu - \frac{1}{2}\sigma^2)t + \sigma W(t)). \quad (3.19)$$

Knowing the exact solution is useful, because we can test the accuracy of finite difference schemes against it, and this gives us some insights into how well these schemes work for a range of parameter values.

It is useful to know how the solution of Equation (3.17) behaves for large values of time; the answer depends on a relationship between the drift and diffusion parameters:

- i) $\mu > \frac{1}{2}\sigma^2$, $\lim_{t \rightarrow \infty} \xi(t) = \infty$ almost surely
- ii) $\mu < \frac{1}{2}\sigma^2$, $\lim_{t \rightarrow \infty} \xi(t) = 0$ almost surely
- iii) $\mu = \frac{1}{2}\sigma^2$, $\xi(t)$ fluctuates between arbitrary large and arbitrary small values as $t \rightarrow \infty$ almost surely.

In general it is not possible to find an exact solution, and in these cases we must resort to numerical approximation techniques.

Equation (3.17) is a *one-factor equation* because there is only one dependent variable (namely $\xi(t)$) to be modelled. It is possible to define equations with several dependent variables. The prototypical non-linear stochastic differential equation is given by the system:

$$dX = \mu dt + \sigma dW(t) \quad (3.20)$$

where:

$$\begin{aligned}\mu &: [0, T] \times \mathbb{R}^n \rightarrow \mathbb{R}^n \text{ (vector)} \\ \sigma &: [0, T] \times \mathbb{R}^n \rightarrow \mathbb{R}^{n \times m} \text{ (matrix)} \\ X &: [0, T] \rightarrow \mathbb{R}^n \text{ (vector)} \\ W &: [0, T] \rightarrow \mathbb{R}^m \text{ (vector).}\end{aligned}$$

In general the drift and diffusion terms in (3.20) are non-linear:

$$\begin{aligned}\mu &= \mu(t, X) \\ \sigma &= \sigma(t, X).\end{aligned}$$

This is a generalisation of Equation (3.17). Thus, instead of scalars this system employs vectors for the solution, drift and random number terms while the diffusion term is a rectangular matrix.

Existence and uniqueness theorems for the solution of the SDE system (3.20) are similar to those in the one-factor case. For example, theorem 5.2.1 in Øksendal (1998) addresses these issues. We discuss SDEs in more detail in Chapter 13.

3.5 NUMERICAL METHODS FOR ODEs

In this section we introduce a class of one-step methods to approximate the solution of ODE system (3.1).

The first step is to replace continuous time by discrete time. To this end, we divide the interval $[0, T]$ into a number of subintervals. We define $N + 1$ *mesh points* as follows:

$$0 = t_0 < t_1 < \dots < t_n < t_{n+1} < \dots < t_N = T.$$

In this case we define a set of *subintervals* (t_n, t_{n+1}) of size $\Delta t_n \equiv t_{n+1} - t_n$, $0 \leq n \leq N - 1$.

In general, we speak of a *non-uniform mesh* when the sizes of the subintervals are not necessarily the same. However, in this book we consider in the main a class of finite difference schemes where the N subintervals have the same length (we then speak of a *uniform mesh*), namely $\Delta t = T/N$. The variable $h = T/N$ is also used to denote the uniform *mesh size*.

In general, we define y_n to be the approximate solution at time t_n and we write the functional dependence of y_{n+1} on t_n, y_n and h by:

$$y_{n+1} - y_n = h\Phi(t_n, y_n, h) \tag{3.21}$$

where Φ is called the *increment function*. For example, in the case of the *explicit Euler method*, this function is:

$$\Phi(t, y, h) = f(t, y).$$

The increment function represents the increment of the approximate solution. In general, the goal is to produce a formula for Φ that agrees with a certain exact relative

increment with an error of $O(h^p)$ where $p > 1$ without making it necessary to compute the derivative of f (Henrici (1962)). A special case of (3.21) is the *fourth-order Runge–Kutta method*:

$$\Phi(t, y, h) = \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4] \quad (3.22)$$

where:

$$\begin{aligned} k_1 &= f(t, y) \\ k_2 &= f(t + h/2, y + \frac{1}{2}hk_1) \\ k_3 &= f(t + h/2, y + \frac{1}{2}hk_2) \\ k_4 &= f(t + h, y + hk_3). \end{aligned}$$

Other methods are:

Second-order *Ralston method*:

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{4}(k_1 + 3k_2) \\ k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + 2h/3, y_n + 2k_1/3). \end{aligned} \quad (3.23)$$

and *Heun’s (improved Euler)* method:

$$\begin{aligned} \tilde{y}_{n+1} &= y_n + hf(t_n, y_n) \\ y_{n+1} &= y_n + \frac{h}{2}[f(t_n, y_n) + f(t_{n+1}, \tilde{y}_{n+1})]. \end{aligned} \quad (3.24)$$

This is a *predictor–corrector method* that also has applications to stochastic differential equations.

In general, explicit Runge–Kutta methods are unsuitable for stiff ODEs.

3.5.1 Code Samples in Python

We show hand-crafted code for the explicit Euler method as well as the schemes (3.22), (3.23) and (3.24). The main reason is to get hands-on experience with coding solvers for ODEs before using production solvers (for example, Python’s or Boost C++ *odeint* libraries), especially for readers for whom numerical ODEs are new, for example readers with an economics or econometrics background. A good strategy is 1) get it working (write your own code and examples), then 2) get it right (use a library with the same examples) and then and only then get it optimised (use a library in a larger application).

The following code is for the methods:

- Explicit Euler
- Fourth-order Runge–Kutta (RK4)
- Second-order Ralston.

- Heun (improved Euler)
- Predictor-corrector method.

We now present the code for a scalar IVP for these methods:

```
# y' = f(x,y) Explicit Euler method, scalar
def Euler(f, x, y, T, N, TOL):
    h = (T - x) / N

    for n in range(1,N+1):
        ynP1 = y + h*f(x,y)
        # Go to next level
        x += h
        y = ynP1

    return ynP1

# y' = f(x,y) Explicit Euler method, system
def Euler2(f, x, y0, T, N, TOL):
    h = (T - x) / N
    m = len(y0)
    ynP1 = y0

    for n in range(N):
        for j in range(m):
            ynP1[j] = y0[j] + h*f[j](x,y0)

        # Go to next level
        x += h
        y0 = ynP1

    return ynP1

def RK4(f, x, y, T, N, TOL):
    # Order 4 Runge Kutta method
    h = (T - x) / N
    k1 = 0; k2 = 0; k3 = 0; k4 = 0
    ynP1 = 0 # y(n+1)

    for n in range(N):
        k1 = f(x, y); k2 = f(x + h / 2, y + h*k1 / 2)
        k3 = f(x + h / 2, y + h*k2 / 2); k4 = f(x + h, y + h*k3);

        ynP1 = y + h*(k1 + 2*k2 + 2*k3 + k4)/6
        # Go to next level
        x += h; y = ynP1

    return ynP1

# https://en.wikipedia.org/wiki/Heun%27s_method
def Heun(f, x, y, T, N, TOL):
```

```

# Modified Euler method (improved polygon method), Collatz 1960

h = (T - x) / N

# Variables
k1 = 0; k2 = 0

for n in range(N):
    ynP1 = y + h*f(x+h/2, y + h*f(x,y)/2)

    # Go to next level
    x += h; y = ynP1

return ynP1

def Ralston2(f, x, y, T, N, TOL):

    h = (T - x) / N

    # Variables
    k1 = 0; k2=0
    ynP1=0 # y(n+1)

    for n in range(N):
        k1 = h*f(x, y); k2 = h*f(x + 2*h / 3, y + 2*k1 / 3)

        ynP1 = y + (k1 + 3 * k2 ) / 4
        # Go to next level
        x += h; y = ynP1

    return ynP1;

def PredictorCorrector(f, x, y, T, N, TOL):
    # PC with fixed step size

    h = (T - x) / N

    for n in range(1,N+1):
        yCorr2 = y + h*f(x, y) # 1. Predictor (explicit Euler)

        # Produce y(n+1) at level x(n+1) from level n, Inner iteration
        diff = 2 * TOL
        while (diff > TOL):
            yCorr = y + 0.5*h*(f(x, y) + f(x + h, yCorr2))
            diff = math.fabs(yCorr - yCorr2) / math.fabs(yCorr)
            yCorr2 = yCorr

        # Stuff has converged at level n+1, now update next level
        x += h
        y = yCorr

    return yCorr

```

A sample test program is:

```
# TestFDMSchemes.py
#
# A catalogue of hand-crafted numerical methods for solving ODEs
#
# (C) Datasim Education BV 2021
#
# Testing

import math
import numpy as np
import FDMSchemes as fdm

def func(t,y):
    return y*(1.0 - y) # logistic ode

t = 0; y = 0.5 #initial time and value
T = 4.0          #end of integration interval
N = 4000         #number of divisions of [0,T]
TOL = 1.0e-5     #for some method, a measure of the desired accuracy

# y' = f(x,y) Explicit Euler method
value = fdm.Euler(func,t,y,T,N,TOL) #scalar
print(value)

value = fdm.PredictorCorrector(func,t,y,T,N,TOL) #scalar
print(value)

value = fdm.RK4(func,t,y,T,N,TOL) #scalar
print(value)

y = 0.5
value = fdm.Heun(func,t,y,T,N,TOL) #scalar
print(value)

value = fdm.Ralston2(func,t,y,T,N,TOL) #scalar
print(value)
```

No doubt the code can be modified to make it more “pythonic” (whatever that means) and reduce the number of lines of code (at the possible expense of readability) if that is a requirement. We have also written these algorithms in C++11 as discussed in Duffy (2018).

3.6 THE RICCATI EQUATION

We return to the non-linear IVP (3.10) that we solve numerically using the explicit Euler method, Richardson extrapolation, and several robust ODE solvers from the Boost C++

library *odeint*. We expect the explicit Euler scheme to perform badly because it is explicit and Equation (3.10) is non-linear. We describe the simple software design for the Euler and extrapolation schemes (the main objective is to show how to map numerical algorithms to code). First, we create a class that models the Riccati Equation (3.10):

```
using value_type = double;
using FunctionType = std::function<value_type (value_type)>;
using FunctionType2 = std::function<value_type (value_type x, value_type y)>;

class GeneralisedRiccati
{
private:
    FunctionType p, q, r;
    FunctionType2 n;
public:
    GeneralisedRiccati(const FunctionType& P, const FunctionType& Q,
        const FunctionType& R, const FunctionType2& N)
        : p(P), q(Q), r(R), n(N)
    {
    }

    double operator() (double x, double y)
    { // Function object (functor)

        return p(x) + q(x)*y + r(x)*y*y + n(x, y);
    }
};
```

We can instantiate this class by calling its constructor, but we prefer to use a *factory method* (this is a *design pattern*) for added flexibility. Notice that we also use C++ *smart pointers* in order to avoid possible memory issues):

```
std::shared_ptr<GeneralisedRiccati> CreateRiccati()
{ // Factory method, specific case

    const double P = 0.0;      const double Q = -10.8;
    const double R = 0.5 * 2.44 * 2.44;
    const double eta = 0.96;   const double lambda = 0.13; double A = 0.42;

    // Generalised Riccati
    auto p = [=](double x) {return P;}; auto q = [=](double x) {return Q;};
    auto r = [=](double x) {return R;};
    auto n = [=](double x, double y) { return (lambda*y) / (eta - y); };
    return std::shared_ptr<GeneralisedRiccati> (new GeneralisedRiccati (p, q, r, n));
}
```

3.6.1 Finite Difference Schemes

We have written some simple code to implement the explicit Euler and Richardson extrapolation methods. We present it mainly for pedagogical reasons. First, we define global arrays that hold the discrete values of the three solutions (two Euler methods on two meshes and the second-order extrapolated scheme):

```
// Global arrays to hold (t, y) data for meshes of size dt and dt/2
// t and y arrays on mesh dt
std::vector<double> tValues;
std::vector<double> yValues;

// t and y arrays on mesh dt/2
std::vector<double> t2Values;
std::vector<double> y2Values;

// 2nd order extrapolated values on mesh dt
std::vector<double> yRichValues;
```

The code for the schemes is:

```
void CurveEuler()
{ // db/dt = F(b), b(0) given
  // b(n+1) = b(n) + dt * F(b(n))

  tValues = CreateMesh(L, U, NT);

  yValues = std::vector<double>(NT+1);
  yValues[0] = A;

  auto riccatiEquation = CreateRiccati();
  double yOld;
  double dt = (U - L) / static_cast<double>(NT);

  for (std::size_t j = 1; j < tValues.size(); ++j)
  {
    yOld = yValues[j-1];
    yValues[j] = yOld + dt*(riccatiEquation)(tValues[j-1], yOld);
  }

}

void CurveRichardson()
{ // O(dt) -> O(dt^2)
  // Exercise: remove duplicate code

  double dt = (U - L)/ double(NT);

  tValues = CreateMesh(L, U, NT);

  yValues = std::vector<double>(NT+1);
  yValues[0] = A;
```

```

auto riccatiEquation = CreateRiccati();
double yOld;
for (std::size_t j = 1; j < tValues.size(); ++j)
{
    yOld = yValues[j-1];
    yValues[j] = yOld + dt*(*riccatiEquation)(tValues[j - 1], yOld);
}

// dt/2
dt *= 0.5;

t2Values = CreateMesh(L, U, 2*NT+1);
y2Values = std::vector<double>(2*NT+1);
y2Values[0] = A;

for (std::size_t j = 1; j < t2Values.size(); ++j)
{
    yOld = y2Values[j-1];
    y2Values[j]
        = yOld + dt*(*riccatiEquation)(t2Values[j - 1], yOld);
}

yRichValues = std::vector<double>(NT+1);
yRichValues[0] = A;

for (std::size_t j = 1; j < yRichValues.size(); ++j)
{
    yRichValues[j] = 2.0* y2Values[2*j] - yValues[j];
}

}

```

Some test code is:

```

int main()
{
    std::cout << "How many steps (need many e.g. 300000?): ";
    std::cin >> NT;

    CurveEuler();
    CurveRichardson();

    std::cout << "Value at T = " << U << " is " << std::setprecision(7)
        << yValues[yValues.size() - 1] << ", press the ANY key " << std::endl;
    std::cout << "Richardson at T = " << U << " is "
        << RichValues[yRichValues.size() - 1] << ", press the ANY key ";

    // Beautiful Excel output
    ExcelDriver xl; xl.MakeVisible(true);
}

```

```

x1.CreateChart(tValues, yValues, std::string("Euler for Riccati"));
x1.CreateChart
(tValues, yRichValues, std::string("Richardson for Riccati"));

    return 0;
}

```

What is the accuracy of these schemes, and how much computer effort (size of NT and the number of function calls) is needed in order to achieve a given accuracy? To this end, we take two cases whose exact solutions have been computed using Boost C++ *odeint*:

- Case I: $T = 1$, exact value = 1.207e-5.
- Case II: $T = 0.2$, exact value = 0.0559.

In general, the library needs approximately 100 function evaluations to reach this level of accuracy. The results for explicit Euler and the extrapolated scheme for cases I and II for $NT = 300, 500, 1000$ and 5000 are:

- Case I: (9.2746e-6, 1.113e-5), (1.0015e-5, 1.118e-5), (1.059e-5, 1.200e-5), (1.083e-5, 1.1206e-5).
- Case II: (0.0554, 0.0559), (0.0556, 0.0559), (0.0558, 0.0559), (0.05589, 0.0559).

For a more complete discussion of Boost *odeint*, see Duffy (2018).

We complete our discussion of finite difference methods for (3.10). The full problem is (3.11); (3.12) is a coupled system of equations, and it can be solved in different ways. The most effective approach at this stage is to use a production third-party ODE solver in C++.

3.7 MATRIX DIFFERENTIAL EQUATIONS

We have seen what scalar and vector ODEs are. Now we consider matrix ODEs.

The Boost *odeint* library can be used to solve matrix ODEs, for example:

$$\frac{dY}{dt} = AY + YB \quad (3.25)$$

where A, B and Y are $n \times n$ matrices.

It can be shown that the solution of system (3.25) can be written as (see Brauer and Nohel (1969)):

$$Y(t) = e^{At}Ce^{Bt}, \text{ where } Y(0) = C, \text{ and } C \text{ is a given matrix.} \quad (3.26)$$

A special case is when:

$B = 0$ (zero matrix)

$C = I$ (identity matrix) in which case we have the solution which is the

exponential of a matrix:

$$Y(t) = e^{At}. \quad (3.28)$$

The conclusion is that we can now compute the exponential of a matrix by solving a matrix ODE. We now discuss this topic using C++. To this end, we examine the system:

$$\begin{cases} \frac{dY}{dt} = AY \\ Y(0) = C. \end{cases} \quad (3.29)$$

where C is a given matrix.

We model this system by the following C++ class:

```
namespace ublas = boost::numeric::ublas;

using value_type = double;
using state_type = boost::numeric::ublas::matrix<value_type>;

class MatrixOde
{
private:
    // dB/dt = A*B, B(0) = C;
    ublas::matrix<value_type> A_;
    ublas::matrix<value_type> C_;
public:
    MatrixOde(const ublas::matrix<value_type>& A,
               const ublas::matrix<value_type>& IC)
        : A_(A), C_(IC) {}

    void operator()(const state_type &x, state_type &dxdt,
                    double t) const
    {
        for( std::size_t i=0 ; i < x.size1(); ++i )
        {
            for( std::size_t j=0 ; j < x.size2(); ++j )
            {
                dxdt(i, j) = 0.0;
                for( std::size_t k = 0; k < x.size2(); ++k )
                {
                    dxdt(i, j) += A_(i,k)*x(k,j);
                }
            }
        }
    };
};
```

There are many dubious ways to compute the exponential of a matrix, see Moler and Van Loan (2003), one of which involves the application of ODE solvers. Other methods include:

- S1: Series methods (for example, truncating the infinite Taylor series representation for the exponential).
- S2: Padé rational approximant. This entails approximating the exponential by a special kind of rational function.
- S3: Polynomial methods using the Cayley–Hamilton method.
- S4: Inverse Laplace transform.
- S5: Matrix decomposition methods.
- S6: Splitting methods.

3.7.1 Transition Rate Matrices and Continuous Time Markov Chains

An interesting application of matrices and matrix ODEs is to the modelling of *credit rating* applications (Wilmott (2006), vol. 2, pp. 665–73). To this end, we define a so-called *transition matrix* P , which is a table whose elements are probabilities representing migrations from one credit rating to another credit rating. For example, a company having a B rating has a probability 0.07 of getting a BB rating in a small period of time. More generally, we are interested in *continuous-time transitions* between states using *Markov chains*, and we have the following *Kolmogorov forward equation*:

$$\frac{dP(t, t')}{dt} = P(t, t')Q \quad (3.30)$$

where t = current time t' = future time and $P(t, t) = I \equiv$ unit matrix unit matrix and $Q = (q_{ij})$, $1 \leq i, j \leq n$ is the *transition rate matrix* having the following properties:

1. $0 \leq -q_{ii} < \infty$
2. $0 \leq -q_{ij}, i \neq j$
3. $\sum_j q_{ij} = 0$ (Row sums) $\forall i = 1, \dots, n$.

The *Kolmogorov backward equation* is:

$$\frac{dP(t, t')}{dt} = -QP(t, t'). \quad (3.31)$$

The objective is to compute the transition rate matrix Q (that is, states that are in one-to-one correspondence with the integers).

In the case of countable space, the *Kolmogorov forward equation* is:

$$\frac{\partial P_{ij}}{\partial t}(s; t) = \sum_k P_{ik}(s; t)Q_{kj}(t) \quad (3.32)$$

where $Q(t)$ is the *transition rate matrix* (also known as *generator matrix*), while the *Kolmogorov backward equation* is:

$$\frac{\partial P_{ij}}{\partial t}(s; t) = \sum_k Q_{ik}(s)P_{kj}(s; t). \quad (3.33)$$

3.8 SUMMARY AND CONCLUSIONS

This chapter took over where Chapter 2 left off. We have tried to give a self-contained overview of the analytic properties of scalar ODEs and systems of ODEs as well as their numerical approximation. The topics are important in their own right, and an understanding of them is important in finance applications. We also gave a short introduction to stochastic differential equations (SDEs) in Section 3.4. We discuss SDEs and their relationship with PDEs in Chapter 13.

An Introduction to Finite Dimensional Vector Spaces

There's no sense in being precise when you don't even know what you're talking about.

John von Neumann

4.1 SHORT INTRODUCTION AND OBJECTIVES

This chapter introduces vector spaces of finite dimension. They can be seen as the n -dimensional generalisation of the two- and three-dimensional vectors that we have become accustomed to. In three dimensions, for example, a vector is a 3-tuple $x = (x_1, x_2, x_3) \equiv (x, y, z)$ consisting of three components (elements), and it can be visualised as a directed line from the point $(0, 0, 0)$ to the point (x_1, x_2, x_3) . In higher dimensions this geometric analogy is lost (unless you are Albert Einstein), and we model vectors as an n -tuple (x_1, x_2, \dots, x_n) of homogeneous components of a certain type (in most cases real or complex variables). In particular, we discuss the following use cases as we progress:

- U1: Addition of vectors.
- U2: Premultiplication of a vector by a scalar (an element of a *field*).
- U3: Inner products in vector spaces.
- U4: Vector space norms.
- U5: The distance between two vectors in some norm.

These abstract properties are applicable to a wide range of data structures that we use in numerical analysis and its (many) applications. They can be specialised to cater for specific data structures such as vectors, matrices and tensors (three-dimensional matrices). For this reason we introduce the reader to *vector space theory*. Studying it will pay dividends in the rest of this book and beyond in the years to come. For example,

vector spaces of finite dimension are generalised to infinite-dimensional Hilbert, Banach and Sobolev spaces, a discussion of which is outside the scope of this book.

4.1.1 Notation

This chapter attempts to present a self-contained and focused introduction to the essential concepts and methods for vector spaces of finite dimension. The applications are numerous, for example numerical linear algebra, finite Markov chains, multivariate optimisation, machine and statistical learning, graph theory and finite difference methods, to name just a few. To this end, we summarise the most important syntax and notation that we use throughout the book:

F or K : a field

\mathbb{R}, \mathbb{C} : set of real and complex numbers, respectively

α, β, a, b : scalars

x, y, z : elements of a vector space

V, W : vector spaces

A, B, M : matrices

$\|\cdot\|$: norm in a vector space

x^\top, A^\top : transpose of a vector, matrix

A^{-1} : inverse of a matrix

λ : eigenvalue of a matrix

$d(x, y)$: distance between vectors x and y

(x, y) : inner product of vectors x and y

$\mathbb{R}^n, \mathbb{C}^n$: n -dimensional real and complex spaces, respectively

$\mathbb{R}^{m,n}$: set of real matrices with m rows and n columns

$V(K)$: a vector space V over a field K (see also $W(K)$)

$\dim V$: dimension of a vector space

$T : V(K) \rightarrow W(K)$: linear transformation between two vector spaces $V(K)$ and $W(K)$ over the same field K

$L(V; W)$: the set of linear transformations from vector space V to vector space W

$N(T)$: null space (kernel) of a linear transformation whose dimension is $n(T)$

$r(T)$: the dimension of the range TV of a linear transformation $T : V \rightarrow W$.

We use the following important syntax to denote matrices:

$$A = (a_{ij}), 1 \leq i \leq m, 1 \leq j \leq n : \text{matrix with } m \text{ rows and } n \text{ columns.} \quad (4.1)$$

These symbols are used in definitions, theorems and algorithms. A good way to learn is to take (simpler) concrete examples before moving to more complex cases and applications.

In time, you should get to the stage whereby you can understand the above notation without having to think twice about it. The same remark holds for all the other notation in this book. It's half the battle!

4.2 WHAT IS A VECTOR SPACE?

From Wikipedia:

In mathematics, a field is a set in which addition, subtraction, multiplication, and division are defined and these operators behave as the corresponding operations on rational and real numbers do. A field is thus a fundamental algebraic structure which is widely used in algebra, number theory, and many other areas of mathematics.

The best known fields are the field of rational numbers, the field of real numbers and the field of complex numbers.

In general, we use the symbol K to denote a generic field, but in most cases we work with real numbers as the underlying field in vector spaces. In some cases, complex numbers are used.

A *vector space V over a field K* (denoted by $V(K)$) is a collection of objects (called *vectors*) together with operations of vector addition and multiplication by elements of K (called *scalars*) satisfying the following axioms for addition of vectors:

- A1: $(x + y) + z = x + (y + z)$, $(x, y, z \in V(K))$
- A2: $x + y = y + x$
- A3: Exists unique 0 in V such that $0 + x = x + 0 = x$ (4.2)
- A4: For each x in V there exists a unique y such that $x + y = 0$ (*the negative of x*), called $-x$.

Axiom A1 states that addition is *associative*, and axiom A2 states that addition is *commutative*. The element 0 is called the *zero element* of the vector space.

Scalar multiplication is defined by the axioms ($a, b \in K$ and $x, y \in V$):

- B1: $a(x + y) = ax + ay$
- B2: $(a + b)x = ax + bx$ (4.3)
- B3: $(ab)x = a(bx)$
- B4: $1x = x$ (1 is the unit element).

From these axioms we see that *subtraction of vectors* is possible because $x - y = x + (-1.y) = x + (-y)$.

The prototypical examples of vector spaces are n -dimensional vectors and rectangular matrices over a field K :

$$\begin{aligned} x &= (x_1, \dots, x_n) \quad (x_j \in K, j = 1, \dots, n) \\ y &= (y_1, \dots, y_n) \quad (y_j \in K, j = 1, \dots, n) \\ x, y &\in K^n \\ (x + y) &= (x_1 + y_1, \dots, x_n + y_n) \\ \lambda x &= (\lambda x_1, \dots, \lambda x_n), \quad \lambda \in K. \end{aligned} \tag{4.4}$$

For matrices:

$$\begin{aligned} m_1 &= (a_{ij}), \quad m_2 = (b_{ij}), \quad 1 \leq i \leq m, \quad 1 \leq j \leq n \\ m_1 + m_2 &= (a_{ij} + b_{ij}) \\ \lambda m_1 &= (\lambda a_{ij}), \quad \lambda \in K. \end{aligned} \tag{4.5}$$

We now define an important non-negative real-valued function on a vector space V called a *norm*. It has the following properties:

$$\begin{aligned} \|x\| &\geq 0; \quad \|x\| = 0 \text{ iff } x = 0 \text{ (if and only if } x = 0) \\ \|\lambda x\| &= |\lambda| \|x\| \\ \|x_1 + x_2\| &\leq \|x_1\| + \|x_2\| \quad (x_1, x_2 \in V, \quad \lambda \in K). \end{aligned} \tag{4.6}$$

Some examples of norms for two-dimensional vectors are:

$$\begin{aligned} x &= (x_1, x_2), x \in \mathbb{R}^2 \\ \|x\| &= \sqrt{x_1^2 + x_2^2} \\ \|x\| &= \max(|x_1|, |x_2|). \end{aligned}$$

The following norms for vectors and matrices are used in applications:

$$\begin{aligned} \text{Euclidean } (l_2) \text{ norm} \quad \|x\|_2 &= \left(\sum_{j=1}^n x_j^2 \right)^{\frac{1}{2}} \\ l_1 \text{ norm} \quad \|x\|_1 &= \sum_{j=1}^n |x_j| \\ l_\infty \text{ norm} \quad \|x\|_\infty &= \max_{1 \leq j \leq n} |x_j|. \end{aligned} \tag{4.7}$$

$$\begin{aligned} L_1 \text{ norm: } \|A\|_1 &= \max_{1 \leq j \leq n} \left(\sum_{i=1}^n |a_{ij}| \right) \\ L_\infty \text{ norm: } \|A\|_\infty &= \max_{1 \leq i \leq n} \left(\sum_{j=1}^n |a_{ij}| \right). \end{aligned} \quad (4.8)$$

Whereas the norm is a measure of the size of a vector, it is also possible to find the distance between two vectors. A natural way to proceed is, given a set X to define a real-valued function on $X \times X$ called a *metric* that satisfies for $x, y, z \in X$:

- D1: $d(x, y) \geq 0; d(x, y) = 0 \iff x = y$
- D2: $d(x, y) = d(y, x)$
- D3: $d(x, y) \leq d(x, z) + d(z, y)$ (*triangle inequality*).

A space X endowed with a metric d is called a *metric space* and is denoted by (X, d) .

Examples of metrics are:

- 1.** $d(x, y) = \|x - y\|, (x, y \in X)$
- 2.** Let X be a non-empty set

$$d(x, y) = \begin{cases} 0, & \text{if } x = y \\ 1, & \text{if } x \neq y. \end{cases} \quad (4.9)$$

- 3.** Let X be a set and let $L^p(X)$ be the set of p -integrable Lebesgue functions on X . If $f, g \in L^p(X)$, then a metric is:

$$d(f, g) = \|f - g\| = \left(\int_X |f(x) - g(x)|^p dx \right)^{1/p} \quad (1 < p < \infty).$$

Norms and metrics are important quantities when proving convergence results in functional and numerical analysis applications.

4.3 SUBSPACES

A non-empty subset X of a vector space $V(K)$ is called a *vector subspace* of $V(K)$ if X forms a vector space over K with the same addition and scalar multiplication as in $V(K)$. For example, let P be the set of polynomials in X with real coefficients, and let polynomial addition and multiplication by real numbers be defined by:

$$\begin{aligned} p(x) &= \sum a_j x^j, \quad q(x) = \sum b_j x^j \\ p(x) + q(x) &= \sum (a_j + b_j) x^j \\ \alpha p(x) &= \sum (\alpha a_j) x^j, \quad \alpha \in \mathbb{R}. \end{aligned} \quad (4.10)$$

Now let P_n be the set of consisting of all polynomials of at most degree $n - 1$ of the form:

$$p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1} = \sum_{j=0}^{n-1} a_j x^j, \quad a_j \in \mathbb{R}, j = 0, \dots, n-1. \quad (4.11)$$

Then P_n is a subspace of P , and it is also a subspace of P_m , $m \geq n$.

We say that a subset X of a vector space $V(K)$ is said to be *closed under addition* if whenever $x_1, x_2 \in X$, then $x_1 + x_2 \in X$. A subset X of a vector space $V(K)$ is said to be *closed under scalar multiplication* if whenever $\lambda \in K$ and $x \in X$ then $\lambda x \in X$.

Theorem 4.1 A subset X of a vector space $V(K)$ is a subspace if and only if:

$$\lambda_1 x_1 + \lambda_2 x_2 \in X \text{ for } \lambda_1, \lambda_2 \in K, \quad x_1, x_2 \in X. \quad (4.12)$$

An exercise: let x_1, \dots, x_r be any r elements of a vector space $V(K)$. Prove that the set U of all elements of $V(K)$ that can be written in the form $\sum_{j=1}^r \lambda_j x_j, \lambda_j \in K, j = 1, \dots, r$ forms a subspace of $V(K)$.

We give an example of a subset X of K^2 defined by:

$$\begin{aligned} X &= \{x = (t, t^2); t \in K\} \\ x_1 + x_2 &= (t_1, t_1^2) + (t_2, t_2^2) = (t_1 + t_2, (t_1^2 + t_2^2)) \\ \lambda x &= (\lambda t, \lambda^2 t^2). \end{aligned} \quad (4.13)$$

It is easily verified that X is a vector space over K , but X is not a subspace of K^2 because:

$$\begin{aligned} x_1, x_2 &\in X \\ \lambda(x_1 + x_2) &\neq \lambda x_1 + \lambda x_2 \end{aligned}$$

and these two quantities are thus not the same!

4.4 LINEAR INDEPENDENCE AND BASES

We are now interested in finding a *minimal subspace* U of independent vectors containing a set X (U contains X) such that any vector in X can be written as a linear combination of these vectors. In this case we say that X spans U . We are particularly interested in the case $X = V(K)$, that is subsets of $V(K)$ that span $V(K)$ itself. Such subsets always exist; for example $X = V(K)$ has this property. We take an example in n -dimensional space. The vectors:

$$\begin{aligned} e_1 &= (1, 0, \dots, 0) \\ e_2 &= (0, 1, \dots, 0) \\ &\vdots \\ e_n &= (0, 0, \dots, 1) \end{aligned} \quad (4.14)$$

span K^n because each element can be written as a linear combination:

$$\lambda_1 e_1 + \dots + \lambda_n e_n.$$

Furthermore, any proper subset of $\{e_1, \dots, e_n\}$ cannot span K^n .

Another example is the vector space generated by polynomials of the form of Equation (4.11) generated the monomials $\{1, t, t^2, \dots, t^{n-1}\}$. It is clear that any proper subset of this set spans a proper subspace of P_n ; it cannot span P_n itself.

Definition 4.1 A vector $x \in V(K)$ is *linearly dependent* on a given subset X of $V(K)$ if x belongs to the subspace generated by the set X .

Definition 4.2 A subset X of $V(K)$ is called a *linearly dependent set* if it contains at least one element that is linearly dependent on the others.

Definition 4.3 A set is called *linearly independent* if it is not linearly dependent.

The elements $\{e_1, \dots, e_n\}$ as defined in Equation (4.14) form a linearly independent set in K^n , and adjoining any other vector to this set makes it linearly dependent.

Summarising, the criterion for linear independence is:

$$\begin{aligned} \text{The set } x = \{x_1, \dots, x_n\} \text{ is linearly independent if } \lambda_1 x_1 + \dots + \lambda_n x_n = 0 \\ \text{implies } \lambda_1 = \lambda_2 = \dots = \lambda_n = 0. \end{aligned} \tag{4.15}$$

Definition 4.4 A *basis* of a vector space $V(K)$ is any linearly independent subset of $V(K)$ which has the property that it spans $V(K)$.

Definition 4.5 The *dimension n* of $V(K)$ (denoted by $\dim V$) is the supremum of the (cardinal) numbers of elements in the linearly independent subsets of $V(K)$.

$V(K)$ is said to be finite-dimensional if n is finite. In this case there exist linearly independent subsets with n elements but no linearly independent subsets with $(n+1)$ elements.

4.5 LINEAR TRANSFORMATIONS

Mappings between vector spaces are at least as interesting as vector spaces themselves. An important property of linear transformations is that they map linearly dependent subsets into linearly dependent subsets. An interesting remark is that the set of all linear transformations between two given vector spaces is itself a vector space.

The mapping:

$$T : V(K) \rightarrow W(K)$$

is called a *linear transformation* from $V(K)$ to $W(K)$ if:

$$\begin{aligned} L1 : T(x_1 + x_2) &= Tx_1 + Tx_2, \quad \forall x_1, x_2 \in V(K) \\ L2 : T(\alpha x) &= \alpha Tx, \quad \forall \alpha \in K, \forall x \in V(K). \end{aligned} \tag{4.16}$$

We see immediately that the zero element in $V(K)$ is mapped to the zero element in $W(K)$.

Some examples of linear transformations are:

$$Tx = 0 \quad \forall x \in V(K) \text{ (zero transformation).}$$

$$Tx = x \quad \forall x \in V(K) \text{ (identity transformation).}$$

$$V = K^3, W = K^4$$

$$T(x_1, x_2, x_3) = (x_1 + x_2, x_3, x_3, 0).$$

A more general linear transformation (in fact, a *vector-valued transformation*) is:

$$T : K^n \rightarrow K^m$$

$$T(x_1, \dots, x_n) = (f_1(x_1, \dots, x_n), \dots, f_m(x_1, \dots, x_n)).$$

Another example of a mapping is $T : V(K) \rightarrow V(K)$, $Tx = \lambda x$, $\lambda \in K$ fixed, and this is called a *magnification* or a *dilation*.

Theorem 4.2 For any given $r \geq 2$:

$$T(\lambda_1 x_1 + \dots + \lambda_r x_r) = \lambda_1 T x_1 + \dots + \lambda_r T x_r$$

$$\lambda_j \in K, j = 1, \dots, r, x_j \in V(K), j = 1, \dots, r$$

is a necessary and sufficient condition for a mapping to be a linear transformation. From this result we can conclude that a linear transformation $T : V(K) \rightarrow W(K)$ is determined by the images $T\varphi_1, \dots, T\varphi_n$ in $W(K)$ of a basis $\{\varphi_1, \dots, \varphi_n\}$ in $V(K)$.

4.5.1 Invariant Subspaces

We adopt the notation $L(V; W)$ to denote the set of linear transformations from the vector space V to the vector space W .

Definition 4.6 Let $T \in L(V; V)$ and let U be a subspace of V such that $TU \subset U$ where $TU = \{Tx; x \in U\}$. Then U is called an *invariant subspace* of V under T , or more briefly, U is T -*invariant*.

We take some examples. Each subspace is invariant with respect to the following operators:

$$T_1, T_2, T_3 \in L(V; V)$$

$$T_1 x = 0 \quad \forall x \in V \text{ (zero operator)}$$

$$T_2 x = x \quad \forall x \in V \text{ (identity operator)}$$

$$T_3 x = \lambda x \quad \forall x \in V, \lambda \in K \text{ (fixed) (similarity operator).}$$

Let $\{e_1, \dots, e_n\}$ be a basis in K^n and suppose that:

$$x = \sum_{j=1}^n \xi_j e_j$$

We define the vector ($m < n$):

$$Px = \sum_{j=1}^m \xi_j e_j.$$

Then P is a linear operator (called the *projection operator*) on to the subspace K^m spanned by the vectors $\{e_1, \dots, e_m\}$.

The projection operator has the following invariant subspaces:

- $K' = \left\{ x = \sum_{j=1}^m \xi_j e_j \right\}$ which remain unchanged and
- $K'' = \left\{ y = \sum_{j=m+1}^n \xi_j e_j \right\}$ that are carried into zero.

4.5.2 Rank and Nullity

Definition 4.7 Let $T \in L(V; W)$. The *rank* of T is the dimension of the range TV of T . It is denoted by $r(T)$.

Definition 4.8 Let $T \in L(V; W)$. The *null space* (or *kernel*) of T is the set of vectors x such that $T(x) = 0$. It is denoted by $N(T)$.

Definition 4.9 The dimension of the subspace $N(T)$ is called the *nullity* and is denoted by $n(T)$.

For example: the *zero operator* ω has $r(\omega) = 0$, $n(\omega) = \dim V$ and the *identity operator* i has $r(i) = \dim V$, $n(i) = 0$.

Theorem 4.3 Let $T \in L(V; W)$. Then $\dim V = r(T) + n(T)$.

We are now interested in determining in how far two vector spaces are ‘similar’ in some sense.

Theorem 4.4 Let $T \in L(V; W)$. Then

T is onto W (surjective) if and only if $r(T) = \dim W$.

T is one-to-one (injective) if and only if $n(T) = 0$.

Definition 4.10 A linear transformation $T \in L(V; W)$ is called an *isomorphism* if it is both injective and surjective.

It is possible to form the sum of linear transformations and to compose linear transformations, and we discuss this topic in Chapter 5.

Eigenvalues (Characteristic Roots) and Eigenvectors (Characteristic Vectors)

Let $T \in L(V; V)$. Then a scalar $\lambda \in K$ is called an *eigenvalue* of L if there exists a non-zero $x \in V(K)$ (called an *eigenvector*) such that:

$$Tx = \lambda x. \quad (4.17)$$

Given a non-zero λ , the set of all x satisfying (4.17) forms a subspace of $V(K)$. The set of all eigenvectors of T corresponding to the same eigenvalue, together with the zero vector, is called an *eigenspace* (or the characteristic space) of T associated with that eigenvalue.

Eigenvalues and eigenvectors are important in numerical linear algebra.

4.6 SUMMARY AND CONCLUSIONS

We have given a precise and compact introduction to finite-dimensional vector spaces and linear transformations between vector spaces. We introduce the notation and jargon associated with the topic, and it forms the basis for many applications. In particular, it clears the way for a study of matrix theory and numerical linear algebra.

We recommend Shilov (1977) as an elegant introduction to linear algebra.

Guide to Matrix Theory and Numerical Linear Algebra

If you can't solve a problem, then there is an easier problem you can solve: find it.

Georg Polya

5.1 INTRODUCTION AND OBJECTIVES

The main goal of this chapter is to introduce matrices: what they are and how to create and use them, as well as classifying matrices based on some of their intrinsic and computed properties. This is not a book on matrix theory, but we think that it is important to introduce matrices upfront and not to relegate them to a two-page appendix at the end of the book. We prefer to inform the reader of the prerequisites in the first part of the book rather than at the end when all the other chapters have been discussed.

We continue with this topic in Chapter 6 when we discuss the role of matrices in numerical linear algebra and their integration with finite difference schemes for ordinary differential equations.

5.2 FROM VECTOR SPACES TO MATRICES

We continue with the topics in Chapter 4 and show how matrices are representations of linear operators.

5.2.1 Sums and Scalar Products of Linear Transformations

We discuss two binary operators on the set $L(V; W)$ where V and W are vector spaces, namely the sum of two linear transformations and multiplication of a linear transformation by a scalar. Each operator produces a new linear transformation.

Definition 5.1 The *sum* of two linear transformations $\alpha, \beta \in L(V; W)$ is a mapping from V to W and is defined by:

$$(\alpha + \beta)(x) = \alpha(x) + \beta(x) \quad \forall x \in V. \quad (5.1)$$

Definition 5.2 The *scalar product* of a linear transformation $\alpha \in L(V; W)$ and a scalar $\lambda \in K$ is defined by:

$$(\lambda\alpha)(x) = \lambda\alpha(x) \quad \forall x \in V. \quad (5.2)$$

Definition 5.3 Let $\alpha \in L(V; U)$, $\beta \in L(W; V)$. Then the *composition* of α and β is defined by:

$$\begin{aligned} (\alpha \cdot \beta)(x) &= \alpha(\beta(x)) \quad \forall x \in W \\ \gamma &\equiv \alpha \cdot \beta. \end{aligned} \quad (5.3)$$

We check that the composition is a linear transformation as follows:

$$\begin{aligned} (\alpha \cdot \beta)(\lambda_1 x_1 + \lambda_2 x_2) &= \alpha(\beta(\lambda_1 x_1 + \lambda_2 x_2)) \\ &= \alpha(\lambda_1 \beta x_1 + \lambda_2 \beta x_2) = \lambda_1 \alpha \beta x_1 + \lambda_2 \alpha \beta x_2 = \lambda_1 (\alpha \cdot \beta) x_1 + \lambda_2 (\alpha \cdot \beta) x_2 \end{aligned}$$

Thus $\gamma(\lambda_1 x_1 + \lambda_2 x_2) = \lambda_1 \gamma x_1 + \lambda_2 \gamma x_2$
for $\lambda_1, \lambda_2 \in K$, $x_1, x_2 \in W$.

5.3 INNER PRODUCT SPACES

An *inner product* (a generalisation of *dot product* from high school calculus) on a real vector space V is a scalar-valued function on the Cartesian product of V with itself having the following axioms:

$$\begin{aligned} \text{Symmetry } (x, y) &= (y, x) \\ \text{Linearity in the first argument } (ax, y) &= a(x, y) \\ (x + y, z) &= (x, z) + (y, z) \\ \text{Positive definiteness } (x, x) &\geq 0 \quad (x, x) = 0 \iff x = 0. \end{aligned} \quad (5.4)$$

Inner products on complex vector spaces are also possible, but a discussion is outside the scope of this chapter. We also note that inner products are sometimes written as $\langle x | y \rangle$ (for example, in physics) instead of (x, y) where x and y are vectors. For the specific case of n -dimensional vectors we usually write the inner product as follows:

$$x^T y \equiv (x, y) = \sum_{j=1}^n x_j y_j \quad x, y \in \mathbb{R}^n \quad (5.5)$$

where x^T is the transpose of vector x .

This latter notation is common in linear algebra and applications. Of course, more general vector spaces will need the more generic form in axioms (5.4).

An *inner product space* is a vector space on which an inner product is defined. A finite-dimensional real inner product space is known as a *Euclidean space*, and a complex inner product space is known as a *unitary space*. The *length* of a vector x in Euclidean space is defined to be $\sqrt{(x, x)} = (x, x)^{1/2}$, and the angle between two vectors x and y is given by:

$$\cos \theta = \frac{(x, y)}{(x, x)^{1/2}(y, y)^{1/2}}. \quad (5.6)$$

We say that two vectors x and y are *orthogonal* if $(x, y) = 0$. We immediately see that the zero vector is orthogonal to every other vector. Another example is $x = (2, 3, 1)$, $y = (4, -2, -2)$; then $(x, y) = 0$.

Definition 5.4 The set $\{x_1, \dots, x_n\}$ in an inner product space is *orthonormal* if:

$$(x_i, x_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j. \end{cases} \quad (5.7)$$

Finding an orthonormal set in an inner product space is analogous to choosing a set of mutually perpendicular unit vectors in elementary vector analysis.

5.3.1 Orthonormal Basis

Let X be an inner product space. The set $\{e_1, \dots, e_n\} \subset X$ is *orthonormal* if $\|e_j\| = \sqrt{(e_j, e_j)} = 1$ for $1 \leq j \leq n$ and $(e_j, e_k) = 0$ for $1 \leq j, k \leq n$, $j \neq k$. The set $\{e_1, \dots, e_n\} \subset X$ is called an *orthonormal basis*.

$$\text{Then } x = \sum_{j=1}^n (x, e_j) e_j \quad \forall x \in X.$$

The inner product space $C[-\pi, \pi]$ (continuous functions) has:

- Orthnormal basis $e_j(t) \equiv \frac{e^{j\pi t}}{\sqrt{2\pi}} (i = \sqrt{-1})$, $j = 1, \dots, n$.
- Orthogonality because $(e_k, e_j) = \int_{-\pi}^{\pi} e^{-i(j-k)t} dt = 0$, $j \neq k$.

An interesting application of inner products is to *kernel theory* to statistical learning in *Learning with Kernels*, Schölkopf and Smola (2002). In this case we do not work in an original (let's say n -dimensional) space X but in a *feature space* H . To this end, consider the map:

$$\Phi : X \rightarrow H, y = \Phi(x), \quad x \in X. \quad (5.8)$$

We embed data into H , and this approach offers several advantages, one of which is that we can define a *similarity measure* from the inner product in H :

$$k(x, x') := (x, x') = (\Phi(x), \Phi(x')), \quad x, x' \in X. \quad (5.9)$$

The function $k(., .)$ is called a *kernel*.

5.4 FROM VECTOR SPACES TO MATRICES

We are almost finished with our introduction to vector spaces and linear transformations. We now discuss how matrices arise and their relationship with the current topics.

We discuss the notion of a matrix for a linear transformation. To this end, consider the linear transformation $A \in L(V; W)$ where V and W are finite-dimensional vector spaces, and let $\{e_1, \dots, e_n\}$ and $\{f_1, \dots, f_m\}$ be bases in V and W , respectively. Then:

$$A(e_j) = \sum_{i=1}^m a_{ij} f_i \quad j = 1, \dots, n. \quad (5.10)$$

for some scalars a_{ij} , $1 \leq i \leq m$, $1 \leq j \leq n$. We can represent these scalars in rectangular form which we call a *matrix*:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}. \quad (5.11)$$

In this case we speak of a *square matrix* when $m = n$; otherwise, it is called a *rectangular matrix*. In short, each linear transformation determines a unique $m \times n$ matrix with respect to the basis functions. Conversely, every such matrix determines a unique linear transformation from V to W defined by Equation (5.10) and the following mapping for a general vector $x \in V$:

$$A(x) = x_1 A(e_1) + \dots + x_n A(e_n) \text{ where } x = \sum_{j=1}^n x_j e_j. \quad (5.12)$$

5.4.1 Some Examples

We take simple two-dimensional problems to model reflection and (counterclockwise) rotation in the plane.

Case 1:

$$A : \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

$$e_1 = (1, 0), e_2 = (0, 1)$$

$$Ae_1 = 1e_1 + 0e_2 = e_1$$

$$Ae_2 = 0e_1 - 1e_2 = -e_2$$

$$A = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Case 2:

$$A : \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

$$Ae_1 = (\cos \theta)e_1 + (\sin \theta)e_2$$

$$Ae_2 = -(\sin \theta)e_1 + (\cos \theta)e_2$$

$$A = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}.$$

From (5.10) we get

$$Ae_1 = a_{11}e_1 + a_{21}e_2$$

$$Ae_2 = a_{12}e_1 + a_{22}e_2$$

which allows us to find the matrix A .

Case 3:

$$A : \mathbb{R}^3 \rightarrow \mathbb{R}^3 (e_1 = (1, 0, 0), e_2 = (0, 1, 0), e_3 = (0, 0, 1))$$

$$A(x_1, x_2, x_3) = (x_1 + x_2, x_1 + x_3, 0)$$

$$\text{Then } Ae_1 = (1, 1, 0) = 1.e_1 + 1.e_2 + 0.e_3$$

$$Ae_2 = (1, 0, 0) = 1.e_1 + 0.e_2 + 0.e_3$$

$$Ae_3 = (0, 1, 0) = 0.e_1 + 1.e_2 + 0.e_3$$

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}.$$

$$(\text{use } Ae_1 = a_{11}e_1 + a_{21}e_2 + a_{31}e_3$$

$$Ae_2 = a_{12}e_1 + a_{22}e_2 + a_{32}e_3$$

$$Ae_3 = a_{13}e_1 + a_{23}e_2 + a_{33}e_3).$$

The trick is to build the matrix column by column, from top to bottom.

5.5 FUNDAMENTAL MATRIX PROPERTIES

We first give a short history of how matrices were discovered.

The term matrix was introduced by the 19th-century English mathematician James Sylvester, but it was his friend the mathematician Arthur Cayley who developed the algebraic aspect of matrices in two papers in the 1850s. Cayley first applied them to the study of systems of linear equations, where they are still very useful. They are also important because, as Cayley recognised, certain sets of matrices form algebraic systems in which many of the ordinary laws of arithmetic (e.g., the associative and distributive laws) are valid but in which other laws (for example, the commutative law) are not valid. (Wikipedia)

We give a precise overview of the operations that can be performed on matrices. We mention that they subsume operations on one-dimensional vectors because the latter can be viewed as matrices with one column (or with one row, depending on the context).

The notation and operations for vectors are:

$$\begin{aligned} a &= (a_1, \dots, a_n), b = (b_1, \dots, b_n) \\ a + b &= (a_1 + b_1, \dots, a_n + b_n) \\ \lambda a &= (\lambda a_1, \dots, \lambda a_n), \lambda \in \mathbb{R} \\ a \cdot b &= \sum_{j=1}^n a_j b_j \\ \|a\| &= \sqrt{a \cdot a} = \left(\sum_{j=1}^n a_j^2 \right)^{1/2} = \sqrt{\sum_{j=1}^n a_j^2} \end{aligned} \tag{5.13}$$

and for rectangular matrices:

$$A = (a_{ij}) = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}. \tag{5.14}$$

Some special cases of matrices are:

- *Row matrix*: has one row (row vector).
- *Column matrix*: has one column (column vector).
- *Zero matrix*: all entries have the value zero.
- *Diagonal matrix*: all entries zero except those on main diagonal.
- *Identity matrix*: diagonal matrix all of whose diagonal elements == 1.

Matrix operations are:

1. Matrix addition (and subtraction).
 2. Scalar multiplication.
 3. Matrix multiplication.
 4. Matrix transpose.
 5. Trace (sum of diagonal elements of a matrix).
- (5.15)

Matrix addition: $A, B \in \mathbb{R}^{m,n}$, $C \in \mathbb{R}^{m,n}$

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & & & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}$$

$$C = A + B = (a_{ij} + b_{ij}) = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2n} + b_{2n} \\ \vdots & & & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \cdots & a_{mn} + b_{mn} \end{bmatrix}$$

Scalar multiplication: $A \in R^{m,n}, \lambda \in \mathbb{R}$

$$\lambda A = (\lambda a_{ij}) = \begin{bmatrix} \lambda a_{11} & \lambda a_{12} & \cdots & \lambda a_{1n} \\ \lambda a_{21} & \lambda a_{22} & \cdots & \lambda a_{2n} \\ \vdots & & & \vdots \\ \lambda a_{m1} & \lambda a_{m2} & \cdots & \lambda a_{mn} \end{bmatrix}$$

Matrix multiplication: $A \in \mathbb{R}^{m,p}, B \in \mathbb{R}^{p,n}, C \in \mathbb{R}^{m,n}$

$$A = (a_{ij})_{\substack{1 \leq i \leq m \\ 1 \leq j \leq p}}, B = (b_{ij})_{\substack{1 \leq i \leq p \\ 1 \leq j \leq n}}$$

$$C = AB, \quad C = (c_{ij})_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}}$$

$$c_{ij} = \sum_{k=1}^p a_{ik} b_{kj}, \quad i = 1, \dots, m, \quad j = 1, \dots, n.$$

Transpose of a matrix A

- A new matrix obtained by writing the rows of A as columns.
- Applicable to rectangular matrices:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$A^\top = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}.$$

5. Matrix trace (the sum of the diagonal element of a square matrix):

$$A = (a_{ij})_{\substack{1 \leq i, j \leq n}} \\ tr(A) = \sum_{i=1}^n a_{ii}$$

If $B = (b_{ij})_{\substack{1 \leq i, j \leq n}}$ then

$$tr(AB) = tr(BA).$$

An example of matrix multiplication is:

$$\begin{bmatrix} 1 & 3 \\ 2 & -1 \end{bmatrix} \times \begin{bmatrix} 17 & -6 & 14 \\ -1 & 2 & -14 \end{bmatrix} = \begin{bmatrix} 2 & 0 & -4 \\ 5 & -2 & 6 \end{bmatrix}.$$

5.6 ESSENTIAL MATRIX TYPES

In this section we classify matrices based on some fundamental (computed) property. These properties are used in various areas of numerical analysis and its applications. It is then a useful reference to group them as we do here. In general, matrices can have real or complex values. In the latter case we recall *complex conjugation*:

$$z = x + iy, \bar{z} = x - iy \text{ (complex conjugate of } z\text{), } z, \bar{z} \in \mathbb{C}.$$

In many cases we are concerned with square matrices with real values.

5.6.1 Nilpotent and Related Matrices

A *nilpotent matrix* A is a square matrix such that $A^p = 0$ for some positive integer p . It is said to be of *index p* if p the least positive integer for which $A^p = 0$. For example, the matrix:

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

is nilpotent with index 2. More generally, a triangular matrix of size n with zeros along the main diagonal is nilpotent with index $\leq n$. For example, the follow matrix is nilpotent with index 3:

$$A = \begin{pmatrix} 1 & 5 & -2 \\ 1 & 2 & -1 \\ 3 & 6 & -3 \end{pmatrix}.$$

The determinant and trace of a nilpotent matrix are always zero. Thus, such matrices are not invertible. However, $I - N$ and $I + N$ are invertible where N is a nilpotent matrix:

$$(I - N)^{-1} = \sum_{n=0}^{\infty} N^n = I + N + N^2 + \dots$$

$$(I + N)^{-1} = \sum_{n=0}^{\infty} (-N)^n = (I - (-N))^{-1} \quad (5.16)$$

where I is the identity matrix. Since there are only finitely many non-zero terms, we see that both sums converge.

An *idempotent matrix* A is one for which $A^2 = A$. Examples are:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, A = \begin{pmatrix} 3 & -6 \\ 1 & -2 \end{pmatrix}.$$

Idempotent matrices arise in regression analysis and econometrics, for example in ordinary least squares problems, in particular when estimating sums of squared residuals.

An *involutory matrix* is one that is its own inverse:

$$A^2 = I$$

An example is:

$$A = \begin{pmatrix} a & b \\ c & -a \end{pmatrix} \text{ provided } a^2 + bc = 1.$$

For example, the Pauli matrices are involutory:

$$\begin{aligned}\sigma_1 &= \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ \sigma_2 &= \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} (i = \sqrt{-1}) \\ \sigma_3 &= \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.\end{aligned}$$

5.6.2 Normal Matrices

We introduce the important class of *normal matrices* by introducing some prerequisite notation. The *transpose* of a real $m \times n$ matrix is an $n \times m$ matrix formed by exchanging the rows and columns of A :

$$\begin{aligned}A &= (a_{ij}), 1 \leq i \leq m, 1 \leq j \leq m \\ A^\top &= (a_{ji}), 1 \leq j \leq n, 1 \leq i \leq m \text{ (transpose).}\end{aligned}$$

In the case of a complex matrix A , the *Hermitian transpose* is the complex conjugate transpose of A :

$$\begin{aligned}\bar{A} &= (\bar{a}_{ij}), 1 \leq i \leq m, 1 \leq j \leq n \\ A^H &= (\bar{A})^\top = \bar{A}^\top \text{ Hermitian transpose.}\end{aligned}$$

We are ready to give some more definitions of special kinds of matrices:

Normal: $A^H A = A A^H$

Hermitian: $A = A^H$

Skew-Hermitian: $A^H = -A$

Symmetric (real) matrix: $A = A^\top$.

An example of a Hermitian (and hence normal matrix) is:

$$A = \begin{pmatrix} 2 & 3+4i \\ 3-4i & -5 \end{pmatrix}, \quad \bar{A} = \begin{pmatrix} 2 & 3-4i \\ 3+4i & -5 \end{pmatrix}.$$

5.6.3 Unitary and Orthogonal Matrices

A matrix U is *unitary* if its inverse equals its Hermitian transpose:

$$U^{-1} = U^H = \overline{U}^T.$$

Unitary matrices are normal: $UU^H = UU^{-1} = I = U^{-1}U = U^HU$. Furthermore, given two vectors, multiplication by U preserves inner products $(Ux, Uy) = (x, y)$. Unitary matrices are important in quantum mechanics because they preserve norms and thus probability amplitudes. See the Appendix in this chapter.

A real square matrix Q is *orthogonal* if:

$$Q^T Q = Q Q^T = I \text{ or } Q^T = Q^{-1}. \quad (5.17)$$

Orthogonal matrices are important in numerical linear algebra applications because of their numeric stability properties. Some application areas are matrix decomposition methods (Golub and van Loan (1996)) such as:

- *QR decomposition* $A = QR$, Q orthogonal, R upper triangular.
- *Singular Value Decomposition (SVD)* $A = U \sum V$, U and V orthogonal, \sum diagonal matrix.
- *Eigendecomposition* $S = Q \wedge Q^T$, S symmetric, Q orthogonal, \wedge diagonal.
- *Polar decomposition* $A = QS$ where Q is orthogonal, S symmetric positive-semidefinite.

A particular application area is solving *overdetermined systems* of linear equations and solving *ill-posed linear systems*.

5.6.4 Positive Definite Matrices

This is another very important class of matrices. Matrices having this property are highly desirable in applications and algorithms. An $n \times n$ matrix A is *positive definite* if:

$$(Ax, x) > 0 \quad \forall x \in \mathbb{R}^n \quad (5.18)$$

and *positive semidefinite* if:

$$(Ax, x) \geq 0 \quad \forall x \in \mathbb{R}^n. \quad (5.19)$$

Some necessary conditions for positive semidefiniteness are:

- The diagonal elements of A must be positive.
- A is positive definite if and only if all its eigenvalues are positive.
- The element of A having the greatest absolute value must be on the diagonal of A .
- $a_{ii}a_{jj} > |a_{ij}|^2 \quad \forall i, j = 1, \dots, n, i \neq j$.

An example of a positive definite matrix A is:

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}.$$

To prove this let $z = (a, b, c)^\top$. Then

$$(Az, z) \equiv z^\top Az = a^2 + (a - b)^2 + (b - c)^2 + c^2 > 0.$$

We can take the *square root* $A^{1/2}$ of a positive definite matrix A , and it is a well-defined function. Furthermore, we can factor a positive definite matrix A into:

$$A = LL^H \quad (5.20)$$

where L is a lower triangular matrix having positive values on its diagonal. Equation (5.20) is called the *Cholesky decomposition* for A .

5.6.5 Non-Negative Matrices

A matrix A is *non-negative* (written $A \geq 0$) if all its elements are real and non-negative. A matrix A is greater than a matrix B if $A - B$ is positive. These kinds of matrices have many applications, for example Markov chains and stochastic matrix theory. They are also relevant when we construct monotone finite difference schemes and M -matrices to approximate the solution of differential equations, as we shall see later in this book.

5.6.6 Irreducible Matrices

A matrix A is said to be *reducible* if there exists a *permutation matrix* P such that:

$$PAP^\top = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}, \quad (A_{11}, A_{12}, A_{22} \text{ are square matrices}).$$

The matrix A is called *irreducible* if no such permutation matrix exists.

The matrix A is said to be *diagonally dominant* if:

$$|a_{jj}| \geq \sum_{\substack{k=1 \\ k \neq j}}^n |a_{jk}| \equiv \rho_j \quad \forall j = 1, \dots, n \text{ where } A = (a_{ij}), \quad 1 \leq i, j \leq n. \quad (5.21)$$

Some results that are used in PDE applications are:

1. If A is strictly diagonally dominant, then it is invertible.
2. If A is irreducible and diagonally dominant and if $|a_{jj}| > \rho_j$ for at least one j , then A is invertible.

5.6.7 Other Kinds of Matrices

We conclude this section with a list of matrices whose properties are defined by the signs of their off-diagonal elements.

A *Metzler matrix* A satisfies $a_{ij} \geq 0, i \neq j$ where $A = (a_{ij})$.

A *Z-matrix* is a negated Metzler matrix $a_{ij} \leq 0, i \neq j$ where $A = (a_{ij})$.

An *M-matrix* is a Z-matrix with eigenvalues whose real parts are non-negative. An *M-matrix* can be expressed in the form $A = sI - B, B = (b_{ij}), b_{ij} \geq 0, i, j = 1, \dots, n$. The scalar s is at least as large as the maximum of the moduli of the eigenvalues of B , and I is the identity matrix.

Some applications of *M-matrices* are from mathematics and economics, for example:

- Establish bounds on eigenvalues.
- Convergence criteria for iterative methods.
- Discretisations of PDEs (for example, in combination with exponential fitting) and monotone finite difference schemes.
- Finite Markov chains.
- Population dynamics.

Finally, *L-matrices* are defined by: $L = (l_{ij}), l_{ii} > 0, l_{ij} < 0, i \neq j$.

5.7 THE CAYLEY TRANSFORM

The Cayley Transform refers to a group of related concepts. The relevance to our work is that it appears when proving the stability of finite difference schemes for two-factor option pricing problems as we shall discuss in Chapters 18, 22 and 23. It has been applied in fixed-income pricing as discussed in Davidson and Levin (2014).

In general, we define:

$$Q = (I - \sigma A)(I + \sigma A)^{-1}, \sigma > 0. \quad (5.22)$$

In the case where A is positive definite, we can compute the norm of Q as follows:

$$\begin{aligned} \|Q\|^2 &= \max_{\varphi} \frac{(Q\varphi, Q\varphi)}{(\varphi, \varphi)} = \max_{\varphi} \frac{((I - \sigma A)\varphi, (I - \sigma A)\varphi)}{((I + \sigma A)\varphi, (I + \sigma A)\varphi)}, \\ &= \max_{\varphi} \frac{(\varphi, \varphi) - 2\sigma(A\varphi, \varphi) + \sigma^2(A\varphi, A\varphi)}{(\varphi, \varphi) + 2\sigma(A\varphi, \varphi) + \sigma^2(A\varphi, A\varphi)} \leq 1. \end{aligned}$$

This result was proved in Kellogg (1964), and it is an important lemma to prove unconditional stability of splitting schemes. In the same way it is possible to show that $\|(I + \sigma A)^{-1}\| < 1$.

Appendix : The Schrödinger Equation

We discuss the time-dependent one-dimensional Schrödinger PDE and its finite difference approximation by the Euler, fully implicit, and Crank–Nicolson schemes. We prove that only the Crank–Nicolson scheme is unitary (preserves norms) as discussed in Section 5.6.3 in the context of unitary matrices.

In some cases and applications we are interested in solving linear systems of equations where the coefficients are complex-valued. We take the example of the time-dependent linear Schrödinger equation in one dimension:

$$i \frac{\partial \psi}{\partial t} = -\frac{\partial^2 \psi}{\partial x^2} + V(x)\psi, \quad -\infty < x < \infty \quad (5.23)$$

where:

ψ = Schrödinger wave function.

$V(x)$ = one-dimensional potential.

$i = \sqrt{-1}$.

Equation (5.23) describes the scattering of a one-dimensional wave packet by the potential $V(x)$. We have assumed for convenience that Planck's constant $h = 1$ in this example.

We rewrite Equation (5.23) in the equivalent form:

$$i \frac{\partial \psi}{\partial t} = H\psi \text{ where } H = -\frac{\partial^2}{\partial x^2} + V(x). \quad (5.24)$$

The operator H is called the *Hamiltonian operator*, and it determines the time variation of the system. We are now interested in approximating equation (5.24) using finite difference schemes. To this end, the *explicit Euler FTCS* scheme is given by:

$$i \frac{\psi^{n+1} - \psi^n}{k} = H\psi^n \text{ or } \psi^{n+1} = (1 - iHk)\psi^n. \quad (5.25)$$

while the *implicit Euler BTCS* scheme is given by:

$$i \frac{\psi^{n+1} - \psi^n}{k} = H\psi^{n+1}$$

or

$$(i - Hk)\psi^{n+1} = i\psi^n \quad (5.26)$$

or

$$(1 + iHk)\psi^{n+1} = \psi^n.$$

Scheme (5.25) is unstable, while (5.26) is stable. However, neither scheme is unitary in the sense of the original problem; that is, the total probability of finding the particle somewhere is 1:

$$\int_{-\infty}^{\infty} |\psi(x)|^2 dx = 1. \quad (5.27)$$

A remedy for this is to use the *Cayley form* (this is essentially the Crank–Nicolson scheme):

$$i \frac{\psi^{n+1} - \psi^n}{k} = \frac{H}{2}(\psi^{n+1} + \psi^n)$$

or

$$(1 + \frac{i}{2}Hk)\psi^{n+1} = (1 - \frac{i}{2}Hk)\psi^n. \quad (5.28)$$

This scheme is unitary; you can check this by a bit of arithmetic using complex arithmetic.

The solution of (5.24) is:

$$\psi(x, t) = e^{-iHt}\psi(x, 0) \quad (5.29)$$

and the solution of (5.28) is:

$$\psi^{n+1} = \left(\frac{1 - \frac{1}{2}iHk}{1 + \frac{1}{2}iHk} \right) \psi^n. \quad (5.30)$$

We can see that (5.30) is the (1,1) Padé approximant to the exponential function. Its absolute value is 1.

5.8 SUMMARY AND CONCLUSIONS

Matrix theory is too important to be ignored or given short shrift in any book on numerical analysis and its applications. For this reason, we gave a reasonably detailed exposition of matrix theory as a companion to the other chapters in this book (and it could possibly be a companion to other books).

Numerical Solutions of Boundary Value Problems

The world is continuous, but the mind is discrete.

David Mumford

6.1 INTRODUCTION AND OBJECTIVES

This chapter builds on the topics that we introduced in Chapters 4 and 5. The focus now is on solving linear systems of equations and in particular tridiagonal matrix systems, as these are very common when we approximate PDEs by the finite difference method. We offer two popular matrix solvers to compute a solution of such systems. For us, they are universal algorithms and are at the core of finite difference schemes for one-factor and two-factor option pricing problems in this book. In particular, we employ them when we use the Crank–Nicolson and fully implicit (Euler) methods for time-dependent PDEs. For example, we use tridiagonal solvers when using ADI and splitting methods (as we shall discuss in Chapters 18, 22 and 23). We note that matrix solvers are not needed when we use the Alternating Direction Explicit (ADE) method (Chapter 19) or the Method of Lines (MOL) (Chapter 20).

In the interest of completeness, we have given an introduction to a number of iterative methods for finding the solution to linear systems. These methods are used when solving time-independent problems for elliptic PDEs and elliptic variational inequalities and applications to American option pricing.

6.2 AN INTRODUCTION TO NUMERICAL LINEAR ALGEBRA

We give a global overview of *numerical linear algebra*. There is a vast literature on this subject, and it is one of the cornerstones of numerical analysis. In fact, many problems can be reduced to a system of linear equations.

Our interest in this chapter is the following problem: given a vector F of length n and a *square matrix* A of size n (that is, one with n rows and n columns), find the unique vector U that satisfies the linear system (assuming this solution exists):

$$AU = F. \quad (6.1)$$

We say that U is the solution of the Equation (6.1), and we write it formally as:

$$U = A^{-1}F. \quad (6.2)$$

In general, we do not calculate the inverse of A directly because this is too cumbersome. Instead we use so-called *matrix solvers* to compute the solution U . There are more efficient and less resource-intensive techniques than using a sledgehammer algorithm to invert the matrix A .

Writing an equation in the form (6.1) is not difficult in general (although, as we shall see in later chapters, assembling the equations involves a lot of basic arithmetic); there are a number of issues to be addressed:

- Does Equation (6.1) have a solution?
- Can we find sufficient conditions on matrix A to produce a solution U to (6.1)?
- How sensitive is the solution U to small perturbations in the matrix A ?
- What are the different kinds of structures that A might have?
- Can we find efficient, reliable and accurate matrix solvers for Equation (6.1)?

We shall discuss these topics in the course of this chapter. First, we draw a distinction between the zero and non-zero elements of the matrix A . If we know that some elements of A are zero and if the distribution of zeros follows a recognisable pattern, then we might hope to develop matrix solvers that take these structures into account with the expectation of making these solvers as efficient as possible. Furthermore, we do not need to store the zero elements in memory, thus adding to good resource utilisation.

There are various kinds of matrix structures that we are exposed to in numerical analysis. Related to this issue is how to store the matrices in memory, how to access the matrix elements (read and write) and how to solve systems of equations in which these matrices play the role of the matrix A in Equation (6.1). Some possible matrix structures are:

- Full dense matrix
- Sparse matrix
- Patterned matrices.

A *full matrix* is one in which all elements must be stored in memory because, essentially, there are very few or no zero elements in it. Such matrices are common in some applications (for example, solving integral equations or *meshless methods*), but they are not common in the finite difference method. In general, a full matrix with n rows and m columns will require nm memory locations. A *sparse matrix* is one in which the majority of elements are zero. This implies that we only have to store a small percentage of

the elements in an appropriate data structure. One of the challenges associated with the choice of data structure is the problem of accessing the non-zero elements in the matrix.

Sparse matrices crop up when modelling multi-dimensional problems using finite differences or the finite element method (FEM), and again there is a vast literature on the subject. A *patterned matrix* is one in which the zero and non-zero elements bear a well-defined structural relationship with each other. There are many kinds of patterned matrices. In general, a *band matrix* is one whose diagonal elements and $2K$ off-diagonal elements are non-zero. An important special case is when $K = 1$; this is the class of *tridiagonal matrices*. Thus, a tridiagonal matrix is one with three non-zero diagonals. These matrices are very important in finite differences methods for ordinary and partial differential equations when we employ three-point difference schemes to discretise the space variable. A more general kind of *tridiagonal matrix* is the so-called *block tridiagonal matrix* (see Isaacson and Keller (1966), Golub and Van Loan (1996)). This type of matrix is needed when we approximate *systems* of partial differential equations by finite differences or when we reduce a second-order parabolic partial differential equation to a first-order system in order to get a good approximation to the solution and its first derivative. (In the case of the Black–Scholes equation, this is equivalent to saying that we approximate the option price and its delta using a block tridiagonal matrix system.)

A *lower triangular* matrix is one whose non-zero elements are all on or below the main diagonal, while an *upper triangular* matrix is one whose non-zero elements are all on or above the main diagonal.

We now discuss two major categories of methods for solving linear systems of the form (6.1).

By a *direct method* for solving a system of linear equations (6.1) we mean a method that gives the exact solution U after a finite number of steps. We mention some common methods:

- Gaussian elimination for full matrices.
- *LU* decomposition techniques.
- Crout's method.
- The Thomas and Double Sweep algorithms for tridiagonal matrices.

In this book we are interested in solving systems (6.1) where the matrix A is tridiagonal. We propose two schemes, one of which is a specialisation of *LU* decomposition.

An *iterative scheme* starts from a first approximation that is then successively improved until a sufficiently accurate solution is obtained (Varga (1962)). The big question is: Does the algorithm implementing the iterative scheme converge, how efficient is the algorithm and how many iterations are needed before the desired accuracy is achieved? Popular iterative methods are discussed in Thomas (1999).

The methods that we discuss in Section (6.6) are:

- The Jacobi method.
- The Gauss–Seidel method.
- Successive overrelaxation (SOR).
- The conjugate gradient method.
- The projected SOR method and its relationship with American option pricing.

In general, iterative methods are suitable for sparse matrix systems. We mention them in this chapter as a resource for future work and because we need to understand the theory underlying the *projected SOR method* (Cryer (1979)) that is used when we discretise the Black–Scholes equation for American options. In this case we get a system of *vector inequalities* having the general form:

$$(U - c) \cdot (AU - b) = 0 \quad (6.3)$$

where $AU \geq b$, $U \geq c$, c is a vector and the dot ‘ \cdot ’ denotes the inner product of two vectors.

6.2.1 BLAS (Basic Linear Algebra Subprograms)

BLAS is a *de facto* application programming interface standard for libraries that perform basic linear algebra operations on vectors and matrices. They promote the readability, modularity and maintainability of software. BLAS promotes *modularity* by identifying frequently occurring operations of linear algebra and by specifying a standard interface to these operations. *Efficiency* is achieved by optimising the code within BLAS. It is important to identify and define a set of basic operations that is rich enough to allow us to model high-level algorithms on the one hand and simple enough to allow optimisation on a range of computers on the other hand.

The advantages of using BLAS are its robustness, portability and readability. The BLAS functionality consists of three levels that we now discuss.

BLAS Level 1

This level consists of low-level operations such as *inner products* (also known as *dot products*) of vectors and the addition of a multiple of one vector to another vector. These *vector-vector operations* are referred to as Level 1 BLAS such as:

$$y \leftarrow \alpha x + y \quad (6.4)$$

where x and y are vectors and α is a scalar.

These operations involve $O(n)$ floating-point operations in general, where n is the length of the vectors. In Equation (6.4) we see that the vector y is being updated, and the operation is sometimes called *saxpy* (‘scalar αx plus y ’).

BLAS Level 2

This level contains matrix-vector operations of the form:

$$y \leftarrow \alpha Ax + \beta y \quad (6.5)$$

where x and y are vectors, A is a matrix and α and β are scalars.

These *matrix-vector operations* occur during the implementation of many of the most common algorithms in linear algebra. The Level 2 BLAS involves $O(mn)$ scalar operations where m and n are the dimensions of the matrix involved in the operations.

Other operations in BLAS Level 2 are:

- Rank-one and rank-two updates:

$$\begin{aligned} A &= \alpha xy^T + A \text{ (rank-1 update)} \\ A &= \alpha xy^T + \alpha yx^T + A \text{ (rank-2 update)} \end{aligned} \quad (6.6)$$

where y^T is the transpose of velocity y .

- Solution of triangular equations of the form:

$$Ty = x$$

where T is a non-singular *upper triangular matrix* (all elements below the main diagonal are zero) or *lower triangular matrix* (all elements above the main diagonal are zero).

In general, the operations apply to general band, Hermitian, Hermitian band and triangular band matrices with real and complex coefficients in both single and double precision.

BLAS Level 3

This level concerns *matrix-matrix operations*, and they are similar to Level 2 operations. In general, we replace the vectors x and y in Equations (6.6) by matrices B and C , for example. This approach keeps the design of the software as consistent as possible with the software of the Level 2 BLAS. It also helps users remember the calling sequences and parameter conventions. Some examples are:

- Matrix-matrix products:

$$\begin{aligned} C &= \alpha AB + \beta C \\ C &= \alpha A^T B + \beta C \\ C &= \alpha AB^T + \beta C \\ C &= \alpha A^T B^T + \beta C. \end{aligned} \quad (6.7)$$

- Rank- k updates of a symmetric matrix C :

Here A , B and C are matrices, α and β are scalars.

$$\begin{aligned} C &= \alpha AA^T + \beta C (A^T \text{ is transpose of } A) \\ C &= \alpha A^T A + \beta C \\ C &= \alpha A^T B + \alpha B^T A + \beta C \\ C &= \alpha AB^T + \alpha BA + \beta C. \end{aligned} \quad (6.8)$$

- Multiplying a matrix B by a triangular matrix T :

$$B = \alpha T B$$

$$B = \alpha T^\top B$$

$$B = \alpha B T$$

$$B = \alpha B T^\top.$$

- Solving triangular systems of equations with multiple right-hand sides:

$$\begin{aligned} B &= \alpha T^{-1} B \quad (T^{-1} \text{ is inverse of } T) \\ B &= \alpha B T^{-1} \end{aligned} \tag{6.9}$$

where α and β are scalars; A , B and C are rectangular matrices; T is an upper or lower triangular matrix. Boost C++ library uBLAS, for example, supports the above BLAS operations, and they are mapped to appropriate C++ function calls.

6.3 DIRECT METHODS FOR LINEAR SYSTEMS

This section is concerned with direct methods for linear systems, with special emphasis on LU decomposition, and the solution of tridiagonal systems using the Thomas and Double Sweep methods.

6.3.1 LU Decomposition

We discuss the problem of solving linear system (6.1) in which the matrix A is dense or has a general structure. Computing the inverse explicitly as in (6.2) is a non-starter, so we need a way to break the problem into simpler subproblems. To this end, we perform a so-called *LU factorisation* on A :

$$A = LU$$

where L is a lower triangular matrix and U is an upper triangular matrix. The factorisation, when it exists, is unique if the elements on the main diagonal of L are all 1s. The form of these matrices is:

$$L = \begin{bmatrix} \alpha_{00} & 0 & 0 & \dots & 0 \\ \alpha_{10} & \alpha_{11} & 0 & \dots & 0 \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & \dots & 0 \\ \vdots & & & & \vdots \\ \alpha_{n-1,0} & \alpha_{n-1,1} & \dots & & \alpha_{n-1,n-1} \end{bmatrix} \quad \begin{array}{l} \alpha_{ij} = 0, \quad i < j \\ i, j = 0, \dots, n-1 \end{array} \tag{6.10}$$

$$U = \begin{bmatrix} \beta_{00} & \beta_{01} & \dots & \beta_{0,n-1} \\ 0 & \beta_{11} & \beta_{12} & \dots & \beta_{1,n-1} \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 0 & \beta_{n-1,n-1} \end{bmatrix} \quad \begin{array}{l} \beta_{ij} = 0, \quad i > j \\ i, j = 0, \dots, n-1. \end{array}$$

The two main use cases are:

1. Computing the matrices L and U .

$$A = LU, A = (a_{ij})$$

$$a_{ii} = 1, i = 0, \dots, n - 1$$

$$\beta_{ij} = a_{ij} - \sum_{k=0}^{i-1} \alpha_{ik} \beta_{kj}, \quad j = 0, \dots, n - 1, i = 0, \dots, j. \quad (6.11)$$

$$\alpha_{ij} = \frac{1}{\beta_{jj}} \left[a_{ij} - \sum_{k=0}^{j-1} \alpha_{ik} \beta_{kj} \right], \quad j = 0, \dots, n - 1, i = j + 1, \dots, n - 1.$$

2. Solving triangular systems:

$$Ly = b, b = (b_0, \dots, b_{n-1})^\top$$

$$y = (y_0, \dots, y_{n-1})^\top$$

$$y_0 = b_0 / \alpha_{00} \quad (6.12)$$

$$y_j = \frac{1}{\alpha_{jj}} \left[b_j - \sum_{k=0}^{j-1} \alpha_{jk} y_k \right], \quad j = 1, \dots, n - 1$$

and:

$$Ux = y, x = (x_0, \dots, x_{n-1})^\top$$

$$y = (y_0, \dots, y_{n-1})^\top$$

$$x_{n-1} = y_{n-1} / \beta_{n-1, n-1} \quad (6.13)$$

$$x_j = \frac{1}{\beta_{jj}} \left[y_j - \sum_{k=j+1}^{n-1} \beta_{jk} x_k \right], \quad j = n - 2, \dots, 0.$$

3. Solving the original matrix system (6.1):

$$Ax = b$$

$$(LU)x = b = L(Ux)$$

$$1^0 Ly = b$$

$$2^0 Ux = y.$$

As a by-product, the determinant of the matrix A is the product of the diagonal elements of U multiplied by the product of the diagonal elements of L (the latter being equal to 1):

$$\det(A) = \prod_{j=1}^n \alpha_{jj} \prod_{j=1}^n \beta_{jj} = \prod_{j=1}^n \beta_{jj}.$$

The C++ source code for *LU* decomposition accompanies the book Duffy (2018) and may be accessed by purchasers of the book by contacting the author.

An example is:

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 2 & -1 & 1 \\ 4 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 2 & -2 & 0 \\ 4 & -1 & -1 \end{pmatrix} \begin{pmatrix} 2 & 1/2 & 1/2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \equiv LU.$$

You can compute the determinant of A in two different ways. It is equal to 4.

6.3.2 Cholesky Decomposition

This is a special case of *LU* decomposition when A is symmetric and positive definite. The algorithm is:

$$\begin{aligned} A &= LU = LL^\top \\ l_{jj} &= \left(a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 \right)^{1/2}, \quad 1 \leq j \leq n \\ l_{ij} &= \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) / l_{jj}, \quad i = j+1, \dots, n. \end{aligned} \tag{6.15}$$

Cholesky decomposition is used when we wish to compute correlated random numbers. See Duffy and Kienitz (2009).

6.4 SOLVING TRIDIAGONAL SYSTEMS

6.4.1 Double Sweep Method

A *band matrix* $A = (a_{ij})$ is a square matrix of size n and *width* $2K + 1$ such that $a_{ij} = 0$ when $|i - j| > K$, where K is a non-negative integer. All non-zero elements are positioned on the main diagonal and on the first K diagonals directly above and below it. Some special band matrices are:

- A *diagonal matrix* is a band matrix with $K = 0$.
- A *Toeplitz matrix* is a band matrix in which each diagonal consists of a single identical element but different diagonals may contain different elements.
- A *tridiagonal (Jacobi) matrix* is a band matrix of width three ($K = 1$).

Tridiagonal matrices are found in numerical analysis applications, for example when approximating the solution of ordinary and partial differential equations by the finite difference and finite element methods. To this end, we discuss how they arise

by first considering the simple *two-point boundary value problem* on the interval $(0, 1)$ with Dirichlet boundary conditions:

$$\begin{cases} \frac{d^2u}{dx^2} = f(x), & 0 < x < 1 \\ u(0) = \varphi, & u(1) = \psi. \end{cases} \quad (6.16)$$

The function $f(x)$ and constants φ and ψ are assumed to be known.

We approximate the solution u by creating *discrete mesh points* defined by $\{x_j\}$, $j = 0, \dots, J$, where J is a positive integer. At each interior mesh point we approximate the second-order derivative in Equation (6.16) by a second-order divided difference. The corresponding discrete scheme is:

$$\begin{cases} U_{j+1} - 2U_j + U_{j-1} = h^2 f(x_j), & 1 \leq j \leq J-1 \ (h = 1/J) \\ U_0 = \varphi, U_J = \psi. \end{cases} \quad (6.17)$$

This scheme can be written as a tridiagonal matrix system as follows:

$$AU = F$$

$$\text{where } U = (U_1, \dots, U_{J-1})^\top, F = (h^2 f(x_1) - \varphi, h^2 f(x_2), \dots, h^2 f(x_{J-1}) - \psi)^\top. \quad (6.18)$$

In this case A is a tridiagonal matrix, and the symbol \top denotes transpose. We solve this system for the unknown vector U .

We see that there are $J - 1$ unknown values (under more general boundary condition there would be $J + 1$ unknown values) because in this current case we are dealing with Dirichlet boundary conditions, which means that values of U are known at $x = 0$ and at $x = 1$. In more general cases, this is not necessarily so.

The next questions are: How do we model tridiagonal matrices, and how do we solve the tridiagonal system (6.18)? The answer to the first question is to model the matrix as three vectors, one of length J (the diagonal) and two vectors of length $J - 1$ (for the subdiagonal and superdiagonal). Regarding the second question (which solver to use), we discuss two solutions.

Double Sweep Method

The *Double Sweep* method is discussed in Godunov and Riabenki (1987), and it is an algorithm to solve three-point difference schemes with given boundary values. It corresponds to a discretisation of two-point boundary value problems by the finite difference method. The objective is to find a *discrete function* $\{U_j\}, j = 0, \dots, J$ satisfying the following set of equations:

$$\begin{cases} a_j U_{j-1} + b_j U_j + c_j U_{j+1} = f_j, & j = 1, \dots, J-1 \\ U_0 = \varphi, U_J = \psi \\ \{a_j\}, \{b_j\}, \{c_j\}, \{f_j\} \text{ known vectors.} \end{cases} \quad (6.19)$$

The *Double Sweep* method uses a *recurrence (backward) relation* to compute the solution starting from index J :

$$\begin{cases} U_J = \psi \\ U_j = L_j U_{j+1} + K_j, j = J-1, \dots, 0 \end{cases} \quad (6.20)$$

where the coefficients L_j, K_j are computed using a *forward recurrence relation*:

$$\begin{cases} L_j = -c_j / (b_j + a_j L_{j-1}) \\ K_j = (f_j - a_j K_{j-1}) / (b_j + a_j L_{j-1}) \\ j = 1, \dots, J-1 \end{cases} \quad (6.21)$$

and:

$$\begin{cases} L_0 = 0 \\ K_0 = \varphi. \end{cases}$$

It is possible to check that these formulae are correct by writing the solution of Equation (6.19) in the following form:

$$U_j = (f_j - a_j U_{j-1} - c_j U_{j+1}) / b_j, \quad j = 1, \dots, J-1 \quad (6.22)$$

and by comparing the terms with the coefficients in Equation (6.20). We leave this as an exercise. It is good practice to check that you get the same results.

Moving to code that implements this algorithm, we first see that we need four input arrays, three of which are the arrays comprising the tridiagonal matrix and the fourth is for the inhomogeneous term in Equation (6.19). We also need three work arrays, namely U (the solution), K and L from Equations (6.20) and (6.21). The arrays K and L are essential, it seems, while we recycle the array for the inhomogeneous term that will hold the values in the solution U . Then the number of work arrays can thus be reduced to two.

6.4.2 Thomas Algorithm

The Thomas algorithm is a popular algorithm, and it is well-documented (Thomas (1995)). It is used when approximating the solution of convection-diffusion partial differential equations (PDEs) using the finite difference method (FDM). It is a special case of *LU* decomposition for tridiagonal matrices.

We implement the Thomas algorithm in two steps. In general, we solve a system of the form $Au = r$ where A is a tridiagonal matrix, r is a given vector, and u is the unknown vector that we wish to find. We decompose A into the product of a *lower triangular matrix* L and an *upper triangular matrix* U as $A = LU$ leading to a simpler set of equations to solve:

$$\begin{cases} Au = r \iff (LU)u = r \\ Lz = r \\ Uu = z. \end{cases} \quad (6.23)$$

The *bidiagonal matrices* L and U are shown as follows:

$$L = \begin{pmatrix} \beta_0 & & & \\ a_1 & \ddots & \ddots & 0 \\ & \ddots & \ddots & \ddots \\ 0 & \ddots & \ddots & a_J \\ & & a_J & \beta_J \end{pmatrix}$$

$$U = \begin{pmatrix} 1 & \gamma_1 & & 0 \\ & \ddots & \ddots & \\ 0 & \ddots & \ddots & \gamma_{J-1} \\ & & & 1 \end{pmatrix}.$$

Their coefficients can be computed as follows:

$$\begin{cases} \beta_0 = b_0, \gamma_0 = c_0/\beta_0 \\ \beta_j = b_j - a_j\gamma_{j-1}, \quad j = 1, \dots, J \\ \gamma_j = c_j/\beta_j, \quad j = 1, \dots, J-1 \end{cases} \quad (6.24)$$

and the algorithm has the following component-wise form for the two iterations, respectively:

$$\begin{cases} z_0 = r_0/\beta_0 \\ z_j = (r_j - a_j z_{j-1})/\beta_j, \quad j = 1, \dots, J. \end{cases} \quad (6.25)$$

$$\begin{cases} U_J = z_J \\ U_j = z_j - \gamma_j U_{j+1}, \quad j = J-1, \dots, 0. \end{cases} \quad (6.26)$$

Examining these algorithms we have used three work arrays (namely, z , γ and β) in the code for convenience (we may be able to reduce this to two work arrays, but we do not consider this optimisation step).

Finally, we note that for the Thomas algorithm to work the corresponding tridiagonal matrix must be *diagonally dominant*, that is:

$$|b_i| \geq |a_i| + |c_i|, \quad i = 0, \dots, J. \quad (6.27)$$

Finally, we remark that the Double Sweep method is approximately 20% more efficient than the Thomas algorithm. This is a rough guideline.

6.4.3 Block Tridiagonal Systems

The LU decomposition technique can be applied to block tridiagonal systems. This is needed when we model first-order systems, partial differential equations and integral equations (Keller (1968)), for example the PDE systems in Chapter 17 based on the CSE approach. Consider the *block-tridiagonal matrix*:

$$A = \begin{pmatrix} A_1 & C_1 & & & \\ B_2 & A_2 & \ddots & & \\ & \ddots & \ddots & \ddots & 0 \\ 0 & & \ddots & \ddots & \ddots \\ & & & & C_{n-1} \\ & & & B_n & A_n \end{pmatrix} \quad (6.28)$$

where:

A_j = square matrix of order m_j

B_j, C_j = rectangular matrices that fit into the “pattern”.

Thus, B_j has m_j rows and m_{j-1} columns, and C_j has m_j rows and m_{j+1} columns.

A special case is where $m_j = m$; then all submatrices are square of order m . For 2×2 systems we have $m = 2$.

We seek a factorisation in the form $A = LU$ where:

$$L = \begin{pmatrix} \tilde{A}_1 & & & \\ B_2 & \tilde{A}_2 & & \\ 0 & \ddots & \ddots & 0 \\ & & \ddots & \ddots \\ & & & B_n & \tilde{A}_n \end{pmatrix} \quad (6.29)$$

and:

$$U = \begin{pmatrix} I_2 & \Gamma_2 & 0 & & \\ & \ddots & \ddots & & \\ 0 & & \ddots & \ddots & \Gamma_{n-1} \\ & & & & I_n \end{pmatrix}. \quad (6.30)$$

where:

I_j = identity matrices of order m_j

\tilde{A}_j = square matrix of order m_j

Γ_j = rectangular matrices with m_j rows, m_{j+1} columns.

Formally, the equality $A = LU$ can be decomposed as follows:

$$\begin{aligned} \tilde{A}_1 &= A_1; \Gamma_1 = A_1^{-1}C_1 \\ \tilde{A}_j &= A_j - B_j\Gamma_{j-1}, \quad j = 2, \dots, n \\ \Gamma_j &= \tilde{A}_j^{-1}C_j, \quad j = 2, 3, \dots, n-1. \end{aligned} \quad (6.31)$$

This algorithm is a generalisation of the scalar LU decomposition algorithm. However, the details of the algorithms will be slightly more complicated because we are solving linear systems with embedded vectors and matrices. From a C++ point of view, we speak of a *composite* or *nested* vector. In general, we wish to solve the system (Isaacson and Keller (1966)):

$$Ax = f$$

$$x = \begin{pmatrix} x^{(1)} \\ \vdots \\ x^{(n)} \end{pmatrix}, f = \begin{pmatrix} f^{(1)} \\ \vdots \\ f^{(n)} \end{pmatrix}. \quad (6.32)$$

The vectors are now calculated by the following scheme:

$$\begin{cases} Ly = f, \quad Ux = y \\ y^{(1)} = \tilde{A}_1^{-1}f^{(1)} \\ y^{(j)} = \tilde{A}_j^{-1}(f^{(j)} - \beta_j y^{(j-1)}), j = 2, 3, \dots, n \end{cases} \quad (6.33)$$

$$\begin{cases} x^{(n)} = y^{(n)} \\ x^{(j)} = y^{(j)} - \Gamma_j x^{(j+1)}, j = n-1, n-2, \dots, 1. \end{cases} \quad (6.34)$$

This completes the algorithm for solving this problem.

6.5 TWO-POINT BOUNDARY VALUE PROBLEMS

We introduce *two-point boundary value problems* (TPBVP) by formulating the most general case. Consider a bounded interval (a, b) with $a < b$, and let f be some non-linear function that depends on three variables. Then the goal is to find a function $u = u(x)$ in the interval (a, b) that satisfies the second-order equation:

$$u'' = f(x, u, u') \text{ on } (a, b) \quad (6.35)$$

in conjunction with the boundary conditions:

$$\begin{aligned} \alpha_0 u(a) + \alpha_1 u'(a) &= \alpha \quad |\alpha_0| + |\alpha_1| \neq 0 \\ \beta_0 u(b) + \beta_1 u'(b) &= \beta \quad |\beta_0| + |\beta_1| \neq 0. \end{aligned} \quad (6.36)$$

In this latter equation (6.36), all parameters are known with the exception of the (unknown) solution u and its derivative.

Equation (6.35) is non-linear, and in general we say that devising robust numerical schemes for such equations in combination with the boundary conditions (6.36) is a real challenge in general. The problem may have no solution, or it may even have multiple solutions. Fortunately, the situation is a bit simpler in this book because we examine a class of *linear* TPBVP that is general enough to model problems in financial engineering.

In this chapter we examine a special linear case of Equation (6.35):

$$Lu \equiv -u'' + p(x)u' + q(x)u = r(x). \quad (6.37)$$

This is a linear equation, and it is general enough for most of our work. If we take the special case of $p(x) = q(x) = r(x) = 0$ and if we then integrate the equation $u'' = 0$ twice, we see that the solution has the form:

$$u(x) = Ax + B$$

where A and B are integration constants. We conclude that we must give two extra conditions in order to specify a unique solution. To this end, we usually specify one constraint at $x = a$ and another constraint at $x = b$. There are four main possibilities:

- Dirichlet boundary conditions:

$$u(a) = \alpha, \quad u(b) = \beta. \quad (6.38)$$

- Neumann conditions:

$$u'(a) = \alpha, u'(b) = \beta. \quad (6.39)$$

- Linearity boundary condition (see Tavella and Randall (2000), p. 122):

$$u''(a) = \alpha, u''(b) = \beta. \quad (6.40)$$

- Robin boundary condition: a combination of Dirichlet and Neumann boundary condition.

Of course, we can give combinations of the above boundary conditions. For example, we can give a Dirichlet condition at $x = a$ and a Neumann condition at $x = b$ and so on. Equation (6.40) is used when a derivative quantity has a payoff that is at most linear in the underlying. We are in fact saying that the option value is nearly linear as a function of the spot price. Note that the pure Neumann boundary conditions (6.39) do not always produce a solution to the corresponding boundary value problem.

6.5.1 Finite Difference Approximation

We divide the interval (a, b) into J equal subintervals, and we let h be the length of each subinterval. Thus, we set up a finite difference scheme on a uniform mesh by approximating the first- and second-order derivatives of u at each mesh point:

$$\begin{aligned} L_h u_j &\equiv -\left(\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2}\right) + p(x_j)\left(\frac{u_{j+1} - u_{j-1}}{2h}\right) + q(x_j)u_j = r(x_j), \quad 1 \leq j \leq J-1 \\ u_0 &= \alpha, u_J = \beta \end{aligned} \quad (6.41)$$

where we have taken Dirichlet conditions for the moment. Grouping terms in (6.41), we can then write the equations as a matrix system:

$$AU = r$$

$$A = \begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & \ddots & \ddots & & & 0 \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ 0 & & \ddots & \ddots & c_{J-1} & \\ & & & a_J & b_J & \end{pmatrix} \quad (6.42)$$

where the coefficients of the matrix A are given by:

$$\begin{aligned} a_j &\equiv -\frac{1}{2} \left[1 + \frac{h}{2} p(x_j) \right] \\ b_j &\equiv \left[1 + \frac{h^2}{2} q(x_j) \right] \\ c_j &\equiv -\frac{1}{2} \left[1 - \frac{h}{2} p(x_j) \right] \end{aligned} \quad (6.43)$$

with $1 \leq j \leq J - 1$ and the vectors U and r are given by:

$$U = (u_1, \dots, u_{J-1})^\top$$

$$r = \begin{pmatrix} r_1 \\ \vdots \\ r_{J-1} \end{pmatrix} = \frac{h^2}{2} \begin{pmatrix} r(x_1) \\ \vdots \\ r(x_{J-1}) \end{pmatrix} - \begin{pmatrix} a_1\alpha \\ \vdots \\ c_J\beta \end{pmatrix}. \quad (6.44)$$

Incidentally, the reader can check how the boundary conditions have been incorporated into the vector r in (6.44).

We call scheme (6.41) a standard difference scheme because it approximates the coefficients and derivatives using conventional divided difference schemes. We shall see that the numerical solution can exhibit *oscillatory behaviour* precisely because we approximate the first-order derivative (convection term) by *centred divided differences*. Unfortunately, this approach is used in financial engineering literature, and authors tend to use workarounds to avoid *spurious oscillations*. We discuss this topic in greater detail in Chapters 14 and 15.

Another sufficient condition for (6.41) to have a unique solution is that the matrix in (6.42) be diagonally dominant, that is:

$$|b_j| > |a_j| + |c_j|, \quad j = 1, 2, \dots, J \quad (a_1 \equiv c_1 \equiv 0). \quad (6.45)$$

In this case the matrix A is non-singular, and (6.42) will have a unique solution.

Can we give sufficient conditions for the system (6.42) to have a unique solution, and can we bound the solution by its input? We state the following:

Assume $|p(x)| \leq P^*$ and:

$$0 \leq Q_* \leq q(x) \leq Q^*, \quad a \leq x \leq b. \quad (6.46)$$

Let $h < \frac{2}{P^*}$. Then scheme (6.41) has a unique solution.

We see that the coefficient of the first order derivative in (6.37) plays an important role. If it has large values, then the mesh-size h must be small. It turns out in practice that if this is not so then spurious oscillations arise in the numerical solution, as is borne out in the engineering and financial engineering literature.

The scheme (6.41) is second-order accurate when the mesh-size h is constant (otherwise it is only first-order accurate!) and if the conditions in (6.46) are true. We can apply extrapolation techniques to improve the accuracy of the solution to fourth-order. To this end, we create meshes of sizes h and $h/2$ and solve the finite difference equations on each one. The result:

$$u_j(h) - u(x_j) = h^2 e(x_j) + O(h^4)$$

then

$$\bar{u}_j \equiv \frac{4}{3} u_{2j}(h/2) - \frac{1}{3} u_j(h) \quad (6.47)$$

is a fourth order approximation to the solution of (6.37).

6.5.2 Approximation of Boundary Conditions

We now discuss how to approximate boundary conditions using finite differences. The main challenge lies in deciding how to approximate the first-order and possibly second-order derivatives of the unknown solution at the end-points. We concentrate on the general Robin boundary conditions for the moment (notice that it includes the Dirichlet and Neumann conditions as special cases). Furthermore, we look at the problem at the left-hand boundary $x = a$.

There are two main techniques for approximating the Robin boundary condition:

- Taking one-sided differences to approximate the first order derivative.
- By defining *ghost points* and using centred differences to approximate the first derivative at the boundary point.

Let us be specific by examining at the boundary condition at $x = a$:

$$\alpha_0 u(a) + \alpha_1 u'(a) = \alpha. \quad (6.48)$$

We take a one-sided approximation to the first-order derivative to get the discrete Robin boundary condition:

$$\begin{cases} \alpha_0 u_0 + \frac{\alpha_1}{h}(u_1 - u_0) = \alpha \\ \text{or } (\alpha_0 h - \alpha_1)u_0 + \alpha_1 u_1 = \alpha h. \end{cases} \quad (6.49)$$

This is a first-order approximation, and it can (potentially) destroy the second-order accuracy of scheme (6.41). In short, the first-order accuracy that we achieve at the boundary can percolate, as it were, into the interior of the interval (a, b) . In order to preserve second-order accuracy at the boundary, we introduce so-called *ghost* or *fictitious points*. These are points that fall just outside the interval (a, b) . Then we approximate the boundary condition (6.48) at $x = a$ by the following centred scheme:

$$\begin{cases} \alpha_0 u_0 + \frac{\alpha_1}{2h}(u_1 - u_{-1}) = \alpha \\ 2h\alpha_0 u_0 + \alpha_1(u_1 - u_{-1}) = 2h\alpha. \end{cases} \quad (6.50)$$

We now eliminate the value at the ghost point from (6.50) by assuming that the difference scheme (6.41) is satisfied at $j = 0$. This leads to the difference equation:

$$-\left(\frac{u_1 - 2u_0 + u_{-1}}{h^2}\right) + p(x_0)\left(\frac{u_1 - u_{-1}}{2h}\right) + q(x_0)u_0 = r(x_0). \quad (6.51)$$

We then use the expression for the value at the ghost point from (6.50) in (6.51), thus eliminating the ghost point entirely from the system of equations while at the same time preserving second-order accuracy. In other words, we have two equations for two unknowns. This technique can be applied to parabolic partial differential equations in general and to the Black–Scholes equation in particular. On the other hand, it is a bit of a fudge in our opinion.

We now discuss how to approximate the boundary conditions (6.40). The approach is similar to that taken for Robin boundary conditions, and in this case we have:

$$\frac{u_1 - 2u_0 + u_{-1}}{h^2} = \alpha. \quad (6.52)$$

6.6 ITERATIVE MATRIX SOLVERS

In this section we take a completely different approach to solving matrix systems. Here we start with some initial approximation to the solution of the matrix system, and we then construct a sequence of estimates where each estimate is closer to the exact solution than the previous estimate. We are thus in the realm of *iterative methods* for solving systems of linear equations. The main methods of interest are:

- The Jacobi method
- The Gauss–Seidel method
- Successive overrelaxation (SOR)
- The conjugate gradient method
- Projected SOR method (for matrix inequalities).

These methods are suitable for sparse matrices. Furthermore, we generalise these methods to solve *linear complementarity problems* (LCP) of the form:

$$\begin{aligned} Ax &\geq b, \quad x \geq c \\ (x - c) \cdot (Ax - b) &= 0. \end{aligned} \quad (6.53)$$

Here A is a square positive-definite matrix, b and c are given vectors, and we seek a solution x that satisfies the conditions in Equation (6.53). Here we speak of vector inequality; by definition, a vector v_1 is greater than a vector v_2 if each element in v_1 is greater than the corresponding element in v_2 . We shall propose an algorithm for solving Equation (6.53). The method was invented by Cryer (1979) and has gained some acceptance in the financial engineering world. This algorithm is called the *projected SOR method*.

6.6.1 Iterative Methods

In general we wish to find the solution of the linear system written in matrix form:

$$Ax = b. \quad (6.54)$$

For a definitive and very clear discussion, see Varga (1962). Let us rewrite matrix A in the following equivalent form:

$$A = D(L + I + U)$$

where D is the diagonal matrix with value zero everywhere except on the main diagonal, where the values are equal to the diagonal elements of A . The matrix I is the identity

matrix, U is upper triangular, and L is lower triangular. We can then rewrite the matrix equation $Ax = b$ in the equivalent form:

$$x = -Lx - Ux + D^{-1}b \text{ or } x = Bx + c \quad (6.55)$$

where $B = -(L + U)$ and $c = D^{-1}b$.

This equation contains the unknown x on both left- and right-hand sides. Here is the crux: we define a one-step sequence of vectors as follows:

$$x^{(k+1)} = Bx^{(k)} + c, \quad k = 0, 1, 2, \dots \quad (6.56)$$

Starting with some initial approximation to the solution, we hope that the sequence will converge to the exact solution x as k increases. We must prove that the sequence does converge; for more information, we refer again to Varga (1962). There are a number of ways to choose to implement the iteration in (6.56), and we shall discuss each one in turn.

6.6.2 Jacobi Method

The Jacobi method is the simplest iterative method. The terms Lx and Ux in (6.55) are both evaluated at level k , and the Jacobi scheme in matrix form is given by:

$$x^{(k+1)} = -(L + U)x^{(k)} + D^{-1}b$$

where k is the *iteration number*, $k = 0, 1, 2, \dots$ or in component form:

$$x_j^{(k+1)} = \frac{-\sum_{i=1}^n a_{ji}x_i^{(k)} + b_j}{a_{jj}}, \quad j = 1, \dots, n.$$

We usually take the initial approximation to be that vector all of whose values are zero. With this method, we do not use the improved values until after a complete iteration.

6.6.3 Gauss–Seidel Method

The Gauss–Seidel method is similar to the Jacobi method, except that the term Lx is evaluated at the level $k + 1$:

$$x^{(k+1)} = -Lx^{(k+1)} - Ux^{(k)} + D^{-1}b.$$

In component form, Gauss–Seidel is:

$$x_j^{(k+1)} = \frac{-\sum_{i=1}^{j-1} a_{ji}x_i^{(k+1)} - \sum_{i=j+1}^n a_{ji}x_i^{(k)} + b_j}{a_{jj}}, \quad j = 1, \dots, n. \quad (6.57)$$

We can rewrite (6.57) to produce a more algorithmic depiction that is suitable for C++ development:

$$\begin{aligned} x_j^{(k+1)} &= x_j^{(k)} + r_j^{(k)} \\ r_j^{(k)} &= \frac{-\sum_{i=1}^{j-1} a_{ji}x_i^{(k+1)} - \sum_{i=j+1}^n a_{ji}x_i^{(k)} + b_j}{a_{jj}}, \quad j = 1, \dots, n \\ \text{STOP? } \|r_j^{(k)}\| &= \text{TOL}. \end{aligned} \tag{6.58}$$

Notice that, in contrast to the Jacobi method, the Gauss–Seidel method uses the improved values as soon as they are computed. This is reflected in equation (6.58).

6.6.4 Successive Over-Relaxation (SOR)

By a simple modification of the Gauss–Seidel method, it is often possible to make a substantial improvement in the *rate of convergence*, by which we mean the speed with which the sequence of approximations converges to the exact solution x of (6.54). To this end, we modify (6.58) by introducing a so-called *relaxation parameter* ω as a coefficient of the residual term:

$$x_j^{(k+1)} = x_j^{(k)} + \omega r_j^{(k)}, \quad j = 1, \dots, n. \tag{6.59}$$

For $\omega = 1$ we get the Gauss–Seidel method as a special case. A major result is:

SOR converges if $0 < \omega < 2$.

Furthermore, it has also been shown by experiment that for a suitably chosen ω , the number of approximations needing to be computed may be reduced by a factor of 100 in some cases. Indeed, for certain classes of matrices this optimal value is known; see Varga (1962) for more information.

6.6.5 Other Methods

For completeness, we mention some other important methods.

6.6.5.1 Conjugate Gradient Method

The conjugate gradient method is a direct method that is useful in practice. We assume that A is positive definite, having n rows and n columns. We start with an initial vector U_0 . We compute:

$$\left. \begin{array}{l} r_0 = F - AU_0 \\ p_0 = r_0 \end{array} \right\}.$$

For $j = 1, \dots, n$ we compute:

$$a_j = \|r_j\|_2^2 / (p_j^\top A p_j)$$

$$U_{j+1} = U_j + a_j p_j$$

$$r_{j+1} = r_j - a_j A p_j$$

$$b_j = \frac{\|r_{j+1}\|_2^2}{\|r_i\|_2^2}$$

$$p_{j+1} = r_{j+1} + b_j p_j.$$

Then U_n will be the solution of the linear system $AU = F$ if rounding errors are neglected.

6.6.5.2 The Linear Complementarity Problem (LCP) and Projected SOR (PSOR)

We now give an introduction to solving problems as shown in Section 6.6 and Equation (6.53). These are the so-called *linear complementarity problem* (LCP) methods, and they arise in financial engineering applications when we discretise Black–Scholes equations with an early exercise option. For the moment, we present the projected SOR algorithm (PSOR) that solves (6.3):

1. Choose: $x^{(0)} \geq c$
2. $y_j^{(k+1)} = \frac{1}{a_{jj}} \left(b_j - \sum_{i=1}^{j-1} a_{ji} x_i^{(k+1)} - \sum_{i=j+1}^n a_{ji} x_i^{(k)} \right)$
 $x_j^{(k+1)} = \max(c_j, x_j^{(k)} + \omega(y_j^{(k+1)} - x_j^{(k)}))$
3. Check: $\|x^{(k+1)} - x^{(k)}\| \leq \text{TOL}.$

If A is positive definite, then PSOR converges for all initial $x^{(0)}$ if and only if $0 < \omega < 2$.

6.7 EXAMPLE: ITERATIVE SOLVERS FOR ELLIPTIC PDEs

We conclude this chapter with a discussion on how to approximate Poisson's equation in the unit square with Dirichlet boundary conditions:

$$\begin{aligned} \Delta u &= \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \quad (x, y) \in \Omega \\ u &= g(x, y) \text{ on } \Gamma \end{aligned} \tag{6.61}$$

where $\Omega = \{(x, y) : 0 < x < 1, 0 < y < 1\}$, $\Gamma = \text{boundary of } \Omega = \partial\Omega$.

We create a partition of the square into equal intervals of size h in both the x and y directions:

$$(x_i, y_j), \quad i, j = 0, \dots, J \quad h = 1/J, \text{ that is } x_i = \frac{i}{J}, \quad y_j = \frac{j}{J}. \quad (6.62)$$

The corresponding finite difference scheme is:

$$\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} + \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h^2} = f_{ij} \quad 1 \leq i, j \leq J-1. \quad (6.63)$$

We need to convert this problem into a matrix system (labelled by a single index). To this end, we can apply a *row-major strategy* to associate interior mesh point (1,1) with index 1, interior mesh point (1,2) with index 2, and so on. We eventually end up with a matrix system of the form:

$$AU = F. \quad (6.64)$$

We assume that the matrix A is non-singular, and it is possible to solve the system by LU decomposition as already discussed in this chapter. The matrix A is both sparse (most entries are zero) and banded (some set of contiguous diagonals contains all the non-zero entries), symmetric and/or block diagonal; efficient direct methods exploit any such special properties. Direct methods produce exact solutions, and they tend to need more computer storage than iterative methods. See Davis (2006).

Iterative methods use multiple indices to identify the unknowns; however, single indexing is used when discussing convergence of iterative methods. We focus on the point iteration variants of the Jacobi and Gauss–Seidel methods to approximate the solutions of (6.63):

They are respectively (the index k is the iteration index):

$$U_{ij}^{k+1} = (U_{i-1,j}^k + U_{i,j-1}^k + U_{i+1,j}^k + U_{i,j+1}^k + f_{ij})/4 \quad (6.65)$$

$$U_{ij}^{k+1} = (U_{i-1,j}^{k+1} + U_{i,j-1}^{k+1} + U_{i+1,j}^k + U_{i,j+1}^k + f_{ij})/4. \quad (6.66)$$

In general, row and block methods have better convergence properties than point methods, while Gauss–Seidel is more efficient than Jacobi.

For more information on matrix iterative methods, see Varga (1962), Golub and van Loan (1996), Hageman and Young (1981), Thomas (1999) and especially Duchateau and Zachman (1986) if you are a novice in this area.

6.8 SUMMARY AND CONCLUSIONS

We have given a self-contained overview of the matrix solvers that we use in this book. The focus was on solving linear systems of equations and in particular tridiagonal matrix systems, as these are very common when we approximate PDEs by the finite difference method.

One final remark which is really an exercise!

Generalise the Double Sweep method from one that supports Dirichlet boundary conditions to one that supports *Robin boundary conditions*:

$$\begin{cases} a_j U_{j-1} + b_j U_j + c_j U_{j+1} = f_j, & 1 \leq j \leq J-1 \\ U_0 = \alpha U_1 + \varphi \\ U_{J-1} = \beta U_J + \psi \end{cases}$$

where α, β, φ and ψ are given constants.

Test the new algorithm by discretising the two-point boundary value problem:

$$\begin{cases} \frac{d^2u}{dx^2} = f(x), & 0 < x < 1 \\ u(0) = \varphi \\ \frac{du(1)}{dx} + \beta u(1) = \psi \end{cases}$$

using the following finite difference scheme:

$$\begin{cases} U_{j+1} - 2U_j + U_{j-1} = h^2 f_j, & 1 \leq j \leq J-1 \\ U_0 = \varphi \\ \frac{U_J - U_{J-1}}{h} + \frac{\beta}{2}(U_J + U_{J-1}) = \psi. \end{cases} \quad (6.67)$$

We note that the *averaged approximation* at the boundary $x = 1$ is second-order accurate.

Answer the following questions:

- a. Modify the Double Sweep algorithm so that it can be applied to problems with Robin boundary conditions.
- b. Test the new code on the above two-point boundary value problem. Assemble the system of equations, and determine the conditions under which the tridiagonal matrix is diagonally dominant, hence ensuring that the system has a unique solution. When is the matrix an M-matrix?
- c. Solve the two-point boundary value problem using the Thomas algorithm, and compare the efficiency and accuracy with the results in part b). How much faster is Double Sweep than the Thomas algorithm, if at all?
- d. Consider first-order approximation at $x = 1$:

$$\frac{U_J - U_{J-1}}{h} + \beta U_J = \psi.$$

Compare its accuracy against the second-order accuracy scheme in (6.67). What is the Double Sweep algorithm in the case?

- e. A sub-project is to investigate whether first-order accuracy on the boundary negativity impairs the global second-order accuracy in the interior of the domain.

The second-order approximate boundary condition, as in (6.67), was used in Sheppard (2007) for the PDE representing the Heston model.

Black–Scholes Finite Differences for the Impatient

It is by logic that we prove, but by intuition that we discover.

—Henri Poincaré

7.1 INTRODUCTION AND OBJECTIVES

In this chapter we introduce several popular finite difference schemes for the partial differential equations for the one-factor heat and Black–Scholes equations. The reason for doing this before taking a more pedagogical style in later chapters is to acquaint the reader with the syntax and notation of numerical analysis as applied to the approximation of PDEs by finite differences. We have tried to avoid too much unnecessary detail that would detract from what we are trying to achieve, namely learning finite differences. Don't worry if you don't understand everything. We shall revisit the topics in later chapters.

Please note that some authors (for example, Hull (2006)) write their PDEs as a *terminal value problem*, that is from $t = T$ (expiration) to $t = 0$ (now). Their rationale is to treat PDEs and lattice models in the same way. This causes no end of confusion, because the standard protocol in mathematics and engineering is to write PDEs as an *initial value problem*, that is from $t = 0$ (now) to $t = T$ (expiration).

If you are a novice, then one way to prepare for later chapters is to read and browse this chapter a few times in order to acquaint yourself with the way of working and gaining familiarity with the notation. Ideally, programming the algorithms in C++ or (maybe even better) in Python represents the icing on the cake as it were (see Duffy (2018)). Finite difference schemes should work both on paper and in the computer. The discussion of the trinomial method in section 7.3 is a first step to understanding the finite difference method.

7.2 THE BLACK-SCHOLES EQUATION: FULLY IMPLICIT AND CRANK-NICOLSON METHODS

We introduce a number of finite difference schemes that are used to approximate the solution of the Black–Scholes equation. We concentrate on *implicit schemes* where the solution of the finite difference equation at time level $n + 1$ is calculated from the solution at time level n by solving a system of linear equations. Since we are using *three-point difference* schemes for one-factor models the matrix appearing in the system is tridiagonal. In this case we can apply either the Double Sweep method or *LU* decomposition (that we discussed in chapter 6) for the solution at each time level:

$$\begin{cases} A^{n+1}U^{n+1} = F^n \\ U^0 \text{ given} \end{cases}$$

where:

$$A^n = \begin{pmatrix} b_1^n & c_1^n & & & & \\ a_2^n & & & & & \\ & \ddots & \ddots & \ddots & 0 & \\ & & \ddots & \ddots & \ddots & \\ 0 & & & \ddots & \ddots & c_{J-1}^n \\ & & & & \ddots & \\ a_J^n & b_J^n & & & & \end{pmatrix}. \quad (7.1)$$

This system of equations is generic and we shall discuss two particularly important implicit schemes that lead to such matrix systems:

- The Euler, fully implicit method.
- The centred difference (or Crank–Nicolson) method.

The difference between these methods lies in how they calculate the solution at time level $n + 1$ in terms of the solution at level n .

Both the fully implicit and Crank–Nicolson methods use centred divided differences to approximate the first and second derivatives of the option price with respect to S while they employ one-step differencing in time t .

How it is done precisely is a generalisation of the methods from chapters 2 and 3.

Please note that we have cast the Black–Scholes equations as a PDE with an *initial condition* rather than a *terminal condition*. Thus, the values of the solution are known at time level n , and we *march* to time level $n + 1$ in order to compute a solution. In the financial literature, we *sometimes* march from a terminal value at level $n + 1$ to the solution at level n as already mentioned in the Introduction.

7.2.1 Fully Implicit Method

We begin with the fully implicit method for the Black–Scholes partial differential equation:

$$-\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0. \quad (7.2)$$

Of course, this equation must get auxiliary information, such as *initial condition (payoff)* and *numerical boundary conditions* in order to have a unique solution and furthermore, we must decide how we are going to define numerical approximations to these auxiliary conditions. For the moment, however, we ignore boundary conditions and concentrate on Equation (7.2) and its approximation by finite differences. To this end, we discretise (S,t) space in the usual way, and we approximate (7.2) by the fully implicit scheme (ΔS and k are mesh sizes in S and t directions, respectively):

$$\begin{aligned} & -\frac{V_j^{n+1} - V_j^n}{k} + \frac{1}{2}\sigma^2 j^2 \Delta S^2 \left(\frac{V_{j+1}^{n+1} - 2V_j^{n+1} + V_{j-1}^{n+1}}{\Delta S^2} \right) + rj\Delta S \left(\frac{V_{j+1}^{n+1} - V_{j-1}^{n+1}}{2\Delta S} \right) \\ & - rV_j^{n+1} = 0. \end{aligned} \quad (7.3)$$

The *stencil* for this scheme is given in Figure 7.1, and we thus see that the scheme is implicit because we have to solve for three unknowns at time $n+1$

$$a_j^{n+1} V_{j-1}^{n+1} + b_j^{n+1} V_j^{n+1} + c_j^{n+1} V_{j+1}^{n+1} = F_j^{n+1} \quad (7.4)$$

where

$a_j^{n+1}, b_j^{n+1}, c_j^{n+1}, F_j^{n+1}$ are known quantities.

7.2.2 Crank–Nicolson Method

The fully implicit scheme has a number of desirable features. First, it is stable, and there is no restriction on the relative sizes of the time mesh size k and the space mesh size ΔS . Furthermore, no spurious oscillations are to be seen in the solution (as is the case with some other methods). A disadvantage is that it is only first-order accurate in k . On the other hand, this problem can be rectified by using extrapolation, and this results in a second-order scheme. We now discuss the Crank–Nicholson method for problem (7.2).

We also use Δt to denote the step size in time when it suits us.

In this case we can view Crank–Nicolson as the averaged scheme by adding the fully explicit and fully implicit schemes together. The stencil in Figure 7.2 motivates

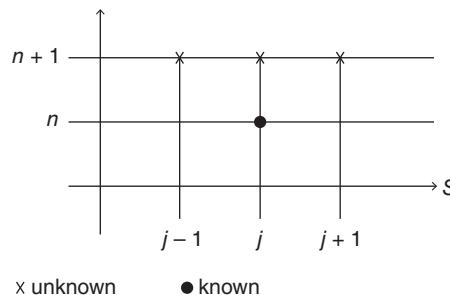


FIGURE 7.1 Stencil for fully implicit scheme.

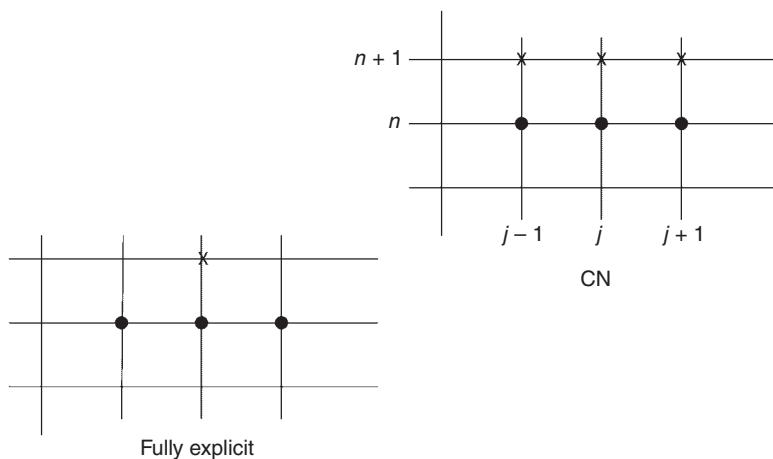


FIGURE 7.2 Crank–Nicolson and fully explicit stencils.

the relationship. We also use the variable h to denote the step size in space (thus, h is the same as ΔS).

We define the quantity:

$$V_j^{\frac{n+1}{2}} \equiv \frac{1}{2}(V_j^{n+1} + V_j^n).$$

Then the Crank–Nicolson method is defined as follows:

$$\begin{aligned} & -\frac{V_j^{n+1} - V_j^n}{k} + rj\Delta S \left(\frac{V_{j+1}^{\frac{n+1}{2}} - V_{j-1}^{\frac{n+1}{2}}}{2\Delta S} \right) + \frac{1}{2}\sigma^2 j^2 \Delta S^2 \left(\frac{V_{j+1}^{\frac{n+1}{2}} - 2V_j^{\frac{n+1}{2}} + V_{j-1}^{\frac{n+1}{2}}}{\Delta S^2} \right) \\ & - rV_j^{\frac{n+1}{2}} = 0. \end{aligned} \quad (7.5)$$

Again, this is a system that can be posed in the forms (7.4) or (7.1) and hence can be solved by standard matrix solver techniques for each time level.

The Crank–Nicolson method has gained wide acceptance in the financial literature, and it seems to be one of the de facto finite difference schemes for one-factor and two-factor Black–Scholes equations. It has second-order accuracy in the parameter k , and it is stable. Unfortunately, it has been known for some time (Il'in (1969)) that centred differencing schemes in space combined with averaging in time (what essentially CN is in this context) leads to spurious oscillations in the approximate solution and the divided differences for approximating its derivatives. These oscillations have nothing to do with the physical or financial problem that the scheme is approximating.

There are two kinds of independent variables associated with the one-factor Black–Scholes, as can be seen in (7.2). These correspond to the x (same as S) and

t variables. We concentrate on the x direction for the moment. We discretise in this direction using centred differences at the point (jh, nk) :

$$\begin{aligned}\frac{\partial^2 u}{\partial x^2} &\sim \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2} \\ \frac{\partial u}{\partial x} &\sim \frac{u_{j+1}^n - u_{j-1}^n}{2h}.\end{aligned}\quad (7.6)$$

Using this knowledge, we can apply the Crank–Nicolson method to (7.2):

$$-\frac{u_j^{n+1} - u_j^n}{k} + \sigma_j^{n+\frac{1}{2}} \frac{u_{j+1}^{n+\frac{1}{2}} - 2u_j^{n+\frac{1}{2}} + u_{j-1}^{n+\frac{1}{2}}}{h^2} + \mu_j^{n+\frac{1}{2}} \frac{u_{j+1}^{n+\frac{1}{2}} - u_{j-1}^{n+\frac{1}{2}}}{2h} + b_j^{n+\frac{1}{2}} u_j^{n+\frac{1}{2}} = 0 \quad (7.7)$$

when $\sigma_j^{n+1/2} = \frac{1}{2}\sigma^2 S_j^2$, $\mu_j^{n+1/2} = rS_j$, $b_j^{n+1/2} = -r$. (In this case these coefficients are independent of time.)

A bit of simple arithmetic allows us to rewrite (7.7) in the form:

$$a_j^n u_{j-1}^{n+1} + b_j^n u_j^{n+1} + c_j^n u_{j+1}^{n+1} = F_j^n \quad (7.8)$$

where a_j^n , b_j^n , c_j^n and F_j^n are known quantities. (Note that $u_j^{n+1/2} \equiv \frac{1}{2}(u_j^n + u_j^{n+1})$.)

This system of equations can be posed in the form of a matrix system as in (7.1). A critical observation is that if the coefficient a_j^n appearing in (7.8) is not positive, then the resulting solution will show oscillatory behaviour. The requirement that this coefficient be positive reduces to the inequality (for generic parameters σ and μ in (7.13)):

$$h \leq \left| \frac{2\sigma}{\mu} \right|. \quad (7.9)$$

This means that h must be chosen so as to satisfy this inequality. This will give problems in general for some applications where the volatility is a decaying function of time, for example:

$$\sigma(t) = \sigma_0 e^{-\alpha(T-t)} \quad (7.10)$$

where σ_0 and α are given constants.

Furthermore, for the standard Black–Scholes equation, the restriction on h is:

$$h \leq \frac{\sigma^2 S}{r}. \quad (7.11)$$

The quantity on the right-hand side of (7.11) or (7.9) is sometimes called the *Reynolds number*, and it is a fundamental quantity when dealing with finite difference schemes for *convection-diffusion* equations, in particular, for *convection-dominated* problems.

We speak of a *singular perturbation* problem associated with problem (7.2) when the coefficient of the second derivative is small. In this case traditional finite difference schemes can perform badly at the so-called *boundary layer* situated at $x = 0$. In fact, if we formally set the diffusion term to zero in equation (7.7), we get a so-called weakly stable difference scheme (see Peaceman (1977)) that approximates the *first-order hyperbolic equation*:

$$-\frac{\partial u}{\partial t} + \mu \frac{\partial u}{\partial x} + bu = f. \quad (7.12)$$

This has the consequence that the initial errors in the scheme are not dissipated, and hence we can expect oscillations. We need other *one-sided schemes* in this degenerate case.

We can discuss the Crank–Nicolson method from a number of viewpoints. For convenience and generality reasons, we cast the Black–Scholes equation as a generic *parabolic initial boundary value problem* in the domain $D = (A, B) \times (0, T)$ where $A < B$:

$$\begin{aligned} Lu &\equiv -\frac{\partial u}{\partial t} + \sigma(x, t) \frac{\partial^2 u}{\partial x^2} + \mu(x, t) \frac{\partial u}{\partial x} + b(x, t)u = f(x, t) \text{ in } D \\ u(x, 0) &= \varphi(x), x \in \Omega = (A, B) \\ u(A, t) &= g_0(t), u(B, t) = g_1(t), t \in (0, T). \end{aligned} \quad (7.13)$$

In this case the time variable t corresponds to increasing time while the space variable x corresponds to the underlying asset price S . We specify *Dirichlet boundary conditions* on the finite space interval Ω and this is a common situation for several kinds of exotic options, for example barrier options. Then we can pose finite difference methods on (7.13).

7.2.3 Final Remarks

This section is meant as a global overview of two popular finite difference schemes for the Black–Scholes PDE (7.2). The goal was to get some insights into how to set up the finite difference schemes for this problem. We shall discuss these and other schemes in great detail as we progress in this book. We note the following:

- The Crank–Nicolson method (7.7) is stable and second-order accurate in space and time, which accounts for its widespread popularity in computational finance. It is *A-stable*, which means that oscillations can arise in the solutions and its sensitivities (greeks), especially for non-smooth payoffs. See Duffy (2004) for a critical discussion.
- The fully implicit scheme (7.3) is stable and second-order accurate in space and first-order accurate in time. It is *L-stable*, which means that oscillations do not arise in the solutions nor in their sensitivities (greeks).
- It is possible to use the fully implicit method for the first few time steps and Crank–Nicolson after that in order to avoid oscillations. This is called the Rannacher method (see Rannacher (1984)).

- The explicit Euler scheme is similar to (7.3) in which all terms (except that term approximating the time derivative) are evaluated at time level n instead of time level $n + 1$.
- Traditional finite difference methods that use centred differencing in space encounter problems when the constraint (7.11) or (7.9) is not met, that is for *convection-dominated* problems.
- We and others have seen that the Crank–Nicolson scheme is 40% less efficient than the *Alternating Direction Explicit* (ADE) method in some cases (Pealat and Duffy (2011), Larkin (1964)).

7.3 THE BLACK–SCHOLES EQUATION: TRINOMIAL METHOD

The Crank–Nicolson method has become a very popular finite difference scheme for solving the Black–Scholes equation in combination with centred differencing in space. This equation is an example of a *convection-diffusion* equation, and it has been known for some time that centred-difference schemes are inappropriate for approximating it (Il'in (1969), Duffy (1980)). In fact, many independent discoveries of novel methods have been made in order to solve difficult convection-diffusion problems in fluid dynamics, atmospheric pollution modelling, semiconductor equations, the Fokker-Planck equation and groundwater transport (Morton (1996)). The main problem is that traditional finite difference schemes start to oscillate when the coefficient of the second derivative (the *diffusion* term) is very small or when the coefficient of the first derivative (the *convection* term) is large. In this case, the mesh size h in the space direction must be smaller than a certain value if we wish to avoid these oscillations. This problem has been known since the 1950s. (See de Allen and Southwell (1955).)

7.3.1 Comparison with Other Methods

For completeness, we use the trinomial method for one-factor option pricing. In general terms we build a trinomial tree of asset prices (the *forward induction step*) using the stochastic differential equation (SDE) for the asset price. We build the tree up to expiration. Having done that, we calculate the option prices using discounted expectations (the *backward induction step*) starting from the payoff function at expiration.

We take a step-by-step approach to explaining the trinomial method. To this end, we assume that the geometric Brownian motion model holds for the asset price behaviour:

$$dS = (r - D)Sdt + \sigma SdW$$

where:

r = risk free interest rate

(7.14)

D = continuous dividend yield

σ = volatility

W = Brownian motion.

We now define the new variable $x = \log S$. We then get the modified SDE:

$$dx = vdt + \sigma dW, \quad v = r - D - \frac{1}{2}\sigma^2. \quad (7.15)$$

We now model (7.15) in a special way (see Figure 7.3). Let us consider what happens to the price x in a small interval of time Δt . We assume that x can take one of three values in this interval: it can go up or down by an amount Δx , or it can stay the same. Each transition is associated with a corresponding probability as shown in Figure 7.3, namely an up, down and no change. We find values for these probabilities, and this is based on a financial argument, namely the relationship between the continuous time and the trinomial process by equating the mean and variance over the time interval Δx and equating the sum of the probabilities to 1:

$$\begin{aligned} E[\Delta x] &= p_u(\Delta x) + p_m(0) + p_d(-\Delta x) = v\Delta t \\ E[\Delta x^2] &= p_u(\Delta x^2) + p_m(0) + p_d(\Delta x^2) = \sigma^2\Delta t + v^2\Delta t^2 \\ p_u + p_m + p_d &= 1. \end{aligned}$$

Let us define the following convenience parameters:

$$\alpha = \frac{v\Delta t}{\Delta x}, \quad \beta = \frac{\sigma^2\Delta t + v^2\Delta t}{\Delta x^2}$$

then a bit of arithmetic shows that:

$$p_u = \frac{\alpha + \beta}{2}, \quad p_d = \frac{\beta - \alpha}{2}, \quad p_m = 1 - \beta, \quad (p_u + p_d + p_m = 1).$$

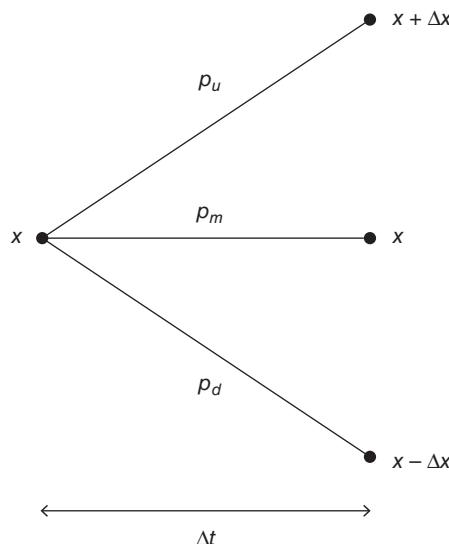


FIGURE 7.3 Trinomial tree model.

We now embark on the mechanics of the trinomial method based on SDE (7.14) and variable S . To this end, we prefer to use the index n to represent time and j to represent the index for the underlying variable S . (Much of the literature uses i and j for indices, but this makes no distinction between space and time variables. This work uses i, j , and k for space indices and n for time. The common practice of using j for the time index does not scale well to multi-dimensional problems and can make algorithms difficult to read.) If S is the price at time $n = 0$, then the price at level j is given by:

$$S_j^n = Se^{j\Delta x}.$$

We compute the array using a vector $Sarr$:

$$\begin{aligned} Sarr[-N] &= Se^{-N\Delta x} \\ Sarr[j] &= Sarr[j - 1]e^{\Delta x}, \quad j = -N + 1, \dots, N. \end{aligned} \tag{7.16}$$

Here N is the number of subdivisions of the interval $(0, T)$ where T is the expiration, that is $N\Delta t = T$. We model call options with price C , and its discrete values will be denoted in the same way as the stock price S , namely C_j^n .

The value of the call option is known at expiration, and the continuous and discrete variants are given by:

$$\begin{aligned} C(S, T) &= \max(S - K, 0) \\ C_j^N &= \max(S_j^N - K, 0). \end{aligned} \tag{7.17}$$

Finally, we compute the call option value at time n as discounted expectations based on the call option values at time $n + 1$ as follows:

$$C_j^n = e^{-r\Delta t}(p_u C_{j+1}^{n+1} + p_m C_j^{n+1} + p_d C_{j-1}^{n+1}) \tag{7.18}$$

where the probabilities are defined as above. Summarising this process as a computational algorithm:

- Create the trinomial tree structure.
- Initialise the stock values in the tree using formula (7.16).
- Compute the vector payoff, Equation (7.17).
- Compute the call values at previous time steps using Equation (7.18).

The binomial and trinomial methods are both examples of *lattice methods*. The binomial method is very popular, but it does have a number of shortcomings.

There is evidence to show that the binomial method with one underlying variable does not always produce accurate numerical results, and in this case the trinomial method is preferred. However, we realise that the trinomial method could be seen as an example of an explicit finite difference scheme, and the conclusion is that it is only conditionally stable. To this end, we now show that the standard explicit finite difference scheme for the Black–Scholes PDE is equivalent to performing discounted expectations in a trinomial tree. Let us first consider again the Black–Scholes PDE written as a terminal value problem (we use the variable x to denote the transformed stock price, that is $x = \log S$):

$$-\frac{\partial C}{\partial t} = \frac{1}{2}\sigma^2 \frac{\partial^2 C}{\partial x^2} + v \frac{\partial C}{\partial x} - rC. \quad (7.19)$$

We are marching from expiration down to time zero, and we construct the explicit finite difference approximation to (7.19) as follows:

$$-\frac{C_j^{n+1} - C_j^n}{\Delta t} = \frac{1}{2}\sigma^2 \frac{C_{j+1}^{n+1} - 2C_j^{n+1} + C_{j-1}^{n+1}}{\Delta x^2} + v \frac{C_{j+1}^{n+1} - C_{j-1}^{n+1}}{2\Delta x} - rC_j^{n+1}.$$

Rearranging terms we get the following representation:

$$C_j^n = p_u C_{j+1}^{n+1} + p_m C_j^{n+1} + p_d C_{j-1}^{n+1} \quad (7.20)$$

where:

$$p_u = A + B, \quad p_m = 1 - 2A - r\Delta t, \quad p_d = A - B$$

and:

$$A = \frac{\Delta t \sigma^2}{2\Delta x^2}, \quad B = \frac{\Delta t v}{2\Delta x}.$$

This scheme is similar to taking discounted expectations. In general the free term rC evaluated at time level n (implicit) to improve stability. Of course the probabilities in equation (7.20) should be positive, and this leads to restrictions on the step size Δt .

We mention that the trinomial method can be used to find an approximate solution for American options. This is easy because it is a variation of the European case. In particular, we must check that the *free boundary condition* remains valid, and hence we must have for an American put option:

$$C_j^n = \max(C_j^n, K - S_j^n)$$

at each time level. In general, the relationship between the time steps in time and S is given by:

$$\Delta x = \sigma \sqrt{3\Delta t}.$$

If this relationship is not satisfied, then we will get negative values for the option price, something that is physically or financially not possible.

The trinomial method can be applied to a range of products such as equities, currencies, interest rates or any other quantity that can be described as a stochastic differential equation. Let us examine the general SDE:

$$dy = A(y, t)dt + B(y, t)dW$$

y = real-valued function

W = Brownian motion.

Furthermore, A and B are given non-linear functions of y in general.

As before, the variable y can rise, fall or remain at the same value in the interval Δt . Let:

$\varphi^+(y, t)$ = probability of a rise

$\varphi^-(y, t)$ = probability of a fall.

Then the mean of the change in y in the given timestep is:

$$(\varphi^+ - \varphi^-)\Delta y$$

where Δy = jump size in y and the variance is given by:

$$(\varphi^+(1 - \varphi^+ + \varphi^-)^2 + (1 - \varphi^+ - \varphi^-)(\varphi^+ - \varphi^-)^2 + \varphi^-(1 + \varphi^+ - \varphi^-)^2)\Delta y^2.$$

The mean and variance in the continuous-time variant are approximately:

$$A(y, t)\Delta t$$

$$B(y, t)^2\Delta t.$$

Equating like terms, we choose:

$$\begin{aligned} \varphi^+(y, t) &= \frac{1}{2} \frac{\Delta t}{\Delta y^2} (B(y, t)^2 + A(y, t)\Delta y) \\ \varphi^-(y, t) &= \frac{1}{2} \frac{\Delta t}{\Delta y^2} (B(y, t)^2 - A(y, t)\Delta y). \end{aligned} \tag{7.21}$$

This result allows us to create a trinomial tree for any SDE. Ideally, a good exercise would be to program this algorithm and test it in C++ with a range of SDEs. The disadvantage of this scheme is that it is only conditionally stable because Δy and $\sqrt{\Delta t}$ must be of the same order.

7.4 THE HEAT EQUATION AND ALTERNATING DIRECTION EXPLICIT (ADE) METHOD

We give a quick tour of the ADE family of methods that we discuss in more detail in Chapter 19 and in Duffy (2018). In practice we use the *Barakat–Clark variant* (this is a forward reference) of ADE to the one-dimensional heat equation with Dirichlet boundary conditions:

$$\begin{cases} \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, & 0 < x < 1, \quad t > 0 \\ u(0, t) = u(1, t) = 0, & t > 0 \\ u(x, 0) = f(x), & 0 \leq x \leq 1. \end{cases} \quad (7.22)$$

The ADE method was first introduced in computational finance by the author (Duffy (2009a)) and subsequently discussed in Duffy and Germani (2013), Duffy (2018) and Pealat and Duffy (2011).

7.4.1 Background and Motivation

In this section we discuss a one-factor, time-independent diffusion equation, and in particular we are interested in approximating the diffusion term by finite differences. To this end, we consider a partition $\{x_0, x_1, \dots, x_J\}$ with $x_0 = 0, x_J = 1$ of the unit interval as shown in Figure 7.4. We then consider approximating the diffusion operator as follows:

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial}{\partial x} \left(\frac{\partial u}{\partial x} \right) \approx \frac{\frac{u_{j+1} - u_j}{h_{j+1}} - \frac{u_j - u_{j-1}}{h_j}}{(h_j + h_{j+1})/2}.$$

The ADE method rests on this special way of approximating the second-order derivative of a function.

We can see this approximation as a divided difference of a divided difference, and in the case of a constant mesh size h it reduces to the well-known centred-difference formula:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{j+1} - u_j - u_j + u_{j-1}}{h^2} = \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2}.$$

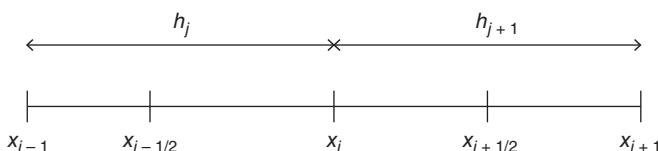


FIGURE 7.4 Non-uniform mesh.

7.5 ADE FOR BLACK–SCHOLES: SOME TEST RESULTS

We have tested ADE using C# and C++; see Duffy and Germani (2013) and Duffy (2018). We now report some results that were generated by C#. We discuss the ADE method in detail in Chapter 19.

Since ADE is a relatively new approach to solving PDEs in computational finance, we compare our results with some other methods (the method itself is more than 60 years old). To be precise, we compare the method with the exact solution (Haug (2007)), the implicit Euler method (Duffy (2006)) and the Monte Carlo method (Duffy and Kienitz (2009)). In general, it is possible to achieve any order of accuracy if we make the time steps small enough unless round-off errors are introduced. But there are also time constraints. For example, we compare the results of the ADE method with the Monte Carlo method with as many as a million draws or simulations, and it is obvious that the response time of the latter method is of the order of minutes and not of milliseconds.

We have tested the method for benign values of the Black–Scholes parameters as well as using more extreme values. We have used the *non-conservative* form of the transformed PDE.

The examples we take are:

- I: ADE for a European put option for a range of values of the mesh sizes k and h .
- II: Stress testing and extreme parameter values (for example, zero interest rate and large volatility).
- III: *Convection-dominated problems* (small diffusion and/or large drift) and exponentially fitted schemes.
- IV: *Constant elasticity of variance* (CEV) model with positive and negative elasticity factor.
- V: Option sensitivities (the Greeks).
- VI: Early exercise feature (American options).

We mention that these values have been computed in C++ using domain truncation as described in Kangro and Nicolaides (2000). These tests can also be applied to other finite difference methods, and not just ADE.

In all these cases we use the original Saul'yev scheme for the diffusion term (see Saul'yev (1964)), while we use the $O(\Delta t^2/h)$ Towler–Yang method for the convection term (Towler and Yang (1978)); we could have used the $O(\Delta t)$ Roberts–Weiss method (Roberts and Weiss (1966)). The examples presented here can be used as test input to other numerical schemes.

EXAMPLE 7.1

Example I: in this case we take an initial example to test the accuracy of ADE. We price a European put option with $K = 65$, $S = 60$, $T = 0.25$, $r = 0.08$, $\sigma = 0.30$. The exact put price is $P = 5.84584$ (call price $C = 2.13293$). Table 7.1 gives the approximate values using ADE for a range of mesh sizes. Of course, increasing the number of steps improves the accuracy. Finally, taking $NY = 1000$ and $NT = 2000$ gives $P = 5.845976$.

(continued)

(continued)

TABLE 7.1 Accuracy of ADE, put option.

NY/NT	100	200	500	1000
200	5.882045411	5.839649	5.844072	5.8413946
500	5.842051731	5.84398279	5.845058	5.845277

In Table 7.2 we show the results for the call option price with the same parameters as above.

TABLE 7.2 Accuracy of ADE, call option.

NY/NT	100	200	500
100	2.107727	2.126138	2.11275419
200	2.107969	2.1267947	2.131108
500	2.10807	2.11270118	2.132138

EXAMPLE 7.2

Example II: In this case we take $r = 0$ (zero interest rate), $K = S = 100$ (ATM) and a range of values for σ and T . We take $NX = NT = 500$ for all cases. We compare the exact and Monte Carlo solutions with the implicit Euler method and with the ADE method. See Table 7.3.

TABLE 7.3 Stress-testing ADE, put option.

	T = 1		T = 5	
	$\sigma = 0.2$	$\sigma = 1.0$	$\sigma = 0.2$	$\sigma = 1.0$
Exact P	7.96632	38.2893	17.6933	73.6463
IEuler	7.965621	38.27265	17.69072	73.63129
Monte Carlo	7.97947	38.322	17.7204	73.5974
ADE	7.963372	38.25255	17.68683	73.88137

EXAMPLE 7.3

Example III: Traditional finite difference schemes show signs of *spatial instability* (see Roache (1998)) when the *drift* (also known as the *convection term*) dominates the *diffusion term*. In order to resolve this problem we have developed *exponentially fitted schemes* (Duffy (1980)), and we subsequently applied them to option

pricing problems. We take an example in which we compare the exact solution of a convection-dominated problem with the ADE solution with and without fitting. The values are $r = 0.20$, $\sigma = 0.1$, $S = K = 100$, $T = 0.5$. We display both put and call prices. The exact values are $P = 0.238825$, $C = 9.75508$. See Table 7.4.

TABLE 7.4 Convection-dominated problems.

Put (0.238825) Call (9.75508)			
	No Fit	Fit	No Fit
200 × 200	.210608	.235072	9.726924
500 × 500	.234498	.238362	9.751284
1000 × 1000	.237825	.238789	9.754936

We see that the exponentially fitted ADE method has better accuracy properties than the unfitted ADE method for this problem.

EXAMPLE 7.4

Example IV: This is the test of the CEV model:

$$-\frac{\partial u}{\partial t} + \frac{1}{2}\sigma^2 S^{2\beta} \frac{\partial^2 u}{\partial S^2} + r \frac{\partial u}{\partial S} - ru = 0 \quad (7.23)$$

where β is the *elasticity factor*. We tested the ADE method using a range of values for the elasticity factor, some of which are shown in Table 7.5. The other parameters are:

$$r = 0.05, T = 0.5, K = 110, S = 100, \sigma = 0.2 (NY = 500, NT = 500).$$

TABLE 7.5 Testing the CEV Model, (put option).

β	Exact	ADE
0	9.95517	9.955171
-1	9.73787	9.737663
-2	9.53622	9.535646
2/3	10.1099	10.11005

EXAMPLE 7.5

Example V: We approximate option sensitivities, and we can report that the values obtained are as accurate as those from the implicit Euler scheme. Some results are presented in Table 7.6 based on $r = 0.05$, $T = 0.5$, $S = K = 100$, $\sigma = 0.2$. The cases ADE I and ADE II refers to mesh sizes $NY = 200$, $NT = 200$ and $NY = 400$, $NT = 400$, respectively. The case ADE III corresponds to $NY = 1000$, $NT = 1000$.

TABLE 7.6 Calculating sensitivities.

	Exact	ADE I	ADE II	ADE III
P	4.418995	4.414167	4.424243	4.420607
Δ	-0.40227	-0.43423	-0.41796	-0.40857
Γ	0.027359	0.029524	0.028439	0.027812

In Chapters 16 and 17 we discuss more robust ways to compute sensitivities.

EXAMPLE 7.6

Example VI: As last set of numerical examples, we examine the ability of the ADE method to price put options with early exercise feature (American options). We have compared the method with a number of other numerical methods from various sources. The basic algorithm for American options is based on that for European options. Having computed the two sweep solutions, we then test if their values are greater than the intrinsic value and the modified value will be the larger of itself and this intrinsic value.

As first example, we take a put option with $K = 10$, $S = 9$, $NY = 400$, $NT = 400$. Tables 7.7 and 7.8 show comparisons with the binomial method in the case when the number of steps $M = 256$. The results are in good agreement with the binomial method.

TABLE 7.7 Early exercise, $\sigma = 0.3$, $r = 0.06$.

T	Binomial (M = 256)	ADE
0.25	1.1260	1.126501
0.50	1.2553	1.2555837
0.75	1.3541	1.354917
1.0	1.4354	1.434963

TABLE 7.8 Early exercise, $\sigma = 0.5$, $r = 0.12$.

T	Binomial (M = 256)	ADE
0.25	1.3805	1.381034
0.50	1.6178	1.616883
1.0	1.9094	1.904257

Finally, we compare our results with implicit finite difference schemes for the CEV model in the case of the square root process. For this example, we take $S = 100$, $\beta = 0.5$, $r = 0.1$, $\sigma = .20$. We present the results for various values of the strike price and expiration in Table 7.9. In the table, the upper values are from another source while the lower values come from ADE. We witness good agreement ($NY = 200$, $NT = 400$).

Table 7.10 shows put option prices based on the implicit Euler method with $NT = 200$, $NY = 400$, $S = 100$, $r = 0.1$, $\sigma = 0.2$, $\beta = 0.5$ for a range of values of the strike price and expiry time.

In conclusion, the ADE is a fast and accurate numerical method for pricing American options. It is easier to model and program than penalty or front-fixing methods, and it is much faster. You can use the numerical results from Section 7.5 as input when performing initial testing of other finite difference methods in later chapters. In other words, the test results are ready to be used!

TABLE 7.9 Comparison of FDM methods.

K/T	1/2	1
90	1.06	1.81
	1.069352	1.824111
100	3.89	4.76
	3.893676	4.765286
110	10.20	10.55
	10.20316	10.55242

TABLE 7.10 Implicit Euler, early exercise.

K/T	1/2	1
90	1.05874	1.813477
100	3.884078	4.755361
110	10.19907	10.54583

7.6 SUMMARY AND CONCLUSIONS

In this chapter we introduced the popular Crank–Nicolson and fully implicit finite difference schemes for the PDE that models plain European options. We discussed a number of properties of these schemes, in particular some issues with the Crank–Nicolson scheme that may not be widely known in the computational finance community (see Duffy (2004)). We also introduced the trinomial method because it is well known. We can stretch the analogy by saying that the trinomial method is an explicit finite difference method, which it is not.

Mathematical Foundation for Two-Factor Problems

Classifying and Transforming Partial Differential Equations

A picture may be worth a thousand words, a formula is worth a thousand pictures.

Edsger Dijkstra

8.1 INTRODUCTION AND OBJECTIVES

The main goal of this chapter is to analyse the structure of two-dimensional elliptic convection-diffusion-reaction equations that represent the time-independent component in Black–Scholes-style PDEs, for example. The results that we present here are applicable to all the financial PDEs in this book, and it is for this reason that we discuss them here in a generic setting. In fact, they are also relevant to many other areas, such as fluid dynamics and heat transfer. We focus on second-order linear elliptic equations in two independent (space) variables. An important theme in this chapter is to transform a PDE containing a *mixed derivative term* to a PDE without such a term.

8.2 BACKGROUND AND PROBLEM STATEMENT

The most general linear time-independent *convection-diffusion-reaction* equation in n dimensions is given by:

$$\begin{aligned} Lu &\equiv \sum_{i,j=1}^n a_{ij}(x) \frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=1}^n b_i(x) \frac{\partial u}{\partial x_i} + c(x)u \\ x &= (x_1, \dots, x_n) \in \mathbb{R}^n. \end{aligned} \tag{8.1}$$

The coefficients a_{ij}, b_j and c are real, $a_{ij} = a_{ji}, i, j = 1, \dots, n$. For the moment, we place no further constraints on these coefficients. We see that Equation (8.1) has a *diffusion term* (second-order derivatives), a *convection term* (also known as *advection*) (first-order derivatives) and a *reaction term* (zero-order derivative) and Equation (8.1) is a central component in all computational finance applications. We concentrate on the structure of *strongly and uniformly elliptic operators*. In succeeding chapters we discuss existence and uniqueness of the solution of elliptic boundary value problems, their qualitative properties, boundary conditions and some well-known examples of elliptic equations. An elliptic PDE in which there is no mixed term is said to be in *canonical form*.

8.3 INTRODUCTION TO ELLIPTIC EQUATIONS

In the sequel we shall take $n = 2$ in Equation (8.1) for convenience. We are particularly interested in certain properties of the *diffusion matrix* $A = (a_{ij}), i, j = 1, \dots, n$. In general, the diffusion terms are more important than the convection and reaction terms for our present objectives.

Elliptic operators can be seen as generalisation of the important *Laplace operator*:

$$\Delta u = 0, \text{ where } \Delta = \sum_{j=1}^n \frac{\partial^2}{\partial x_j^2}.$$

The Laplace operator has many applications in fluid dynamics, electromagnetics and potential theory (Kellogg (1953)). It is a model for more general elliptic equations that we discuss in succeeding chapters. It is arguably the most influential PDE in all of mathematical physics. Studying it will give us insights into many branches of mathematics and its applications. A function that satisfies the Laplace equation is called *harmonic*.

8.3.1 What is an Elliptic Operator?

Let the operator L be defined as in Equation (8.1). It is defined by the condition that the coefficients of the highest-order second derivatives (not including the mixed derivative terms) be positive. This implies that the principal symbol is invertible, and this is equivalent to saying that there are no real characteristic directions, as we shall later see.

Formally, let D be an n -dimensional domain. L is said to be of *elliptic type* (*elliptic*) at a point $x^0 \in D$ if the matrix $(a_{ij}(x^0)), i, j = 1, \dots, n$ is *positive definite*, that is:

$$\sum_{i,j=1}^n a_{ij}(x^0) \xi_i \xi_j > 0 \text{ for } \xi \in \mathbb{R}^n, \xi = (\xi_1, \dots, \xi_n)^\top, \xi \neq 0. \quad (8.2)$$

In general, a solution of an elliptic equation will be smooth if the coefficients of its operator are smooth.

The operator L as already defined is *strongly elliptic* if for each $x \in \mathbb{R}^n$ the following inequality holds:

$$0 < \lambda(x) \|\xi\|^2 \leq \sum_{i,j=1}^n a_{ij}(x) \xi_i \xi_j \leq \Lambda(x) \|\xi\|^2, \quad \xi = (\xi_1, \dots, \xi_n)^\top$$

where:

$$\begin{cases} \lambda(x) = \text{smallest eigenvalue of matrix } A = (a_{ij}) \quad i, j = 1, \dots, n \\ \Lambda(x) = \text{largest eigenvalue of matrix } A \\ \|\xi\|^2 = \sum_{i=1}^n |\xi_i|^2. \end{cases} \quad (8.3)$$

We take an example of (8.1) in the case of an elliptic PDE with a mixed derivative term, and we calculate the eigenvalues of A :

$$\begin{aligned} \frac{\partial^2 u}{\partial u^2} + \frac{\partial^2 u}{\partial y^2} + \rho \frac{\partial^2 u}{\partial x \partial y} &= 0 \\ a_{11} = a_{22} &= 1; a_{12} = a_{21} = \rho/2 \\ A &= \begin{pmatrix} 1 & \rho/2 \\ \rho/2 & 1 \end{pmatrix}. \end{aligned}$$

The eigenvalues of A are $1 \pm \rho/2$.

8.3.2 Total and Principal Symbols

The *symbol* of a linear differential operator is a polynomial in several variables representing the operator. We replace each partial derivative by a new variable, one for each dimension. To this end, we consider a differential operator using general *multi-index notation*:

$$P = p(x, D) = \sum_{|\alpha| \leq 2} a_\alpha(x) D^\alpha$$

D = derivative, $\alpha = (\alpha_1, \alpha_2)$ (multi-index), $|\alpha| = \alpha_1 + \alpha_2$.

Then the *total* and *principal symbols* in the two-dimensional case are defined by:

$$\begin{aligned} p(x, \xi) &= \sum_{|\alpha| \leq 2} a_\alpha(x) \xi^\alpha \\ \sigma_p(\xi) &= \sum_{|\alpha|=2} a_\alpha(x) \xi^\alpha \end{aligned} \quad (8.4)$$

where $\xi \in \mathbb{R}^2$, $\xi^\alpha = \xi_1^{\alpha_1} \xi_2^{\alpha_2}$, $\alpha = (\alpha_1, \alpha_2)$,

respectively. The properties of the solution of an elliptic equation are largely determined by the principal symbol. For example, the principal symbol of an elliptic operator is nowhere zero. It is an important computational device for studying the singularities of these types of equations.

We now take an example of a *convection-diffusion-reaction* equation containing a mixed derivative term:

$$a \frac{\partial^2 u}{\partial x^2} + 2b \frac{\partial^2 u}{\partial x \partial y} + c \frac{\partial^2 u}{\partial y^2} + d \frac{\partial u}{\partial x} + e \frac{\partial u}{\partial y} + fu = g. \quad (8.5)$$

Then the total and principal symbols in this case are:

$$\begin{aligned} p(\xi_1, \xi_2) &= a\xi_1^2 + 2b\xi_1\xi_2 + c\xi_2^2 + d\xi_1 + e\xi_2 + f \\ \sigma_p(\xi) &= a\xi_1^2 + 2b\xi_1\xi_2 + c\xi_2^2. \end{aligned} \quad (8.6)$$

Another example is:

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \rho \frac{\partial^2 u}{\partial x \partial y} &= 0 \\ p(x, \xi) = \sigma_p(\xi) &= \xi_1^2 + \xi_2^2 + \rho\xi_1\xi_2. \end{aligned} \quad (8.7)$$

8.3.3 The Adjoint Equation

By definition, the adjoint equation corresponding to (8.1) is (Friedman (1992) p. 26):

$$L^*v \equiv \sum_{i,j=1}^n a_{ij}(x) \frac{\partial^2 v}{\partial x_i \partial x_j} + \sum_{i=1}^n b_i^*(x) \frac{\partial^2 v}{\partial x_i} + c^*(x)v$$

where:

$$b_i^* = -b_i + 2 \sum_{j=1}^n \frac{\partial a_{ij}}{\partial x_j}, \quad i = 1, \dots, n \quad (8.8)$$

$$c^* = c - \sum_{i=1}^n \frac{\partial b_i}{\partial x_i} + \sum_{i,j=1}^n \frac{\partial^2 a_{ij}}{\partial x_i \partial x_j}.$$

Some algebra shows that the adjoint equation can also be written as:

$$L^*v = \sum_{i,j=1}^n \frac{\partial^2}{\partial x_i \partial x_j} (a_{ij}v) - \sum_{i=1}^n \frac{\partial}{\partial x_i} (b_i v) + cv. \quad (8.9)$$

In the one-dimensional case, we show the equivalence of the two forms as follows:

$$Lv \equiv a \frac{d^2v}{dx^2} + b \frac{dv}{dx} + cv. \quad (8.10)$$

$$L^*v = \frac{d^2}{dx^2}(av) - \frac{d}{dx}(bv) + cv = a \frac{d^2v}{dx^2} + (2a' - b) \frac{dv}{dx} + (a'' - b' + c)v. \quad (8.11)$$

The *adjoint form* will be discussed in Chapter 13 in the context of stochastic processes.

8.3.4 Self-Adjoint Operators and Equations

We say that the operator is *self-adjoint* if $L = L^*$. We reduce the scope in this section, and we concentrate on second-order ODEs of the form:

$$\frac{d}{dx} \left(p(x) \frac{du}{dx} \right) + q(x)u = 0. \quad (8.12)$$

We can reduce second-order ODEs of the form:

$$a \frac{d^2u}{dx^2} + b \frac{du}{dx} + cu = 0 \quad (8.13)$$

to the form (8.12). We leave this as an exercise.

As last example, we take the elliptic part (time-independent) of the Black–Scholes equation and its adjoint form:

$$\frac{1}{2} \sigma^2 S^2 \frac{d^2u}{ds^2} + rS \frac{du}{ds} - ru = 0 \quad (8.14)$$

$$\begin{aligned} & \frac{d}{ds} \left(p(S) \frac{du}{ds} \right) + q(S)u = 0 \\ & p(S) = S^{2r/\sigma^2}, \quad q(S) = \frac{-2r}{\sigma^2} S^{2(\frac{r}{\sigma^2}-1)} \end{aligned} \quad (8.15)$$

for two known functions $p(S)$ and $q(S)$.

This approach is also applicable to the full time-dependent Black–Scholes equation and similar equations (for example, Cox–Ingersoll–Ross (CIR)), as we shall see in Chapter 25. It can also be applied to other two-factor models.

Are there advantages in trying to formulate a differential equation using form (8.12) rather than form (8.13)?

Some reasons are:

- The resulting PDE is simpler than the original PDE because the convection term has ‘disappeared’ as it were. It is also applicable to two-factor models.
- PDEs in adjoint form are suitable for numerical schemes such as the finite element method (FEM) or finite volume method (FVM) that take a variational approach to PDE (Aubin (1980), Achdou and Pironneau (2005), Strang and Fix (1973)).
- We do not have to worry about convection terms, especially at boundaries. In particular, convection dominance is less of an issue than for PDEs in non-adjoint form.
- Discretising PDEs using the finite difference method allows us to use *non-uniform meshes*, as we shall see in Section 8.3.5.

In Chapter 13 we discuss the Fokker–Planck and Kolmogorov backward equations. The former equation is the dual of the latter equation.

8.3.5 Numerical Approximation of PDEs in Adjoint Form

Elliptic equations in adjoint form occur in many application areas such as steady heat transfer and steady two-dimensional flow in non-homogeneous anisotropic porous media (Peaceman (1977), Bear (1979), Huyakorn and Pinder (1983)). To this end we discuss the self-adjoint semi-linear equation:

$$\frac{\partial}{\partial x} \left(\sigma_1(x, y) \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\sigma_2(x, y) \frac{\partial u}{\partial y} \right) + f(x, y, u) = 0, \quad 0 < x < L, \quad 0 < y < M \quad (8.16)$$

with Neumann boundary conditions:

$$\begin{aligned} \frac{\partial u}{\partial x} &= 0, \quad x = 0 \text{ and } x = L \\ \frac{\partial u}{\partial y} &= 0, \quad y = 0 \text{ and } y = M. \end{aligned} \quad (8.17)$$

This problem occurs in many physical applications (see Peaceman (1977)). We define meshpoints $\{x_i\}$ and $\{y_j\}$, $i = 0, \dots, I$, $j = 0, \dots, J$ in the x and y directions respectively, and to this end we adopt the following notation:

$$\begin{aligned} x_{i \pm \frac{1}{2}} &= \frac{1}{2}(x_i + x_{i \pm 1}) \\ y_{j \pm \frac{1}{2}} &= \frac{1}{2}(y_j + y_{j \pm 1}) \\ \Delta x_i &\equiv x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}} \\ \Delta y_j &\equiv y_{j+\frac{1}{2}} - y_{j-\frac{1}{2}}. \end{aligned} \quad (8.18)$$

We approximate the separate terms in Equation (8.16) as follows:

$$\frac{\partial}{\partial x} \left[\sigma_1(x_i, y_j) \frac{\partial u}{\partial x} \right]_{i,j} \approx \frac{\sigma_1 \left(x_{i+\frac{1}{2}}, y_j \right) \frac{u_{i+1,j} - u_{i,j}}{x_{i+1} - x_i} - \sigma_1 \left(x_{i-\frac{1}{2}}, y_j \right) \frac{u_{i,j} - u_{i-1,j}}{x_i - x_{i-1}}}{\Delta x_i} \quad (8.19)$$

in the x direction, and:

$$\frac{\partial}{\partial y} \left[\sigma_2(x_i, y_j) \frac{\partial u}{\partial y} \right]_{i,j} \approx \frac{\sigma_2 \left(x_i, y_{j+\frac{1}{2}} \right) \frac{u_{i,j+1} - u_{i,j}}{y_{j+1} - y_j} - \sigma_2 \left(x_i, y_{j-\frac{1}{2}} \right) \frac{u_{i,j} - u_{i,j-1}}{y_j - y_{j-1}}}{\Delta y_j} \quad (8.20)$$

in the y direction.

This model problem can be used in more complex finance applications. A study of this equation in the wider area of Black–Scholes PDEs and their approximation is an interesting research topic to pursue.

8.3.6 Elliptic Equations with Non-Negative Characteristic Form

We now consider a special case of *degenerate elliptic equations* for which the condition (8.2) is replaced by the following condition:

$$\sum_{i,j=1}^n a_{ij}(x) \xi_i \xi_j \geq 0, \quad \xi \in \mathbb{R}^n \quad (8.21)$$

which is defined in some domain (Fichera (1956), Oleinik and Radkevic (1973)). In particular, the *characteristic form* (8.21) can be *degenerate* (zero) at certain points of the domain (in the main, on its boundary). This property has major consequences for elliptic boundary value problems and applications to finance.

Closely related to the current topic is the class of *semi-elliptic* PDEs (Øksendal (1998)) in which the eigenvalues of the symmetric diffusion matrix are non-negative, whereas they are positive for the diffusion matrix of elliptic equations.

8.4 CLASSIFICATION OF SECOND-ORDER EQUATIONS

We now reduce the scope by examining linear second-order elliptic equations in two dimensions as described by Equation (8.5). The goal in this section is to transform this PDE to a form that simplifies its principal part as defined in Equation (8.6). In particular, we wish to have a technique to reduce (8.5) to a form in which the mixed derivative term has been removed in some (magical) way. We say that the PDE is reduced to *canonical form* in which there is no mixed derivative term.

In order to understand this section, you need to be fluent in using the chain rule for differentiation of functions of two variables (as discussed in Chapter 1) and how to apply this rule to generic two-factor convection-diffusion-reaction equations such as

(8.5) and its special cases in finance. Practice, practice, practice, ideally with pencil and paper until you get the correct answer.

We sometimes use the following shorthand for first and second order derivatives:

$$\begin{aligned} u_x &= \frac{\partial u}{\partial x}, \quad u_y = \frac{\partial u}{\partial y} \\ u_{xx} &= \frac{\partial^2 u}{\partial x^2}, \quad u_{yy} = \frac{\partial^2 u}{\partial y^2} \\ u_{xy} &= \frac{\partial^2 u}{\partial x \partial y}, \quad u_{yx} = \frac{\partial^2 u}{\partial y \partial x}. \end{aligned}$$

8.4.1 Characteristics

We usually consider Equation (8.5) in some region of two-dimensional space. There are two main cases to be considered (Courant and Hilbert (1968)):

1. How can a coordinate transformation be used to simplify the principal part of (8.5)?
2. How is knowledge of characteristic curves together with (8.5) sufficient to uniquely determine a solution?

To answer the first question we assume a locally invertible change of independent variables:

$$\xi = \phi(x, y), \quad \eta = \psi(x, y) \quad (\phi_x \psi_y - \phi_y \psi_x \neq 0) \quad (8.22)$$

and we transform the principal part of (8.5) to a PDE of the form:

$$A u_{\xi\xi} + 2B u_{\xi\eta} + C u_{\eta\eta} + R$$

where:

$$\begin{aligned} A &= a\phi_x^2 + 2b\phi_x\phi_y + c\phi_y^2 \\ B &= a\phi_x\psi_x + b(\phi_x\psi_y + \phi_y\psi_x) + c\phi_y\psi_y \\ C &= a\psi_x^2 + 2b\psi_x\psi_y + c\psi_y^2 \\ R &\equiv (a\phi_{xx} + 2b\phi_{xy} + c\phi_{yy})u_\xi + (a\psi_{xx} + 2b\psi_{xy} + c\psi_{yy})u_\eta. \end{aligned} \quad (8.23)$$

Note that we used the chain rule to arrive at (8.23) (we advise you to check this fact), for example:

$$\begin{aligned} u_x &= u_\xi \phi_x + u_\eta \psi_x \\ u_{xx} &= u_\xi \phi_{xx} + (u_\xi)_x \phi_x + u_\eta \psi_{xx} + (u_\eta)_x \psi_x. \end{aligned}$$

and similarly for u_y, u_{yy} and u_{xy} .

Some algebra shows that:

$$B^2 - AC = (b^2 - ac)(\phi_x \psi_y - \phi_y \psi_x)^2 \quad (8.24)$$

which shows that (8.5) is invariant under an invertible change of independent variables. Then ϕ and ψ are both solutions of the *characteristic equation* of (8.5), namely:

$$az_z^2 + 2bz_x z_y + cz_y^2 = 0. \quad (8.25)$$

We also say that the level curves $z(x, y) = \text{const}$ of the solution of (8.25) are called *characteristic curves* of (8.5). It can then be shown that $z(x, y) = \text{const}$ is a solution of (8.25) if and only if $z(x, y) = \text{const}$ is a characteristic of (8.5):

$$\begin{aligned} a\left(\frac{dy}{dx}\right)^2 - 2b\left(\frac{dy}{dx}\right) + c &= 0 \\ \frac{dy}{dx} &= \frac{b \pm \sqrt{b^2 - ac}}{a}. \end{aligned} \quad (8.26)$$

We say that (8.5) is:

- Elliptic if $b^2 - ac < 0$
- Hyperbolic if $b^2 - ac > 0$
- Parabolic if $b^2 - ac = 0$.

In general, elliptic PDEs are the most commonly encountered equations in this book.

8.4.2 Model Example

Transforming a second-order PDE in two dimensions to one in canonical form (mixed derivative term absent) is not difficult, but it is probably a good idea to execute the steps that are needed to achieve this end by taking a concrete example (practice makes perfect). We take the elliptic PDE:

$$u_{xx} + 2u_{xy} + 17u_{yy} = 0 \quad (a = 1, b = 1, c = 17).$$

We now describe the steps to convert this PDE to canonical form.

1. The characteristic Equation (8.26) in this case becomes:

$$\frac{dy}{dx} = \frac{b \pm i\sqrt{b^2 - ac}}{a} = 1 \pm 4i, \quad (i = \sqrt{-1}).$$

2. We take the positive solution in this case, and the characteristic curve is then computed by integrating this ODE in step 1):

$$z = (x - y) + 4ix = \text{const.}$$

3. Set $\xi = \phi(x, y) = x - y$, $\eta = \psi(x, y) = 4x$.

4. Then as a check:

$$\begin{aligned}\phi_x &= 1, \phi_y = -1, \phi_{xx} = \phi_{yy} = \phi_{xy} = 0 \\ \psi_x &= 4, \psi_y = 0, \psi_{xx} = \psi_{yy} = \psi_{xy} = 0.\end{aligned}$$

5. We now compute the coefficients of the PDE in transformed coordinates using the formulae in (8.23) to give (we recommend that you check):

$$A = 16, B = 0, C = 16, R = 0.$$

6. Finally, we have the canonical form of the PDE:

$$u_{xx} + 2u_{xy} + 17u_{yy} = 16(u_{\xi\xi} + u_{\eta\eta}) = 0.$$

Thus, u satisfies Laplace's equation in (ξ, η) space.

This approach can be applied to PDEs with non-constant coefficients, as we shall see in Section 8.5. More generally, this technique can be used to transform two-factor Black–Scholes-style PDEs to canonical form, easing the way for a solution using the finite difference method and avoiding approximating the mixed derivative term. We also need to transform the convection part of the PDE as well.

8.4.3 Test your Knowledge

We conclude this section by showing the linear *Euler–Tricomi* PDE that occurs in the study of transonic flow:

$$u_{yy} + xu_{xx} = 0. \quad (8.27)$$

This PDE is hyperbolic for $x < 0$, elliptic for $x > 0$, and parabolic for $x = 0$. In the first case, we get:

$$\begin{aligned}\xi &= \varphi(x, y) = \frac{3}{2}y + (\sqrt{-x})^3 \\ \eta &= \psi(x, y) = \frac{3}{2}y - (\sqrt{-x})^3\end{aligned}$$

while in the elliptic case we get:

$$\xi = \frac{3}{2}y, \quad \eta = i\sqrt{x^2}.$$

We see that the characteristics are real in the hyperbolic case and complex in the elliptic case.

We leave it as an exercise to find the transformed PDEs in canonical form; of course, the PDE does not have a mixed term.

8.5 EXAMPLES OF TWO-FACTOR MODELS FROM COMPUTATIONAL FINANCE

In later chapters of this book we discuss two-factor option problems and approximation by the finite difference method. The underlying stochastic processes tend to be correlated in general, and this leads to the emergence of a mixed derivative term that must somehow be approximated by divided differences. In practice, the methods in Craig and Sneyd (1988) and Yanenko (1971) have worked well, but their use leads to non-monotone schemes. To complicate matters, there are several ways to approximate the mixed derivative term, and guidance on which choice is mathematically or numerically optimal is a source of ongoing debate.

8.5.1 Multi-Asset Options

Options on two assets comprise a number of specific cases such as correlation options, exchange options, spread options and options on the minimum or maximum of two risky assets, to name a few (Haug (2007)). Analytical solutions and approximations are available in many cases, for example by using the formula for the bivariate normal distribution (BVN) (see Duffy (2018)). We use these formulae when testing the accuracy of our finite difference schemes (as ‘second opinion’ as it were). But in general we use numerical methods because analytical solutions do not exist or would take weeks or months to come up with an answer.

Having decided on using the finite difference method, we carry out a number of tasks, one of which is how to deal with mixed derivative terms. The first approach is to transform the PDE to canonical form (as we do in this chapter) and then approximate the transformed PDE. The second approach is to keep the mixed derivative term in the PDE and then to approximate the PDE numerically using the Craig–Sneyd or Yanenko methods for the mixed derivative term. We discuss how to do this in Chapter 23.

As an example, consider the PDE for a prototypical two-factor model:

$$\begin{aligned} & \frac{1}{2}\sigma_1^2 S_1^2 \frac{\partial^2 u}{\partial S_1^2} + \frac{1}{2}\sigma_2^2 S_2^2 \frac{\partial^2 u}{\partial S_2^2} + \rho\sigma_1\sigma_2 S_1 S_2 \frac{\partial^2 u}{\partial S_1 \partial S_2} \\ & + (r - q_1)S_1 \frac{\partial u}{\partial S_1} + (r - q_2)S_2 \frac{\partial u}{\partial S_2} = ru - \frac{\partial u}{\partial t}. \end{aligned} \tag{8.28}$$

In this case we have:

$$a = \frac{1}{2}\sigma_1^2 S_1^2, \quad b = \frac{\rho}{2}\sigma_1\sigma_2 S_1 S_2, \quad c = \frac{1}{2}\sigma_2^2 S_2^2 \quad (x = S_1, y = S_2).$$

The characteristic equation for (8.28) in this case becomes:

$$\frac{dy}{dx} = \alpha_1 \frac{y}{x} + i\alpha_2 \frac{y}{x}; \quad \alpha_1 = \frac{\rho\sigma_2}{\sigma_1}, \quad \alpha_2 = \frac{\sqrt{1 - \rho^2}\sigma_2}{\sigma_1}.$$

We take the positive solution in this case and the characteristic curve is:

$$\begin{aligned} z &= \alpha_1 \log x - \log y - i\alpha_2 \log x \\ \varphi &= \xi = \alpha_1 \log x - \log y; \quad \psi = \eta = \alpha_2 \log x \\ x &= e^{\eta/\alpha_2}, \quad y = e^{\left(\frac{\alpha_1}{\alpha_2}\eta - \xi\right)}. \end{aligned}$$

Then some simple calculus shows that:

$$\begin{aligned} A &= \frac{1}{2}\sigma_2^2(1 - \rho^2) \\ C &= \frac{1}{2}\sigma_2^2(1 - \rho^2) \\ B &= 0 \\ R &= \left(-\frac{1}{2}\sigma_1^2\alpha_1 + \frac{1}{2}\sigma_2^2\right)u_\xi - \frac{1}{2}\sigma_1^2\alpha_2u_\eta. \end{aligned}$$

Finally, the transformed convection terms are:

$$\begin{aligned} (r - q_1)x\frac{\partial u}{\partial x} &= (r - q_1)\left[\alpha_1\frac{\partial u}{\partial \xi} + \alpha_2\frac{\partial u}{\partial \eta}\right] \\ (r - q_2)y\frac{\partial u}{\partial y} &= -(r - q_2)\frac{\partial u}{\partial \xi}. \end{aligned}$$

We end up with a PDE in canonical form, that is one in which the mixed derivative term is absent.

Reminder: we have to transform both the diffusion and convection terms in general.

8.5.2 Stochastic Dividend PDE

In the Black–Scholes model, it is assumed that the dividend rate is constant. Ignoring this term can have significant impact on pricing errors (Broadie, Detemple, Ghysels and Torres (2000), Wilmott (2006)). To this end, we model dividends as a stochastic process:

$$dD = p(s, D, t)dt + q(S, D, t)dW. \quad (8.29)$$

For example, dividends can be modelled as a mean-reverting process:

$$dD = (bS + \beta D)dt + \sqrt{D\left(S - \frac{D}{a}\right)}dW \quad (b \in \mathbb{R}, a > 0, \beta \in \mathbb{R}, \beta < 0, \sigma > 0). \quad (8.30)$$

Then an option that jointly prices derivatives on the stock and dividends is given by the PDE:

$$\frac{\partial V}{\partial t} = \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \rho\sigma q S \frac{\partial^2 V}{\partial S \partial D} + \frac{1}{2}q^2 \frac{\partial^2 V}{\partial D^2} + (R - D)S \frac{\partial V}{\partial S} + (p - \lambda q) \frac{\partial V}{\partial D} - rV. \quad (8.31)$$

Following the steps in Section 8.4.2, we can reduce PDE (8.31) to canonical form. The characteristic equation is given by:

$$\frac{dy}{dx} = \frac{\alpha}{x} + \frac{i\beta}{x}, \quad \alpha = \frac{\rho q}{\sigma}, \beta = \frac{\sqrt{1 - \rho^2}q}{\sigma} \quad (x \equiv S, y \equiv D)$$

and the characteristic curves are given by:

$$z = \alpha \log x - y + i\beta \log x \quad (i = \sqrt{-1}).$$

We leave it as an exercise to the reader to give the full specification of the canonical PDE.

In general, dividend derivatives have no closed solution. In that case, we can use the Monte Carlo method, and of course we can apply the finite difference method directly to PDE (8.31). It is also possible to transform other PDEs to canonical form, such as two-factor interest rate, stochastic volatility and convertible bond problems.

8.6 SUMMARY AND CONCLUSIONS

In this chapter we introduced elliptic operators and elliptic equations. These equations are the backbone of the time-dependent partial differential equations in finance. In particular, we defined what a two-dimensional elliptic equation is. We also introduced a mathematical technique (using a change of independent variables) to convert an elliptic PDE to another elliptic PDE with no mixed derivative term. We gave both generic examples and examples from finance to show how this technique works.

A final remark on the relative merits of producing analytical solutions versus defining a finite difference framework that subsumes and supports a range of PDEs sharing similar structure and functionality: in the former case each new specific option type will need its own specific formula (at possible great expense in human resource terms and in terms of the origination and testing of analytical option pricing formulae). Reusability of the algorithm and of code is not guaranteed, and the algorithm may place restrictions on the parameters if it is to converge. Furthermore, the accuracy is fixed, and it is usually not possible to customise the algorithm to suit difference performance and accuracy needs. For example, it is difficult to customise the algorithm that produces six-digit accuracy on the one hand or to produce results to two-digit accuracy on the other hand. In many cases it tends to be ‘all or nothing’. In the latter case (using finite differences), we only need to set up a flexible software framework once, and then it can be instantiated for specific types, for example spread options. A priori accuracy is determined by the mesh sizes and the scheme that we use. This usually involves nothing more than using a different payoff function, with no modification to the rest of the code being needed. This is the stage in which we can apply software design patterns to achieve this desired level of flexibility. Finally, the software framework can be used for problems for which an analytical solution is not known.

Transforming Partial Differential Equations to a Bounded Domain

The first rule of discovery is to have brains and good luck. The second rule of discovery is to sit tight and wait till you get a bright idea.

George Pólya

9.1 INTRODUCTION AND OBJECTIVES

In Chapter 8 we introduced elliptic equations. In general, these equations are defined in a bounded or unbounded domain in two-dimensional space and the goal of the current chapter is to study PDEs in such domains. From a computational point of view we transform a PDE on an unbounded domain to one on a bounded domain, either by a change of variables or by truncating the unbounded domain. We propose a number of ways of achieving this goal and we give some examples of one-factor and two-factor PDEs in finance.

To our knowledge, the results in this chapter are relatively new for finance applications and we see them as a major improvement on the heuristic *domain truncation* technique.

9.2 THE DOMAIN IN WHICH A PDE IS DEFINED: PREAMBLE

In Chapter 8 we introduced elliptic PDEs in two independent variables that we usually denote by x and y . We did not discuss what the range of values of these variables are, in other words the domain in which the PDE is defined. There are various choices, for example:

$$0 < x, y < \infty$$

$$-\infty < x, y < \infty$$

$$A < x < B, C < y < D \text{ (bounded domain).}$$

Here we speak of problems in a quarter-plane, in the plane and in a bounded domain. In general, we wish to formulate a boundary value problem for elliptic and parabolic equations with auxiliary conditions to ensure that the problem has a unique solution. Most PDEs in finance are defined in a quarter-plane; for example, the PDE for a two-asset option problem has non-negative underlying stock variables. We speak of the *near-field boundary* where the value of underlying is zero and the *far-field boundary* corresponding to infinity. Since we cannot model infinity in a computer, we must transform a PDE that is defined in an infinite domain to a PDE in a new bounded domain. In most cases (and without loss of generality in the main), the unit square or unit interval is the bounded domain of choice. Thus, we are mainly interested in interval transformations such as:

$$(0, \infty) \rightarrow (0, 1)$$

$$(-\infty, \infty) \rightarrow (-1, 1).$$

The generalisation to two-dimensional regions is easy. And we shall see examples from finance in later chapters.

Some of the issues to be addressed when transforming PDEs in one coordinate system to another new coordinate system are:

- A1: Choice of the transformer function and its inverse. Which transformer is ‘best’ and most appropriate for a given problem?
- A2: The form of the PDE in the new coordinate system.
- A3: Possible singularities in the transformer function and its inverse.
- A4: Defining *hotspot points* in the transformed domain where we wish to find an approximate solution. To this end, we define a *scale factor* that allows us to define a 1:1 relationship between given points in the original and transformed domains.

We examine these important topics in this and succeeding chapters as well as using them in finite difference schemes for one-factor and two-factor option pricing problems.

9.2.1 Background and Specific Mappings

The idea of coordinate and domain transformations is not new, and it has been applied in many areas of engineering and fluid dynamics, as well as in applied and numerical mathematics (Roache (1998), Boyd (2000)). The technique has wide applicability in areas such as:

- *Mapping infinite and semi-infinite intervals* to $[-1, 1]$ so that Chebyshev polynomials can be applied.
- *Regions of very rapid change* in fluid flow applications, for example near-singularities and internal boundary layers. A suitable transformation can result in improved efficiency.
- *Integration of functions* with end-point singularities (Mori and Sugihara (2001)): we consider the integral $I = \int_a^b f(x)dx$. The interval (a, b) may be infinite, semi-infinite

or finite. The function f must be analytic on (a, b) but it may have a singularity at $x = a$ or $x = b$, or both. We now apply the variable transformation:

$$\phi : \mathbb{R} \rightarrow (a, b)$$

$x = \phi(t)$ such that $a = \phi(-\infty)$, $b = \phi(\infty)$, where ϕ is analytic on $(-\infty, \infty)$.

The integral then becomes $I = \int_{-\infty}^{\infty} f(\phi(t))\phi'(t)dt$. Now, the *double-exponential rule* for the interval $(-1, 1)$ and for a specific transformation becomes (when we approximate the integral by the trapezoidal rule is given by):

$$I = \int_{-1}^1 f(x)dx$$

$$\text{the transformation } x = \phi(t) = \tanh\left(\frac{\pi}{2} \sinh t\right)$$

$$I_h^{(N)} = h \sum_{k=-N_-}^{N_+} f\left(\left(\frac{\pi}{2} \sinh kh\right)\right) \frac{\pi/2 \cosh kh}{\cosh^2(\pi/2 \sinh kh)}.$$

- *Constrained optimisation* problems in which a transformation reduces a constrained optimisation problem to a form in which no constraints explicitly appear (Box (1966)). We take a simple example in three variables:

$$\min f(x_1, x_2, x_3) \text{ such that } 0 \leq x_1 \leq x_2 x_3.$$

Then we can define the new variables y_1, y_2 and y_3 by the transformation:

$$\begin{aligned} x_1 &= y_1^2 \\ x_2 &= y_1^2 + y_2^2 \\ x_3 &= y_1^2 + y_2^2 + y_3^3. \end{aligned}$$

We then solve an unconstrained problem in y -space. Transformations that arise in fluid dynamics are described in Roache (1998):

$$\begin{aligned} a) \quad y &= \frac{ax^b}{ax^b + 1}, [0, \infty] \rightarrow [0, 1] \\ b) \quad y &= 1 - e^{-ax}, [0, \infty] \rightarrow [0, 1] \\ c) \quad y &= \tanh(ax), [-\infty, \infty] \rightarrow [-1, 1] \\ d) \quad y &= \frac{1}{2}[1 + \tanh(ax)], [-\infty, \infty] \rightarrow [-1, 1] \end{aligned} \tag{9.1}$$

where $a_0 > 0, b > 0$ are positive constants.

In fact, we use cases *a*) (with $b = 1$) and *c*) for equity and interest-rate problems.

9.2.2 Initial Examples

We introduce a transformation and its inverse as follows:

$$\begin{aligned} y &= \frac{x}{x + \alpha} \quad ((0, \infty) \rightarrow (0, 1)) \\ x &= \frac{\alpha y}{1 - y}. \end{aligned} \tag{9.2}$$

As an example, we take a generalisation of the Black–Scholes equation called the *constant elasticity of variance* (CEV) model:

$$\frac{\partial V}{\partial t} = \frac{1}{2} \sigma^2 S^{2\beta} \frac{\partial^2 v}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV. \tag{9.3}$$

which is a special case of a more generic convection-diffusion-reaction PDE:

$$\frac{\partial u}{\partial t} = a(x, t) \frac{\partial^2 u}{\partial x^2} + b(x, t) \frac{\partial u}{\partial x} + c(x, t)u + f(x, t), \quad 0 < x < \infty, \quad 0 < t < T.$$

We discuss transforming the PDE (9.3) to a PDE on a bounded domain. We describe the steps in bullet form and you should check the steps:

1. Introduce a new independent variable:

$$y = \frac{x}{x + \alpha}.$$

2. The transformed PDE becomes:

$$\frac{\partial u}{\partial t} = A(y, t) \frac{\partial^2 u}{\partial y^2} + B(y, t) \frac{\partial u}{\partial y} + C(y, t)u + F(y, t) \quad 0 < y < 1$$

where:

$$A(y, t) = \frac{1}{2} \sigma^2 \alpha^{2\beta-2} y^{2\beta} (1-y)^{4-2\beta}$$

$$B(y, t) = -\sigma^2 \alpha^{2\beta-2} y^{2\beta} (1-y)^{3-2\beta} + ry(1-y), \quad y = \frac{S}{S + \alpha}$$

$$C(y, t) = -r, \quad F(y, t) = 0.$$

A special case is $\beta = 1$, which is the standard Black–Scholes PDE. This is a useful first check that we have computed the coefficients correctly.

Finally, we take two examples:

$$y = \tanh \alpha x \text{ (Case 1)} \text{ and } y = \frac{1}{1 + \alpha x} \text{ (Case 2).}$$

Then the transformations become:

Case 1:

$$x = \frac{1}{2\alpha} \log \left(\frac{1+y}{1-y} \right) = \log \left(\frac{1+y}{1-y} \right)^{1/2\alpha}$$

$$\begin{aligned}\frac{dy}{dx} &= \alpha(1 - y^2) \\ \frac{\partial u}{\partial x} &= \alpha(1 - y^2)\frac{\partial u}{\partial y} \\ \frac{\partial^2 u}{\partial x^2} &= \alpha^2(1 - y^2)\frac{\partial}{\partial y}\left((1 - y^2)\frac{\partial u}{\partial y}\right).\end{aligned}$$

Case 2:

$$\begin{aligned}x &= \frac{1-y}{\alpha y}, \frac{dy}{dx} = -\alpha y^2 \\ \frac{\partial u}{\partial x} &= -\alpha y^2 \frac{\partial u}{\partial y} \\ \frac{\partial^2 u}{\partial x^2} &= \alpha^2 y^2 \frac{\partial}{\partial y}\left(y^2 \frac{\partial u}{\partial y}\right).\end{aligned}$$

9.3 OTHER EXAMPLES

We recall from Chapter 8 that it is possible to transform a PDE to a PDE in canonical form $(x, y) \leftrightarrow (\xi, \eta)$.

Typical examples are:

$$\begin{aligned}\xi &= x - y; \quad \eta = 4x \\ \xi &= \alpha \log x - \log y; \quad \eta = \beta \log x.\end{aligned}\tag{9.4}$$

and the inverse mappings are:

$$\begin{aligned}x &= \eta/4; \quad y = \eta/4 - \xi \\ x &= e^{\eta/\beta} \quad y = e^{(\alpha\eta/\beta - \xi)}.\end{aligned}\tag{9.5}$$

What do we do with a PDE in canonical form? First, it is difficult to discretise it because the coordinates are rotated or stretched, and then the original boundary conditions will be difficult to enforce. We can alleviate this problem by domain transformation to produce a PDE on the box $(0, 1) \times (0, 1)$ with coordinates z and w . We can then solve this PDE using splitting and ADE schemes that we discuss in Chapters 18, 19, 22 and 23. The only small attention point is to realise that the finite difference schemes compute its results in (z, w) space while the financial problem is in (x, y) space. Thus, we need a pair of two-way mappings:

$$(x, y) \leftrightarrow (\xi, \eta) \leftrightarrow (z, w)$$

whose traceability we realise using *hotspots*. If we get to this stage (that is, a PDE in canonical form), then Marchuk's Two-Cycle method is recommended.

9.4 HOTSPOTS

We see the presence of a so-called *scale factor* α in equation (9.2). The reason for introducing it is to associate points in the original and transformed PDE spaces. We take an example in which the variable x in Equation (9.2) plays the role of stock S . Rearranging we see that:

$$\alpha = \frac{S(1-y)}{y}.$$

In order to compute the scale factor, we first give the value of (say $S = 100$) for which we wish to compute the option price as well a value of y that you wish it to correspond to, for example $y = 1/2$. Then in this case we get $\alpha = 100$ by using the above formula.

For two-factor problems we will have two transformations and two scale factors. For example, for an Asian-style option (Duffy (2018)) we have the following transformations and their inverses:

$$\begin{aligned} x &= \frac{S}{S+a}, & y &= \frac{A}{A+b} \\ S &= \frac{ax}{1-x}, & A &= \frac{by}{1-y}. \end{aligned}$$

as we shall see in Chapter 24. In this formula, S is the stock and A is some kind of average of C

9.5 WHAT HAPPENED TO DOMAIN TRUNCATION?

Most option pricing problems are defined for non-negative values of the underlying stock whose values (at least theoretically) have no upper bound. From a computational viewpoint we need to truncate a semi-infinite interval to a bounded interval in which the original PDE in question will be defined:

$$\begin{aligned} [S_{\min}, S_{\max}], & \quad S_{\min} < S_{\max} \\ S_{\min} &= \text{near-field boundary (usually 0)} \\ S_{\max} &= \text{far-field boundary.} \end{aligned}$$

This approach is different from the domain transformation approach because on the one hand the original PDE is now defined on a truncated interval and it does not need to be transformed to another PDE. However, this approach is not robust in general, and it has weak mathematical foundations in general. An exception is Kangro and Nicolaides (2000), in which the authors produced the following estimate for the far-field value as some multiple of the strike price K for a specific problem:

$$S_{\max} = \max(\alpha K, K \exp(\sqrt{2\sigma^2 T \log 100})) \text{ for } \alpha \geq 2.$$

The volatility σ plays an important role and we see that the value of the far-field boundary is influenced by it. Some of the (negative) consequences are:

- It is not clear how to find precise estimates for the far-field boundary; small values will probably cause spurious oscillations and reflections, while large values will demand that we take a large (and unnecessary) number of mesh points.
- How to prescribe boundary conditions is mostly a question of trial and error, using tricks and assumptions in combination with numerical experimentation. For example, the ubiquitous *linearity boundary condition*:

$$\frac{\partial^2 V}{\partial S^2}(S_{\max}, t) = 0$$

is popular but it requires at least a three-point finite difference operator at the boundary, destroying the tridiagonal structure of the space discretisation matrix (Tavella and Randall (2000)).

- In some cases, it is not allowed to prescribe boundary conditions for certain mathematical reasons that we discuss in Chapter 11 (Fichera theory). In the case of the Cox–Intgersoll–Ross (CIR) model, for example, no explicit boundary condition is needed/allowed if the *Feller condition* is satisfied (Tavella and Randall (2000)). We discuss a structural solution to this problem in Chapter 25.

Despite its perceived shortcomings, domain truncation does offer a number of advantages:

- It is easy to implement when compared to domain transformation. If it works well, then there may be no need for more complex solutions.
- In some two-factor models we could use domain transformation for one state variable and domain truncation for the other state variable.

9.6 ANOTHER WAY TO REMOVE MIXED DERIVATIVE TERMS

In Chapter 8 we showed how to remove the mixed derivative term from an elliptic PDE by transforming it to canonical form. In this section we propose a different approach to achieve the same goal. We wish to end up with a constant-coefficient convection-diffusion-reaction equation in the plane containing no mixed derivative terms. The first step is to use the well-known *log transformation* $x = \log S$ to produce a constant-coefficient PDE. The mixed derivative term is then eliminated by a *rotation of the axis*.

The Black–Scholes PDE is the most important PDE in computational finance. In this case it describes the price of an option as a function of several independent variables such as stock price, volatility and interest rates. In this case we examine the two-dimensional case for $0 \leq S_1, S_2 < \infty$:

$$-\frac{\partial V}{\partial t} + L_1 V + L_2 V + \rho \sigma_1 \sigma_2 S_1 S_2 \frac{\partial^2 V}{\partial S_1 \partial S_2} - rV = 0$$

where: (9.6)

$$L_j V \equiv \frac{1}{2} \sigma_j^2 S_j^2 \frac{\partial^2 V}{\partial S_j^2} + (r - D_j) S_j \frac{\partial V}{\partial S_j}, \quad j = 1, 2.$$

This PDE is thus defined on a semi-infinite (space) region. The objective of this *particular* discussion is to transform the PDE to a form that does not contain a mixed derivative term. A major advantage of doing this is that we do not have to approximate the *mixed derivative* when approximating the PDE by the finite difference method. We use the chain rule for differentiation in the sequel.

We take the following steps:

- a. Define the transformation $x_1 = \log(S_1)$, $x_2 = \log(S_2)$. Using the *chain rule for differentiation*, the transformed PDE in the *infinite domain* $-\infty < x_1, x_2 < \infty$ becomes:

$$-\frac{\partial V}{\partial t} + M_1 V + M_2 V + \rho \frac{\partial^2 V}{\partial X_1 \partial X_2} - rV = 0$$

where: (9.7)

$$M_j V \equiv \frac{1}{2} \frac{\partial^2 V}{\partial X_j^2} + \frac{\mu_j}{\sigma_j} \frac{\partial V}{\partial X_j} \quad j = 1, 2.$$

- b. Now define the transformation $X_1 = X = x_1/\sigma_1$, $X_2 = Y = x_2/\sigma_2$. We can show that the transformed PDE in the infinite domain $-\infty < X_1, X_2 < \infty$ becomes:

$$-\frac{\partial V}{\partial t} + M_1 V + M_2 V + \rho \frac{\partial^2 V}{\partial X_1 \partial X_2} - rV = 0$$

where: (9.8)

$$M_j V \equiv \frac{1}{2} \frac{\partial^2 V}{\partial X_j^2} + \frac{\mu_j}{\sigma_j} \frac{\partial V}{\partial X_j} \quad j = 1, 2.$$

- c. Now define the transformations:

$$\begin{aligned} x &= \frac{X_1 + X_2}{2} \equiv \frac{X + Y}{2} \\ y &= \frac{X_1 - X_2}{2} \equiv \frac{X - Y}{2}. \end{aligned}$$

The transformed PDE in the infinite region $-\infty < X_1, X_2 < \infty$ becomes a *convection-diffusion-reaction* equation with no mixed derivative:

$$-\frac{\partial V}{\partial t} + \beta_1 \frac{\partial^2 V}{\partial x^2} + \beta_2 \frac{\partial^2 V}{\partial y^2} + \alpha_1 \frac{\partial V}{\partial x} + \alpha_2 \frac{\partial V}{\partial y} - rV = 0 \quad (9.9)$$

where:

$$\beta_1 = \frac{1}{4} (1 + \rho)$$

$$\beta_2 = \frac{1}{4} (1 - \rho)$$

$$\alpha_1 = \frac{1}{2} \left(\frac{\mu_1}{\sigma_1} + \frac{\mu_2}{\sigma_2} \right)$$

$$\alpha_2 = \frac{1}{2} \left(\frac{\mu_1}{\sigma_1} - \frac{\mu_2}{\sigma_2} \right).$$

The domain of this PDE is the plane, that is $-\infty < x, y < \infty$.

What's next? We can take equation (9.9) as starting point for further analysis. Some use cases are:

- U1: Pose (9.9) as a PDE in adjoint form.
- U2: Transform (9.9) to a PDE on the unit square.
- U3: Applying Fichera theory (as will be discussed in Chapter 11).
- U4: Determining numerical boundary conditions.
- U5: Choice of finite difference scheme (Splitting, ADI, ADE, MOL).
- U6: Using exponential fitting for convection-dominated problems (Sheppard (2007)).
- U7: Proving well-posedness of (9.9) by showing that the solution of (9.9) is continuously dependent on its data.

We shall discuss all of these topics in later chapters. Use cases U1 and U7 could form the basis of a useful MSc student project.

9.7 SUMMARY AND CONCLUSIONS

In this chapter we discussed how to transform a PDE that is defined on an infinite domain to one on a bounded domain. The technique is mathematically rigorous, and it avoids many of the problems associated with the ad hoc (and popular) domain truncation approach in finance.

CHAPTER 10

Boundary Value Problems for Elliptic and Parabolic Partial Differential Equations

It ain't over till it's over.

Yogi Berra

10.1 INTRODUCTION AND OBJECTIVES

In this relatively short chapter we continue with our study of elliptic partial differential equations from Chapters 8 and 9. In those chapters we discussed what elliptic equations are, some examples from finance, and how to remove the mixed derivative term. We focused on the mechanics of elliptic partial differential equations, as it were. The treatment to date has been (deliberately) incomplete, and we still have to address a number of topics:

- A1:** Model elliptic PDEs such as the Laplace equation and some of its important properties that can be generalised to more complex PDEs.
- A2:** What kinds of *boundary conditions* are there for elliptic partial differential equations?
- A3:** *Well-posedness*, maximum-minimum principle.
- A4:** Separation of Variables techniques.
- A5:** Laplace's equation in various geometries.

We focus on uniformly and strongly elliptic equations in this chapter. Then it is relatively easy to specify boundary conditions. The case of non-negative characteristic forms will be discussed in Chapter 11.

10.2 NOTATION AND PREREQUISITES

We use the results, notation and techniques from Chapters 8 and 9 throughout the current chapter. We also use the chain rule for differentiation.

10.3 THE LAPLACE EQUATION

We begin with the Laplace equation, which is undoubtedly one of the most important partial differential equations of mathematical physics. It has many applications in fluid dynamics, heat flow, electromagnetics, and potential theory (Kellogg (1953)). Studying it is important because we can approach it from several angles, and the results can be generalised to more complex cases.

Laplace's equation in n dimensions is given by:

$$\Delta u \equiv \nabla^2 u = \sum_{j=1}^n \frac{\partial^2 u}{\partial x_j^2} = 0. \quad (10.1)$$

A function u that satisfies (10.1) is called *harmonic*. In the sequel, we assume that $n = 2$, that is we examine Laplace's equation in two dimensions. We say that a function u is *subharmonic* if $\Delta u \geq 0$ and *superharmonic* if $\Delta u \leq 0$.

Examples of harmonic functions are:

$u(x, y) = x^2 - y^2$ in any region Ω of the xy plane.

$u(x, y, z) = (x^2 + y^2 + z^2)^{-1/2}$ in any three-dimensional region that does not contain the origin.

10.3.1 Harmonic Functions and the Cauchy–Riemann Equations

There is a strong connection between harmonic functions and *complex function theory*. To this end, we give necessary and sufficient conditions for a complex function to be analytic in terms of its conjugate functions and that both conjugate functions satisfy Laplace's equation. We now describe what we mean by this statement.

Let Ω be a non-empty connected open subset of the complex plane. Then the function f is *analytic* (also known as *regular*, *holomorphic*) at the point $z_0 \in \Omega$ if the following limit exists:

$$\lim_{z \rightarrow z_0} \frac{f(z) - f(z_0)}{z - z_0} \equiv f'(z_0) \quad (10.2)$$

or alternatively

$$\lim_{\Delta z \rightarrow 0} \frac{f(z + \Delta z) - f(z)}{\Delta z} \equiv f'(z).$$

An *entire function* is one that is analytic in the whole complex plane.

Some examples of analytic and entire functions are:

- The functions e^z , $\sin z$, $\cos z$ are entire.
- The function $w(z) = \frac{1+z}{1-z}$, $\frac{dw}{dz} = \frac{1}{(1-z)^2}$ is analytic for $z \neq 1$ ($z = 1$ is a *singular point*).
- The complex conjugate $\bar{z} = x - iy$ ($z = x + iy$) is not analytic anywhere. (The limit in (10.2) depends on the manner in which $\Delta z \rightarrow 0$.)
- Polynomials of all orders of a complex argument are analytic.

The *Cauchy–Riemann* (CR) conditions are necessary for a function $w = f(z) = u(x, y) + iv(x, y)$ ($z = x + iy$) to be analytic:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}, \quad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}. \quad (10.3)$$

Furthermore, if these partial derivatives are continuous, then CR conditions are sufficient for f to be analytic (u and v are called *conjugate functions*).

We take two examples of entire functions, and we show the corresponding Cauchy–Riemann conditions:

1. $f(z) = z^2$

$$z^2 = (x + iy)^2 = x^2 - y^2 + 2ixy = u + iv$$

$$\frac{\partial u}{\partial x} = 2x, \quad \frac{\partial v}{\partial y} = 2x$$

$$\frac{\partial u}{\partial y} = -2y, \quad \frac{\partial v}{\partial x} = 2y.$$

2. $f(z) = e^z = e^{x+iy} = e^x(\cos y + i \sin y) = u + iv$

$$\frac{\partial u}{\partial x} = e^x \cos y, \quad \frac{\partial v}{\partial y} = e^x \cos y$$

$$\frac{\partial u}{\partial y} = -e^x \sin y, \quad \frac{\partial v}{\partial x} = e^x \sin y.$$

Continuing, both conjugate functions of an analytic function satisfy Laplace's equation:

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

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = 0$$

(v is called the *harmonic conjugate* of u).

As an example, we show that the conjugate functions of the exponential function satisfy Laplace's equation by executing the following steps:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = e^x \cos y - e^x \cos y = 0$$

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = e^x \sin y - e^x \sin y = 0.$$

Complex variables are used in option pricing applications as discussed in Carr and Madan (1998), Cont and Tankov (2000), Lewis (2000), and Lewis (2016). These methods use the Fourier transform and its inverse; in the one-dimensional case they are defined as follows:

$$Ff(y) = \int_{-\infty}^{\infty} e^{-ixy} f(x) dx, \quad (i = \sqrt{-1})$$

$$F^{-1}f(x) = \int_{-\infty}^{\infty} e^{ixy} f(y) dy.$$

where f is a real-valued function, $x, y \in \mathbb{R}$.

Further elaboration is outside the scope of this book. A good introduction to complex variables is Spiegel (1999). We give a relatively self-contained introduction to complex analysis in Chapter 16.

10.4 PROPERTIES OF THE LAPLACE EQUATION

In general, it is not possible to find a closed solution for elliptic boundary value problems. However, we may wish to test the accuracy of a finite difference scheme, and it is then useful to have some solutions to benchmark against. To this end, we give a crash course in the *Separation of Variables* technique for two-dimensional problems. This is a topic that is popular in undergraduate mathematics and engineering courses and textbooks. We examine Laplace's equation on a rectangular region with Dirichlet boundary conditions:

$$\begin{cases} \text{(a)} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad 0 < x < L, \quad 0 < y < M \\ \text{(b)} u(0, y) = u(L, y) = 0, \quad 0 < y < M \\ \text{(c)} u(x, M) = 0, \quad u(x, 0) = f(x), \quad 0 < x < L. \end{cases} \quad (10.4)$$

We then seek a solution $u(x, y)$ of the *boundary value problem* (10.4) by using the *ansatz* (assumption):

$$u(x, y) = XY, \quad X = X(x), \quad Y = Y(y).$$

Plugging this representation into Equation (10.4)(a), we get:

$$\frac{d^2 Y}{dy^2} / Y = - \frac{d^2 X}{dx^2} / X \quad (10.5)$$

and the pair of ordinary differential equations:

$$\begin{aligned} \frac{d^2X}{dx^2} + \lambda X = 0 & \quad (\text{a}) \\ \frac{d^2Y}{dy^2} - \lambda Y = 0 & \quad (\text{b}) \end{aligned} \tag{10.6}$$

where λ is a constant.

From Equations (10.4(b)) and (10.4(c)) we see that:

$$\begin{aligned} X(0) = X(L) = 0 & \quad (\text{a}) \\ Y(M) = 0 & \quad (\text{b}). \end{aligned} \tag{10.7}$$

We then get the representation (Kreider, Kuller, Ostberg and Perkins (1966)):

$$\begin{aligned} X_n(x) &= A_n \frac{\sin n\pi x}{L}, \quad n = 1, 2, \dots \\ \lambda = \lambda_n &= \left(\frac{n\pi x}{L}\right)^2 \end{aligned}$$

where X_n is the solution of the *Sturm–Liouville system* (10.6)(a) with boundary conditions (10.7)(a).

Using this fact in the ordinary differential equation for Y gives us:

$$\frac{d^2Y}{dy^2} - \frac{n^2\pi^2}{L^2} Y = 0.$$

that has the solution (containing two constants to be determined by $Y(M) = 0$):

$$\begin{aligned} Y_n(y) &= B_n \sinh \frac{n\pi y}{L} + C_n \cosh \frac{n\pi y}{L} \\ B_n &= -\cosh \frac{n\pi M}{L}, \quad C_n = \sinh \frac{n\pi}{L} M. \end{aligned}$$

Using the formula:

$$\sinh \alpha \cosh \beta - \cosh \alpha \sinh \beta = \sinh(\alpha - \beta)$$

gives us the general expression:

$$Y_n(y) = \sinh \frac{n\pi}{L} (M - y)$$

and hence:

$$u(x, y) = \sum_{n=1}^{\infty} u_n(x, y) = \sum_{n=1}^{\infty} A_n \sin \frac{n\pi x}{L} \sinh \frac{n\pi}{L} (M - y).$$

There is only one unknown term left, and we use the fact $u(x, 0) = f(x)$:

$$u(x, 0) = \sum_{n=1}^{\infty} A_n \sinh\left(\frac{n\pi M}{L}\right) \sinh \frac{n\pi x}{L} = f(x)$$

$$A_n = \frac{2}{L \sinh \frac{n\pi M}{L}} \int_0^L f(x) \sin\left(\frac{n\pi x}{L}\right) dx.$$

Finally, the exact solution of the boundary value problem (10.4) is given by:

$$u(x, y) = \frac{2}{L} \sum_{n=1}^{\infty} \frac{\int_0^L f(x) \sin(n\pi x/L) dx}{\sinh(n\pi M/L)} \sin\left(\frac{n\pi x}{L}\right) \sinh \frac{n\pi}{L}(M - y).$$

This solution is valid when f is sufficiently smooth. In particular, if the function f and its first-order derivatives are piecewise continuous in the interval $[0, L]$, then the formal solution is uniformly and absolutely convergent to the exact solution in $[0, L] \times [0, M]$.

The Separation of Variables technique that we have just discussed can be applied to more general boundary condition, for example Neumann, Robin, and the following non-homogeneous Dirichlet boundary conditions:

$$u(0, y) = f_1(y), \quad u(L, y) = f_2(y), \quad 0 < y < M$$

$$u(x, 0) = f_3(x), \quad u(x, M) = f_4(x), \quad 0 < x < L.$$

For more details, see Tolstov (1962).

Learning how to solve linear, constant-coefficient partial differential equations using the Separation of Variables method is a useful technique in itself, and the results can be used as data to compare against numerical solutions. We shall also see how to apply this technique to finding analytic solutions to time-dependent problems.

For more background on this method, see Kreider, Kuller, Ostberg and Perkins (1966) and Tolstov (1962). But in practice, the method has limited applicability. Nonetheless, it is a useful skill set to have.

10.4.1 Maximum-Minimum Principle for Laplace's Equation

A question we would like to answer is how the minimum and maximum values of the solution of an elliptic equation in a bounded domain are bounded by its values on the boundary of the domain. We first consider the case of the Laplace equation, and we will generalise the results to general elliptic and parabolic equations and their numerical approximation.

Theorem 10.1 Let Ω be a bounded domain with boundary Γ , and let the function u be harmonic in the closure $\overline{\Omega}$ of Ω . Let:

$$M = \max_{x \in \Gamma} u(x), \quad m = \min_{x \in \Gamma} u(x).$$

Then the *weak minimum-maximum principle* is:

$$m \leq u(x) \leq M \quad \forall x \in \overline{\Omega} = \Omega \cup \Gamma$$

and the *strong maximum-minimum principle* is:

$$\text{either } m < u(x) < M \quad \forall x \in \Omega \text{ or else } m = u(x) = M \quad \forall x \in \overline{\Omega}.$$

10.5 SOME ELLIPTIC BOUNDARY VALUE PROBLEMS

Elliptic PDEs usually have infinitely many solutions. To select a single solution, we must impose certain auxiliary conditions that characterise a system being modelled. In the case of elliptic equations, these are called *boundary conditions* that are specified at points of the boundary Γ of a spatial domain Ω in which a PDE is defined. To this end, let $\eta = \eta(x)$ be the unit *outward* normal vector to Γ that is defined by $\eta = (\eta_1, \eta_2)$, $|\eta| = \sqrt{\eta_1^2 + \eta_2^2} = 1$. We let g , α and β be functions prescribed on Γ . The three main types of boundary conditions are:

Dirichlet condition $u = g$,

$$\text{Neumann (flux) condition} = \frac{\partial u}{\partial \eta} = g, \tag{10.8}$$

$$\text{Mixed (Robin, radiation) condition} \alpha u + \beta \frac{\partial u}{\partial \eta} = g.$$

These conditions appear in many places throughout this book.

10.5.1 Some Motivating Examples

We analyse several elliptic boundary value problems in bounded and unbounded domains based around Laplace's equation. If the domain is unbounded, then in addition to boundary conditions a solution is generally required to satisfy a *condition at infinity*.

10.6 EXTENDED MAXIMUM-MINIMUM PRINCIPLES

We now generalise results from previous sections for the case of the Laplace equation to general elliptic equations. To this end, let L be the uniformly elliptic operator (8.1) in Chapter 8. We now define some notation:

$$C^0(\overline{\Omega}) = \text{space of functions continuous in } \overline{\Omega}$$

$$C^2(\Omega) = \text{space of twice continuously differentiable functions in } \Omega.$$

$|a_{ij}|, |b_j|, |c| \leq K, i, j = 1, \dots, n$ (coefficients of operator L)

$$\sum_{i,j=1}^n a_{ij}(x) \xi_i \xi_j \geq \lambda \sum_{j=1}^n \xi_j^2 \quad \forall \xi \in \mathbb{R}^n.$$

We are now ready to state:

Theorem 10.2 Let L be an elliptic operator as defined in Equation (8.1). Consider the elliptic boundary value problem with Dirichlet boundary conditions:

$$\begin{aligned} Lu &= f \text{ in } \Omega \\ u &= g \text{ on } \Gamma = \partial\Omega. \end{aligned} \tag{10.9}$$

If $u \in C^2(\Omega) \cap C^0(\bar{\Omega})$, $f \in C^0(\Omega)$, $g \in C^0(\Gamma)$, then we have the *a priori* estimate:

$$|u(x_0)| \leq \max_{x \in \Gamma} |g(x)| + M \max_{x \in \bar{\Omega}} |f(x)| \text{ where } M = M(\lambda, K).$$

We see that the (growth of) the solution of (10.9) is bounded by its input. We say that it is a *well-posed problem*.

Other boundary value problems are:

Neumann problem ($c < 0$):

$$\begin{aligned} Lu &= f \text{ in } \Omega \\ \frac{\partial u}{\partial \eta} &= g \text{ on } \Gamma. \end{aligned} \tag{10.10}$$

Mixed (Robin) problem ($\alpha\beta > 0$):

$$\begin{aligned} Lu &= f \text{ in } \Omega \\ \alpha u + \beta \frac{\partial u}{\partial \eta} &= g \text{ on } \Gamma. \end{aligned} \tag{10.11}$$

In general, the three main use cases that need to be addressed when studying PDEs are:

UC1: Uniqueness of the solution.

UC2: Continuous dependence of the solution on the data (well-posedness).

UC3: The existence of a solution to a boundary value problem.

These use cases are also applicable to time-dependent partial differential equations, as we shall see in later chapters. Maximum-minimum principles and *energy-integral arguments* can be used to address UC1 and UC2. For UC3, we can produce an analytic

solution (if we are lucky) but in many cases this is not possible. However, all is not lost because we can prove that a solution to a boundary value problem exists without having to construct it. Then we approximate the boundary value problem by numerical methods, for example the finite difference method or the finite element method.

Theorem 10.3 Define the operator in a planar bounded domain:

$$Lu = u_{xx} + u_{yy} + b_1(x, y)u_x + b_2(x, y)u_y. \quad (10.12)$$

If u satisfies $Lu = f$ in Ω with $f > 0$, then u attains its maximum on Γ and not inside Ω . If u satisfies $Lu \geq 0$ in Ω and if $u \leq M$ on Γ then $u \leq M$ in $\bar{\Omega}$ for Poisson's equation.

10.6.1 An Example

We now take a special case of Theorems 10.2 and 10.3. To this end, consider the boundary value problem (Duchateau and Zachman (1986), p. 29):

$$\begin{aligned} u_{xx} + u_{yy} &= f \text{ in } \Omega \\ u &= g \text{ on } \Gamma \end{aligned} \quad (10.13)$$

for which we prove the *a priori* estimate:

$$|u| \leq \max_{\Gamma} |g(x, y)| + M \max_{\bar{\Omega}} |f(x, y)| \quad (10.14)$$

where M is a constant that depends on the size of Ω .

Proof of (10.14).

We assume that $0 \leq x \leq a$ for given $a > 0$. We define the *comparison function*:

$$v = \max_{\Gamma} |g| + (e^a - e^x) \max_{\bar{\Omega}} |f|.$$

Defining $Lv \equiv v_{xx} + v_{yy}$, it is easy to check that:

$$\begin{aligned} v &\geq g \text{ on } \Gamma \\ Lv &= -e^x \max_{\bar{\Omega}} |f| \leq f \text{ in } \Omega. \end{aligned}$$

Hence $u \leq v$ in $\bar{\Omega}$. Similarly, choosing $w = -v$ we get:

$$\begin{aligned} w &\leq g \text{ on } \Gamma \\ Lw &\geq f \text{ in } \Omega \end{aligned}$$

and hence $u \geq -v$ in $\bar{\Omega}$. We can now conclude that:

$$|u| \leq \max_{\Gamma} |g| + M \max_{\bar{\Omega}} |f| \text{ where } M = e^a - 1.$$

10.7 SUMMARY AND CONCLUSIONS

In this chapter we introduced boundary value problems for elliptic partial differential equations and for the Laplace equation in particular. We discussed several techniques to give us more insights into understanding the use cases associated with these kinds of equations, and that will allow us to progress to more complex time-dependent PDEs and their numerical solutions in later chapters.

Fichera Theory, Energy Inequalities and Integral Relations

Space: the final frontier. These are the voyages of the starship Enterprise. Its five-year mission: to explore strange new worlds. To seek out new life and new civilizations. To boldly go where no man has gone before.

Captain James T. Kirk

11.1 INTRODUCTION AND OBJECTIVES

In this pivotal chapter we give an introduction to a number of mathematical and standardised techniques to analyse partial differential equations before approximating them by the finite difference method. The main technique is to transform a PDE on an unbounded domain to a modified PDE on a bounded domain. Having done that, we apply the Fichera theory to determine what the behaviour will be on the boundary.

The Fichera theory does not seem to be widely known in the computational finance community at the moment of writing.

11.2 BACKGROUND AND PROBLEM STATEMENT

The topics in this chapter can be difficult to grasp on a first reading. For this reason we take the simplest and most representative model problem to show the necessary steps that will also be used for more complex applications.

11.2.1 The 'Big Bang': Cauchy–Euler Equation

The *Cauchy–Euler equation* of order n is a linear homogeneous ordinary differential equation (ODE) with variable coefficients. It is an *equidimensional equation* and for

this reason it can be solved explicitly:

$$\sum_{j=0}^n a_j x^j \frac{d^j u}{dx^j} = 0, \quad 0 < x < \infty. \quad (11.1)$$

We are particularly interested in the *second-order Cauchy–Euler equation* that can be written in the following form without loss of generality:

$$x^2 \frac{d^2 u}{dx^2} + ax \frac{du}{dx} + bu = 0, \quad 0 < x < \infty. \quad (11.2)$$

This equation arises in several applications, for example solving Laplace's equation in polar coordinates. More importantly, it has the same form as the elliptic part of the one-factor and two-factor Black–Scholes PDEs. Thus, the results that we announce for (11.2) will also be applicable to finance problems. In this case we only need to explain the solutions once and then apply them to a wide range of one-factor and two-factor option pricing problems. The two major use cases are:

- U1: Transform (11.2) to an ODE with constant coefficients on the real line (infinite interval).
- U2: Transform (11.2) to an ODE on the unit interval.

The case U1 corresponds to the popular *log transformation* for the Black–Scholes equation, and U2 corresponds to the author's preferred approach because the resulting equation is now defined on a bounded interval and we can then consider how to define the associated boundary conditions. For U1 we will still need to either map the log-transformed differential equation to a bounded interval or truncate the infinite interval before we approximate the equation using the finite difference method. We now describe the execution of U1 and U2 for Equation (11.2).

For U1, we define the change of independent variable:

$$z = \log x, \quad \varphi(z) = u(x) = \varphi(\log x)$$

that results in an ODE with constant coefficients:

$$\frac{d^2 \varphi}{dz^2} + (a - 1) \frac{d\varphi}{dz} + b\varphi = 0, \quad -\infty < z < \infty. \quad (11.3)$$

The *characteristic equation* is $\lambda^2 + (a - 1)\lambda + b = 0$, roots $\{\lambda_1, \lambda_2\}$, and the solution is:

$$\varphi(z) = c_1 e^{\lambda_1 z} + c_2 e^{\lambda_2 z} \quad (c_1, c_2 \text{ constants}).$$

For U2, we define the transformation:

$$y = \frac{x}{x+1}; \quad x = \frac{y}{1-y}. \quad (11.4)$$

Some basic calculus produces an equivalent differential equation:

$$y^2 \frac{d}{dy} \left((1-y)^2 \frac{du}{dy} \right) + ay(1-y) \frac{du}{dy} + bu = 0 \quad (11.5)$$

or equivalently by expanding (11.5):

$$y^2(1-y)^2 \frac{d^2u}{dy^2} + \{ ay(1-y) - 2y^2(1-y) \} \frac{du}{dy} + bu = 0, \quad 0 < y < 1. \quad (11.6)$$

Equation (11.5) is in *self-adjoint form*, while Equation (11.6) contains both explicit diffusion and convection terms. In general, the latter equation is a more popular formulation in the literature for approximation by the finite difference method. Form (11.5) is useful in *integral formulations*.

It is subsequently possible to convert (11.3) to an equation on the interval $(-1,1)$ by a transformation, for example $w = \tanh z$, $z = \frac{1}{2} \log \left(\frac{1+y}{1-y} \right)$, to give the equivalent differential equation:

$$(1-w^2) \frac{d}{dw} \left((1-w^2) \frac{du}{dw} \right) + (a-1)(1-w^2) \frac{du}{dw} + bu = 0 \quad (11.7)$$

or equivalently:

$$(1-w^2)^2 \frac{d^2u}{dw^2} + (-2w(1-w^2) + (a-1)(1-w^2)) \frac{du}{dw} + bu = 0, \quad -1 < w < 1. \quad (11.8)$$

We are finished!

Summarising, U1 and U2 are special cases of the generic equations:

$$Lu \equiv a(x) \frac{d^2u}{dx^2} + b(x) \frac{du}{dx} + c(x)u = 0 \quad a < x < b \quad (11.9)$$

$$Lu \equiv a(x) \frac{d}{dx} \left(b(x) \frac{du}{dx} \right) + c(x)u = 0 \quad a < x < b. \quad (11.10)$$

The analysis in this section will reappear in many forms (and in C++ code) throughout this book.

11.3 WELL-POSED PROBLEMS AND ENERGY ESTIMATES

The discussion in Section 11.2 leads on to time-dependent PDEs of the form:

$$\frac{\partial u}{\partial t} = Lu, \quad a < x < b, \quad t > 0. \quad (11.11)$$

The elliptic operator L represents form (11.9) or (11.10). In general, we wish to prove that (11.11) is a *well-posed problem* (this topic is formally introduced in Chapter 12).

We motivate the approach by showing that the solution of (11.11) depends continuously on its data. To give an example, we (drastically) scope the problem by examining the one-factor Black–Scholes equation with zero interest rate and zero dividend:

$$\frac{\partial u}{\partial t} = \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2}, \quad 0 < S < \infty. \quad (11.12)$$

Applying the transformation $y = \frac{S}{S+\alpha}$, $S = \frac{\alpha y}{1-y}$ (α is a free parameter), this equation becomes:

$$\frac{\partial u}{\partial t} = \frac{1}{2}\sigma^2 y^2 \frac{\partial}{\partial y} \left\{ (1-y)^2 \frac{\partial u}{\partial y} \right\}, \quad 0 < y < 1. \quad (11.13)$$

The general approach when computing *energy estimates* is to multiply (11.13) by u on both sides, integrate by parts (and noticing that several terms disappear on the boundary) until we can invoke *Gronwall's inequality*, and then conclude that the solution is bounded by the data. The *initial integral* relation is:

$$\int_0^1 \frac{\partial u}{\partial t} u dy = \frac{1}{2}\sigma^2 \int_0^1 y^2 u \frac{\partial}{\partial y} \left((1-y)^2 \frac{\partial u}{\partial y} \right) dy \equiv \frac{1}{2}\sigma^2 I_1. \quad (11.14)$$

The steps in computing the right-hand integral I_1 are:

$$\begin{aligned} I_1 &= \int_0^1 y^2 u \frac{\partial}{\partial y} ((1-y)^2 \frac{\partial u}{\partial y}) dy \\ &= \left[y^2 u (1-y)^2 \frac{\partial u}{\partial y} \right]_0^1 - \int_0^1 \frac{\partial}{\partial y} (y^2 u) (1-y)^2 \frac{\partial u}{\partial y} dy \\ &= - \int_0^1 2y(1-y)^2 u \frac{\partial u}{\partial y} dy - \int_0^1 y^2 (1-y)^2 \left(\frac{\partial u}{\partial y} \right)^2 dy \\ &\leq - \int_0^1 y(1-y)^2 \frac{\partial u^2}{\partial y} dy = - [y(1-y)^2 u^2]_0^1 + \int_0^1 c(y) u^2 dy \\ &= \int_0^1 c(y) u^2 dy \quad (c(y) = \frac{d}{dy} y(1-y)^2). \end{aligned} \quad (11.15)$$

From (11.15) and (11.14) we get:

$$\frac{1}{2} \frac{d}{dt} \int_0^1 u^2 dy \leq \frac{1}{2}\sigma^2 \int_0^1 c(y) u^2 dy \leq M \int_0^1 u^2 dy. \quad (11.16)$$

By integrating (11.16) on $(0, t)$ we get:

$$\int_0^1 u^2(y, t) dy \leq 2M \int_0^t \int_0^1 u^2(y, s) dy ds + \int_0^1 u^2(y, 0) dy. \quad (11.17)$$

Finally, we conclude that the solution is bounded by its initial condition (pay-off). We are finished. We shall apply the same analysis to the CIR problem in Chapter 25 to reproduce the famous Feller condition.

11.3.1 Time to Reflect: What Have We Achieved and What's Next?

The main reason for writing this chapter is to unify several techniques and methods that allow us to produce unambiguous and implementable specifications of initial boundary value problems. To date, this endeavour has been haphazard, and ad hoc solutions seem to be the rule rather than the exception. The two major challenges to resolve are first to determine how a PDE behaves at the *near field* (where the underlying variable is zero) and how to transform a PDE (using *domain truncation*, for example) on an unbounded domain to a PDE on a bounded domain. Some of the problems and assumptions that we wish to address are:

- Lack of awareness of the Fichera theory for problems with non-negative characteristic form and its application to the Black–Scholes equation, allowing us to rigorously define near-field behaviour.
- Using the *energy method* (Kreiss and Lorenz (2004)) to prove well-posedness of initial boundary value problems. An added benefit of this method is that it shows which boundary conditions to apply in order to ensure well-posedness. We shall give an example in Chapter 25 when we discuss the Cox–Ingersoll–Ross (CIR) model and we show how to recover the *Feller condition* using this approach. The same analysis can also be applied to the Heston model.
- ‘Guesstimates’ using financial heuristics concerning how to prescribe boundary conditions. This approach may or may not always work in practice. The imposition of ad hoc and *premature* boundary conditions may be incorrect, leading to mathematical nonsense. In general, we take the viewpoint that PDEs in finance do not have associated boundary conditions but rather they have limiting or asymptotic near-field and far-field behaviour that we quantify and then approximate by stable and accurate *numerical boundary conditions*. In other words, we can choose whatever *numerical boundary condition* that best satisfies our requirements. We give examples of spread options in Chapter 23, and we shall see that several styles of numerical boundary condition are both stable and accurate.
- We prefer domain transformation to domain truncation because the former does not introduce errors and it leads to well-posed PDEs that can be approximated by finite differences. On the other hand, we can use the easier to implement domain truncation technique when we are sure that its use does not affect the integrity of our analysis or accuracy of the associated finite difference schemes.
- Using *weak (integral) formulations* of initial boundary value problems in combination with integration by parts and Green’s function and identities provides insights into boundary behaviour (Achdou and Pironneau (2005)). In fact, the finite element method (FEM) uses this principle.
- The Fichera theory and the energy method seem to be two sides of the same coin.

In short, the Fichera theory, the energy method and domain transformation are powerful tools to specify problems as a preprocessing stage before we approximate these problems by the finite difference method.

We now proceed with an introduction to the Fichera theory and its applications in finance.

11.4 THE FICHERA THEORY: OVERVIEW

The theory of partial differential equations with non-negative characteristic forms was introduced by the Italian mathematician Gaetano Fichera (Fichera (1956)). The monograph by Oleinik and Radkevic (1973) is the standard references on the subject. A good source of examples can be found in Meyer (2015). The first application of Fichera theory in finance seems to be Duffy (2009a), elaborated in Duffy (2018).

11.5 THE FICHERA THEORY: THE CORE BUSINESS

In this section we define a process to help find the mathematically and financially appropriate boundary conditions for general parabolic partial differential equations with *non-negative characteristic form* (Oleinik and Radkevic (1973)) and in particular for one-factor and multi-factor PDEs in derivatives pricing. To this end, we discuss the *Fichera theory*.

The Fichera theory is applicable to a range of elliptic, parabolic and hyperbolic PDEs, and it is particularly useful for PDEs whose coefficients are zero on certain boundaries of a bounded domain Ω in n -dimensional space (for more information, see Fichera (1956), Oleinik and Radkevic (1973)). We depict this domain, its boundary Σ and *inward unit normal* ν in Figure 11.1. Let us examine the elliptic equation defined by:

$$Lu \equiv \sum_{i,j=1}^n a_{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=1}^n b_i \frac{\partial u}{\partial x_i} + cu = f \text{ in } \Omega \quad (11.18)$$

where:

$$\sum_{i,j=1}^n a_{ij} \xi_i \xi_j \geq 0 \text{ in } \Omega \cup \Sigma \quad \forall \xi = (\xi_1, \dots, \xi_n)^\top \in \mathbb{R}^n. \quad (11.19)$$

The *characteristic form* (11.19) is strictly positive in most cases, but we are interested in finding the subsets of the boundary Σ where it is identically zero. To this end, we define:

$$\Sigma_3 = \left\{ x \in \Sigma : \sum_{i,j=1}^n a_{ij} \nu_i \nu_j > 0 \right\}. \quad (11.20)$$

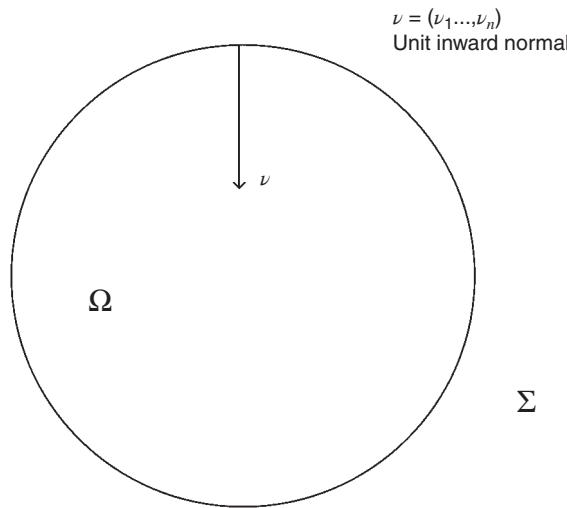


FIGURE 11.1 Region and boundary.

On the boundary where the characteristic form is zero (the *characteristic boundary* that is, $\Sigma - \Sigma_3$) we define the *Fichera function*:

$$b \equiv \sum_{i=1}^n \left(b_i - \sum_{k=1}^n \frac{\partial a_{ik}}{\partial x_k} \right) \nu_i \quad (11.21)$$

where ν_i is the i th component of the inward unit normal ν on Σ . Having computed the Fichera function, we then determine its sign on all parts of the characteristic boundary. There are three mutually exclusive options:

$$\begin{aligned} \Sigma_0 &: b = 0 \\ \Sigma_1 &: b > 0 \\ \Sigma_2 &: b < 0. \end{aligned} \quad (11.22)$$

In other words, the boundary consists of the following sub-boundaries:

$$\Sigma \equiv \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup \Sigma_3. \quad (11.23)$$

We demand that no boundary conditions (BC) are allowed when the Fichera function is zero or positive (in other words, on Σ_0 and Σ_1), and then the PDE (11.11) can degenerate to a lower-order PDE or ODE on these boundaries. When $b < 0$ (that is, on Σ_2), we need to define a boundary condition. What that boundary condition is depends on the context and needs to be investigated on a case-by-case basis.

We write parabolic PDEs in the form:

$$\frac{\partial u}{\partial t} = Lu + f \text{ or } -\frac{\partial u}{\partial t} + Lu = -f \quad (11.24)$$

where L is an elliptic operator. Then the same conclusions concerning characteristic and non-characteristic boundaries hold as in the elliptic case. In other words, we focus on the elliptic part of the PDE where we wish to calculate the Fichera function.

We summarise the steps to take when applying the Fichera theory to find the boundary conditions for initial boundary value problems. Readers may have difficulty in calculating the Fichera function. The mathematics is not difficult to understand, but the steps must be correctly executed in the following prescribed order:

1. Determine the boundary of the domain and its unit inward normal. In many cases the boundary is the union of hyperplanes parallel to the coordinate axes.
2. Calculate the characteristic form (Equation (11.19)) that leads to the characteristic boundary.
3. Calculate the Fichera function (Equation (11.21)) on the characteristic boundary.
4. Determine the subset of the boundary where no boundary conditions are needed (see choices in (11.22)). Determine what kind of equation is defined on this subset.
5. In the case where the Fichera function is negative, we must also determine what the boundary condition will be. This is a Dirichlet boundary condition in many cases.

A visualisation can be seen in Figure 11.2 in Section 11.6.3 for the case of the Heston model.

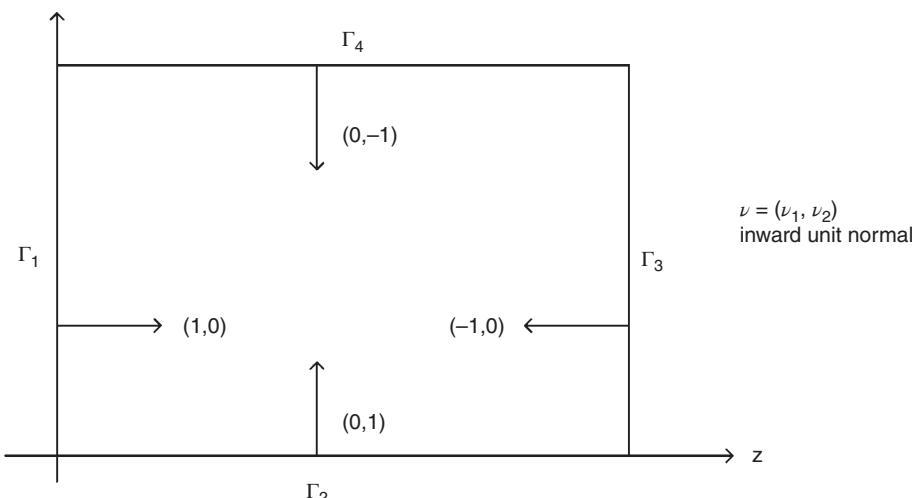


FIGURE 11.2 Boundaries for Heston model

We now apply the Fichera function (11.21) to the transformed Black–Scholes PDE:

$$\frac{\partial V}{\partial t} = \frac{1}{2}\sigma^2y^2\frac{\partial}{\partial y}\left\{(1-y)^2\frac{\partial V}{\partial y}\right\} + ry(1-y)\frac{\partial V}{\partial y} - rV$$

or equivalently:

$$\frac{\partial V}{\partial t} = \frac{1}{2}\sigma^2y^2(1-y)^2\frac{\partial^2 V}{\partial y^2} + \{ry(1-y) - \sigma^2y^2(1-y)\}\frac{\partial V}{\partial y} - rV.$$

In this case the Fichera function becomes:

$$b = \{(ry(1-y) - \sigma^2y(1-y)^2\} v(y) \quad (11.25)$$

where:

$$v(0) = 1, v(1) = -1.$$

We thus see that the Fichera function is zero at both end points $y = 0$ and at $y = 1$, and then no boundary conditions need be prescribed at these points. Instead, the PDE degenerates to the following ordinary differential equation at these points:

$$\frac{\partial V}{\partial t} + rV = 0 \text{ at } y = 0, y = 1. \quad (11.26)$$

The solution to (11.26) is given by:

$$V(y, t) = C(y)e^{-rt} \text{ at } y = 0, y = 1 \quad (11.27)$$

where the factor $C(y)$ can be found by demanding *compatibility* between the solution (11.27) and the initial condition corresponding to the PDE at the points $y = 0$, $y = 1$. Doing this will lead to the well-known boundary conditions for the Black–Scholes equation for call and put options. For example, in the case of a put option we have $C(y) = K$.

11.6 THE FICHERA THEORY: FURTHER EXAMPLES AND APPLICATIONS

We take a number of representative examples from the perspective of the Fichera theory.

11.6.1 Cox–Ingersoll–Ross (CIR)

In this case we examine the PDE that prices a *zero-coupon bond* under a *Cox–Ingersoll–Ross (CIR) interest-rate model*:

$$\frac{\partial B}{\partial t} + \frac{1}{2}\sigma^2r\frac{\partial^2 B}{\partial r^2} + (a - cr)\frac{\partial B}{\partial r} - rB = 0. \quad (11.28)$$

Please note that we are using backward time in this case, hence the sign difference when compared with Equation (11.11). You should be aware of this subtle point, but it has no bearing on the current discussion. Using the definition in Equation (11.21), we see that the Fichera function is given by:

$$b = ((a - cr) - \sigma^2/2)\nu \quad (11.29)$$

where ν is the inward unit normal at $r = 0$ ($\nu = 1$) or at $r = r_{\max}$ ($\nu = -1$).

We are interested in the case $r = 0$ (because this is the only characteristic boundary for this problem), and we see that each choice in (11.29) can be valid depending on the relative sizes of the parameters a and σ :

$$\begin{aligned} \Sigma_2 : b < 0 &\rightarrow \sigma > \sqrt{2a} \quad (\text{BC needed}) \\ \Sigma_0 : b = 0 &\rightarrow \sigma = \sqrt{2a} \quad (\text{No BC needed}) \\ \Sigma_1 : b > 0 &\rightarrow \sigma < \sqrt{2a} \quad (\text{No BC needed}). \end{aligned} \quad (11.30)$$

In the last two cases we see that no boundary condition (BC) is allowed (or needed) and then the PDE (11.28) reduces to the first-order hyperbolic PDE:

$$\frac{\partial B}{\partial t} + a \frac{\partial B}{\partial r} = 0 \quad (11.31)$$

on $r = 0$ which is a characteristic boundary. These results are consistent with the conclusions in Tavella and Randall (2000), pp. 126–28. In general, we need to solve (11.31) either analytically or numerically. From a financial perspective, the third condition in (11.30) when the Fichera function is positive states that the interest rate cannot become negative. The inequality in this case is called the *Feller condition* for the CIR process. We have also investigated it for the process for the Heston square-root model, and we have reproduced the well-known *Feller condition*:

$$\theta \geq \frac{1}{2}\sigma^2 \quad (a = \kappa\theta \text{ in this case}). \quad (11.32)$$

For a discussion on solutions of the Heston PDE using finite difference methods, see the thesis by Sheppard (2007).

We note that we can reproduce the Feller condition (11.32) using the energy method, as we shall see in Chapter 25. The condition emerges naturally from the corresponding integral formulation.

11.6.2 Heston Model Fundamentals

The Heston model supports stochastic volatility. The Black–Scholes equation assumes that volatility is flat (constant). The corresponding time-dependent convection-diffusion-reaction PDE contains a mixed-derivative term that originates from the correlation between the two underlying processes for the asset price and variance. Much has been written on this model, and the recurring themes from a PDE perspective

are 1) specifying appropriate boundary conditions, 2) mapping the PDE to a truncated domain and 3) approximating the mixed derivative term by finite differences.

We give an overview of the (heuristic) approach to defining boundary conditions for the Heston model. We then proceed to formalising the problem. In short, we map the problem into the unit square, and we apply the Fichera theory to show that boundary conditions are not needed, for example at $v = 0$, where we retrieve the Feller condition.

Since there are two factors in the Heston model, we need two SDEs. First, the spot asset price satisfies the SDE:

$$dS_t = \mu S_t dt + \sqrt{v(t)} S_t dW_t^{(1)} \quad (11.33)$$

where:

- S_t = spot price
- $W_t^{(1)}$ = Wiener process
- $v(t)$ = variance
- μ = (risk neutral) drift.

Second, the variance $v(t)$ satisfies an *Ornstein–Uhlenbeck* process defined by the SDE:

$$d\sqrt{v(t)} = -\beta\sqrt{v(t)}dt + \sigma dW_t^{(2)}.$$

It can be shown that:

$$dv(t) = \kappa[\theta - v(t)]dt + \sigma\sqrt{v(t)}dW_t^{(2)} \quad (11.34)$$

where:

- σ = volatility of the volatility
- $0 < \theta$ = long term variance
- $0 < \kappa$ = rate of mean reversion
- $W_t^{(2)}$ is a Wiener process and ρ is the correlation value.

The correlation between the two Wiener processes is given by:

$$dW_t^{(1)}dW_t^{(2)} = \rho dt. \quad (11.35)$$

In general, an increase in ρ generates an asymmetry in the distribution, while a change of volatility of variance σ results in a higher kurtosis.

Finally (Heston (1993)), the PDE for a contingent claim U is given by:

$$\frac{\partial U}{\partial t} + L_s U + L_v U + \rho\sigma\nu S \frac{\partial^2 U}{\partial S \partial v} = 0 \quad (11.36)$$

where:

$$\begin{aligned} L_S U &\equiv \frac{1}{2} v S^2 \frac{\partial^2 U}{\partial S^2} + r S \frac{\partial U}{\partial S} - r U = 0 \\ L_v U &\equiv \frac{1}{2} \sigma^2 v \frac{\partial^2 U}{\partial v^2} + \{ \kappa[\theta - v(t)] - \lambda(S, v, t) \} \frac{\partial U}{\partial v} \end{aligned}$$

and λ = market price of volatility risk.

Let us examine Equation (11.36). We see that the PDE is a convection-diffusion-reaction equation in two variables and there is a mixed derivative term appearing in the equation.

We now discuss how to augment the PDE (11.36) by a variety of boundary conditions and let us focus on European options. In general, we define boundary conditions at the following near and far fields:

$$\begin{aligned} S &\rightarrow 0, \quad S \rightarrow \infty \\ v &\rightarrow 0, \quad v \rightarrow \infty. \end{aligned} \tag{11.37}$$

Thus, we give some kind of boundary condition at each of these four boundaries. (Intuitively, we need four conditions because integrating the second derivatives in S and v in the PDE (11.36) gives us four constants that can be found from the four conditions in equation (11.37).)

We now look at some particular examples of boundary conditions from the literature. Caveat: the choice of the boundary conditions is based on heuristic reasoning in the following discussion.

11.6.2.1 Standard European Call Option

This is the formulation as first mentioned in Heston (1993). When $S = 0$ we consider the call to be worthless; when S becomes very large, we use a Neumann boundary condition which is similar to a linearity boundary condition. When v is 0 we assume that the PDE (11.36) is satisfied on the line $v = 0$. Finally, when v is very large, we assume that the option behaves as a standard European option. Summarising, the boundary conditions become:

$$U(0, v, t) = 0 \quad (S = 0) \tag{11.38}$$

$$\frac{\partial U}{\partial S}(\infty, v, t) = 1 \quad (S = \infty) \tag{11.39}$$

$$\frac{\partial U}{\partial t} + r S \frac{\partial U}{\partial S} - r U + \kappa \theta \frac{\partial U}{\partial v} = 0 \quad (v = 0) \tag{11.40}$$

$$U(S, \infty, t) = S \quad (v = \infty). \tag{11.41}$$

These boundary conditions are easy to approximate numerically, with the possible exception of (11.40), which we must handle with care. The boundary conditions (11.38) to (11.41) are those as specified in Heston (1993) and are justified based on financial heuristic reasoning. Other variations have also been discussed in the literature.

11.6.2.2 European Put Options

We define the boundary conditions for a put option:

$$U(0, v, t) = K$$

$$\frac{\partial U}{\partial S}(\infty, v, t) = 0$$

$$U(S, 0, t) = \max(K - S, 0)$$

$$\frac{\partial U}{\partial v}(S, \infty, t) = 0.$$

These boundary conditions are easy to approximate.

11.6.2.3 Other Kinds of Boundary Conditions

Another vision and interpretation on how to define boundary conditions for the Heston model is:

$$\frac{\partial U}{\partial t} - rU + L_v U = 0 \quad (S = 0)$$

$$\left. \begin{array}{l} U = S(\text{call}) \\ U = 0(\text{put}) \end{array} \right\} (S \rightarrow \infty)$$

$$\frac{\partial U}{\partial t} + rS \frac{\partial U}{\partial S} - rU + \kappa\theta \frac{\partial U}{\partial v} = 0 \quad (v \rightarrow 0) \quad (11.42)$$

$$\frac{\partial U}{\partial t} + \frac{1}{2}vS^2 \frac{\partial^2 U}{\partial S^2} + rS \frac{\partial U}{\partial S} - rU = 0 \quad (v \rightarrow \infty). \quad (11.43)$$

As before, some of the boundary conditions (for example, equation (11.43)) have an analytical solution. If this is not possible, we must resort to a finite difference scheme to approximate (11.42) and (11.43), for example.

Finally, in their article In 't Hout and Foulon (2010) approximate the Heston model for call options using the following heuristic boundary conditions in an ADI framework:

$$\begin{aligned} U(S, v, 0) &= \max(0, S - K) \\ U(0, v, t) &= 0, \quad 0 \leq t \leq T \\ \frac{\partial U}{\partial S}(S_{\max}, v, t) &= e^{-r_f t}, \quad 0 \leq t \leq T \\ U(S, V_{\max}, t) &= Se^{-r_f t}, \quad 0 \leq t \leq T, \end{aligned} \quad (11.44)$$

when r_f is the foreign interest rate.

They use the truncation values:

$$S_{\max} = 8K$$

$$V_{\max} = 5. \quad (11.45)$$

In general, we do not use ADI in this book, preferring to use splitting methods that we discuss in Chapters 18 and 22. A good project could be the application of the Method of Lines (MOL) (see Chapter 20) and the Alternating Direction Explicit (ADE) method (see Chapter 19) to this problem using combinations of domain truncation and transformation in S and v . For example, we could try domain transformation in S and domain truncation in v . See Sheppard (2007), in which Yanenko splitting is used in combination with domain truncation in S and v for the Heston model.

11.6.3 Heston Model by Fichera Theory

We examine the Heston PDE (11.36) again. For convenience, we write it as a forward-in-time PDE, and we let the market price of volatility risk be zero. We also redefine the independent variables as $x = S$ and $y = v$ for convenience (readability).

We define new variables as follows:

$$z = \frac{x}{x+1}, w = \frac{y}{y+1} \left(x = \frac{z}{1-z}, y = \frac{w}{1-w} \right). \quad (11.46)$$

Then we can write Equation (11.36) in the modified form using basic differentiation:

$$\frac{\partial u}{\partial t} = L_1 u + L_2 u + L_3 u, \quad 0 < z, w < 1 \quad (11.47)$$

where:

$$\begin{aligned} L_1 u &= \frac{1}{2} \left(\frac{w}{1-w} \right) z^2 \frac{\partial}{\partial z} ((1-z)^2 \frac{\partial u}{\partial z}) + r z (1-z) \frac{\partial u}{\partial z} - r u \\ L_2 u &= \frac{1}{2} \sigma^2 w (1-w) \frac{\partial}{\partial w} ((1-w)^2 \frac{\partial u}{\partial w}) + \kappa (\theta (1-w)^2 - w (1-w)) \frac{\partial u}{\partial w} \\ L_3 u &= \rho \sigma z w (1-z) (1-w) \frac{\partial^2 u}{\partial z \partial w}. \end{aligned}$$

Using Figure 11.2 as a reference and computing the Fichera function b_F on each boundary, we conclude that:

$$\begin{aligned} b_F &= 0 \text{ on } \Gamma_1 \\ b_F &= 0 \text{ on } \Gamma_3 \\ b_F &= \kappa \theta - \frac{1}{2} \sigma^2 \text{ on } \Gamma_2 \text{ (Feller condition)} \\ b_F &= 0 \text{ on } \Gamma_4. \end{aligned} \quad (11.48)$$

In particular, we can now see how Equation (11.40) was arrived at as well as the Feller condition (11.32). We employ the same technique in the same way for other two-factor models. We note that we have not yet discussed which boundary conditions to prescribe. It is too premature, as this is a topic to be addressed when approximating the solution of (11.47) by finite differences in combination with *numerical boundary conditions*.

11.6.4 First-Order Hyperbolic PDE in One and Two Space Variables

A useful application of the Fichera theory is to first-order (convection) advection PDEs, that is PDEs that have no diffusion term. We take a model hyperbolic equation. It is of no consequence that one of the independent variables is called t (representing time); we could have used y , for example:

$$-\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = 0, \quad 0 < x < 1, \quad 0 < t < T \quad (a > 0). \quad (11.49)$$

Since we need only two conditions on two boundaries, we apply Fichera theory to show us that they correspond to $t = 0$ (also known as an *initial condition*) and $x = 1$, and they are called *inflow boundaries*. The boundaries $t = T$ and $x = 0$ are then called *outflow (downstream) boundaries*. It is thus important to know in which direction information is flowing. Finite difference schemes should preserve this regime by using *upwind schemes*, for example. We shall see examples in Chapter 24 when we discuss Asian options.

We now consider the problem (11.49) in modified form on a semi-infinite interval:

$$-\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = 0, \quad 0 < x < \infty, \quad 0 < t < T \quad (a > 0). \quad (11.50)$$

We map the problem to the unit interval to get:

$$\begin{aligned} y &= \frac{x}{x+1} \\ -\frac{\partial u}{\partial t} + a(1-y)^2 \frac{\partial u}{\partial y} &= 0, \quad 0 < y < 1, \quad 0 < t < T. \end{aligned} \quad (11.51)$$

Then on $y = 1$ the PDE degenerates to an ODE there that can be solved exactly to give a candidate for a numerical boundary condition. This is also the technique used when solving the Black–Scholes equation. We must be careful to employ one-sided difference schemes at the outflow boundary and to avoid spurious reflections there (see Vichnevetsky and Bowles (1982)).

We examine first-order hyperbolic PDEs in three independent variables (for example Equation (11.40) above). We write (11.40) in the more convenient form (as a terminal value problem):

$$\frac{\partial U}{\partial t} + \alpha \frac{\partial U}{\partial S} + \beta \frac{\partial U}{\partial v} + bU = 0 \quad (v = 0) \quad (11.52)$$

where the new coefficients are defined by:

$$\alpha = rS, \quad \alpha > 0$$

$$b = -\kappa, \quad b < 0$$

$$\beta = \theta, \quad \beta > 0.$$

We mention that the signs of the coefficients α and β determine where the information in the system is coming from.

One final remark: the use of the Fichera theory applied to first-order hyperbolic PDEs allows us to locate those boundaries where boundary conditions should be specified. It complements heuristic reasoning to find the appropriate boundary conditions. Traditionally, these kinds of equations and their generalisations are found in fluid and gas dynamics application. We discuss finite difference schemes for (11.52) in Chapter 25 (Section 25.6).

11.7 SOME USEFUL THEOREMS

For completeness, we introduce a number of theorems that can be called ‘integration by parts in two and three dimensions’. These theorems are used for problems that are posed in *weak or variational form* (see Friedman (1982), Achdou and Pironneau (2005)). In fact, the starting point for the finite element method (FEM) is to map a PDE to integral form. Due to time limitations, we have unfortunately been unable to discuss FEM in this book.

We first develop some notation. We assume that we are working in three-dimensional space (similar results will also hold in the two-dimensional case). A *vector* is a quantity having both magnitude and direction, Examples are velocity, force and displacement. We represent vectors as 3-tuples, for example $A = (A_1, A_2, A_3)$, $B = (B_1, B_2, B_3)$. The *magnitude* of a vector and the *dot product* of two vectors are respectively:

$$\begin{aligned} |A| &= \sqrt{A_1^2 + A_2^2 + A_3^2} \\ A \cdot B &= A_1 B_1 + A_2 B_2 + A_3 B_3. \end{aligned} \tag{11.53}$$

The vector-valued *gradient function* of a *scalar field* $\phi(x, y, z)$ is:

$$\nabla \phi = \left(\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}, \frac{\partial \phi}{\partial z} \right) \equiv \text{grad } \phi. \tag{11.54}$$

The gradient is a vector field as can be seen from the example:

$$\begin{aligned} \phi &= 3x^2y - y^3z^2 \\ \nabla \phi &= (6xy, 3x^2 - 3y^2z^2, -2y^3z). \end{aligned}$$

The *divergence* is a scalar-valued function of a vector field:

$$\begin{aligned} V &= (V_1, V_2, V_3) \\ \text{div } V &= \nabla \cdot V = \frac{\partial V_1}{\partial x} + \frac{\partial V_2}{\partial y} + \frac{\partial V_3}{\partial z}. \end{aligned} \tag{11.55}$$

An example is:

$$V = (x^2z, -2y^3z^2, xy^2z),$$

$$\nabla \cdot V = 2xz - 6y^2z^2 + xy^2.$$

We can recover the *Laplace operator* by taking the dot product of the gradient operator with itself:

Let $\phi = \phi(x, y, z)$ be a scalar field; then

$$\nabla\phi = \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z} \right), \nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right).$$

Hence:

$$\begin{aligned} \nabla \cdot \nabla\phi &= \frac{\partial}{\partial x} \left(\frac{\partial\phi}{\partial x} \right) + \frac{\partial}{\partial y} \left(\frac{\partial\phi}{\partial y} \right) + \frac{\partial}{\partial z} \left(\frac{\partial\phi}{\partial z} \right) \\ &= \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = \nabla^2\phi \equiv \Delta\phi. \end{aligned} \quad (11.56)$$

Hence $\nabla \cdot \nabla = \nabla^2 = \Delta$, the Laplacian operator.

11.7.1 Divergence (Gauss–Ostrogradsky) Theorem

This theorem states that the volume (three-dimensional) integral of the divergence of a vector field F over a domain V is equal to the flux of F through the boundary S of V , oriented in terms of its outward normal.

Suppose V is a compact volume in \mathbb{R}^3 with smooth boundary $\partial V \equiv S$. Let F be a continuously differentiable vector field, then:

$$\int \int_V \int (\nabla \cdot F) dv = \int_S \int (F \cdot n) ds$$

n = outward pointing normal to surface S . (11.57)

The left-hand side (volume integral) represents the total amount of the sources in the volume V , while the right-hand side (surface integral) represents the total flow across the boundary S .

This theorem describes fluid flowing in some region, and it calculates the rate at which fluid flows out of the region by adding up the sources within the region and subtracting the sinks. The divergence describes the strength of the source or sink. Thus, the integral of the vector field's divergence should equal the integral of the vector field over the region's boundary.

11.7.2 Green's Theorem/Formula

Green's theorem is a special case of the divergence theorem in two dimensions. Let $L = L(x, y)$, $M = M(x, y)$, then:

$$\int_C (Ldx + Mdy) = \int_D \int \left(\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y} \right) dx dy$$

(path integration along C is *anticlockwise*).

11.7.3 Green's First and Second Identities

Let $\phi = \phi(x, y, z)$ and $\psi = \psi(x, y, z)$ be scalar fields. Then *Green's first and second identities* are:

$$\int \int_V \int \phi \Delta \psi dV = \int_S \int \phi \frac{\partial \psi}{\partial n} dS - \int_V \int \int \nabla \phi \cdot \nabla \psi dV$$

and:

$$\int \int_V \int (\phi \Delta \psi - \psi \Delta \phi) dV = \int_S \int (\phi \nabla \psi - \psi \nabla \phi) \cdot n dS = \int_S \int \left(\phi \frac{\partial \psi}{\partial n} - \psi \frac{\partial \phi}{\partial n} \right) dS$$

respectively. We can get the second identity by interchanging ϕ and ψ from the first identity and subtracting.

Here we use the definition of the Laplacian:

$$\nabla^2 = \Delta \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}.$$

11.8 SUMMARY AND CONCLUSIONS

In this chapter we gave an introduction to a number of mathematical techniques for partial differential equations in order to make them more suitable for processing by the finite difference method. The main technique is to transform a PDE on an unbounded domain to a modified PDE on a bounded domain. Having done that, we apply the Fichera theory to determine what the behaviour will be on the boundary.

The methods used here are universal in the sense that they can be used for a range of linear and nonlinear PDEs in finance.

CHAPTER 12

An Introduction to Time-Dependent Partial Differential Equations

Time and Space . . . It is not nature which imposes them upon us, it is we who impose them upon nature because we find them convenient.

Henri Poincaré

12.1 INTRODUCTION AND OBJECTIVES

The goal of this chapter is to develop the mathematical theory of time-dependent equations as preparation for PDE models in computational finance. We focus on the two-dimensional heat equation because first, its study follows naturally from previous chapters that discuss the Laplace equation and its associated properties and boundary value problems. Second, we lay the foundations for work on convection-diffusion-reaction equations and their numerical approximation by the finite difference method. In particular, the *maximum-minimum principle* for PDEs and finite difference schemes will be important. We also discuss the concept of *well-posed problems*.

12.2 NOTATION AND PREREQUISITES

The notation, results and techniques follow from Chapter 10. We take an incremental approach by trying to preserve the same notation as in previous chapters and proceeding in well-defined steps.

12.3 PREAMBLE: SEPARATION OF VARIABLES FOR THE HEAT EQUATION

We introduce two-dimensional PDEs by examining the heat equation on a bounded domain with a combination of Dirichlet and Neumann boundary conditions. In this case we apply the Separation of Variables method that we introduced in Section 10.4 for the Laplace equation. In the current case the solution will be a product of three terms in x, y and t , and the method uses the same tricks as in Section 10.4. Furthermore, we see how *initial conditions* are incorporated into the *initial boundary value problem* formulation (IBVP) for the heat equation.

The *heat equation* is one of the most famous equations in mathematical physics. In two dimensions it is given by:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}. \quad (12.1)$$

This equation is usually defined on a bounded, semi-infinite or infinite two-dimensional region. On the boundaries we define boundary conditions as well as an associated initial condition when $t = 0$. For example, on a bounded rectangle $[0, L] \times [0, M]$ we can define the Dirichlet boundary conditions on one part of the boundary and Neumann boundary conditions on the other part:

Dirichlet:

$$u(x, 0, t) = 0, \quad 0 < x < L \quad (12.2)(a)$$

$$u(x, M, t) = 0, \quad 0 < x < L.$$

Neumann:

$$\frac{\partial u}{\partial x}(0, y, t) = 0, \quad 0 < y < M \quad (12.2)(b)$$

$$\frac{\partial u}{\partial x}(L, y, t) = 0, \quad 0 < y < M.$$

Finally, we prescribe the initial condition:

$$u(x, y, 0) = f(x, y), \quad 0 \leq x \leq L, \quad 0 \leq y \leq M. \quad (12.3)$$

We call the equations (12.1), (12.2) and (12.3) the *initial boundary value problem (IBVP) for the heat equation*.

We can apply the Separation of Variables technique to find a solution to the IBVP (12.1), (12.2) and (12.3) in the form of a *biorthogonal Fourier series*. The details are discussed in Tolstov (1962), for example, and we summarise the main results here. To this end, we seek a solution in the form:

$$u(x, y, t) = X(x)Y(y)T(t).$$

The components are given by:

$$\begin{aligned} Y_n(y) &= A_n \sin \frac{n\pi y}{M}, \quad n = 1, 2, \dots, \\ X_m(x) &= B_m \cos \frac{m\pi x}{L}, \quad m = 0, 1, 2, \dots \\ T &= \exp(-\pi^2 [m^2/L^2 + n^2/M^2] t). \end{aligned}$$

Then:

$$u(x, y, t) = \sum_{\substack{m=0 \\ n=1}}^{\infty} u_{mn}(x, y, t)$$

where:

$$u_{mn}(x, y, t) = A_{mn} \cos \frac{m\pi x}{L} \sin \frac{n\pi y}{M} e^{-\pi^2 \left[\frac{m^2}{L^2} + \frac{n^2}{M^2} \right] t}.$$

We find the constant term A_{mn} in this last equation by using the initial condition (12.3) and some integration. When $t = 0$ we get:

$$f(x, y) = \sum_{\substack{m=0 \\ n=1}}^{\infty} A_{mn} \cos \frac{m\pi x}{L} \sin \frac{n\pi y}{M}$$

where the coefficients are given by:

$$\begin{aligned} A_{0n} &= \frac{2}{LM} \int_0^M \int_0^L f(x, y) \sin \left(\frac{n\pi y}{M} \right) dx dy \quad (m = 0) \\ A_{mm} &= \frac{4}{LM} \int_0^M \int_0^L f(x, y) \cos \left(\frac{m\pi x}{L} \right) \sin \left(\frac{n\pi y}{M} \right) dx dy \quad (m \neq 0). \end{aligned}$$

You can use this example in benchmarks to test the effectiveness of FDM schemes. In general, you will need to use two-dimensional numerical quadrature to compute the above integrals.

We investigate the initial boundary value problem for the one-dimensional diffusion equation (Tolstov (1962)) that describes heat flow in a rod of length L :

$$\begin{cases} \frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2}, & 0 < x < L, \quad t > 0 \\ u(x, 0) = f(x), & 0 \leq x \leq L \\ u(0, t) = A, \quad u(L, t) = B, & t > 0. \end{cases} \quad (12.4)$$

In this case we can assume without loss of generality that $L = 1$. Here a , A and B are constants.

We can find a solution to the system (12.4) in the case when $A = B = 0$ and $a = 1$ by the method of *Separation of Variables* (Kreider, Kuller, Ostberg and Perkins (1966)). In this case the *analytical solution* is given by:

$$u(x, t) = \frac{8}{\pi^2} \sum_{n=1}^{\infty} \frac{1}{n^2} \left(\sin \frac{n\pi}{2} \right) (\sin n\pi x) \exp(-n^2\pi^2 t) \quad (12.5)$$

and we use this solution as the benchmark against which numerical solutions can be compared.

Students of mathematics and various branches of engineering will be familiar with this method, but we thought that it is useful to discuss it for those who do not necessarily have this background. In fact, the method touches on other areas of mathematics and applications to finance, for example Bessel's functions (Watson (2006)) and Sturm-Liouville problems (Kreider, Kuller, Ostberg and Perkins (1966)).

More generally, having an analytical solution is useful because it can be used as a baseline for testing numerical solutions for the heat equation.

12.4 WELL-POSED PROBLEMS

The mathematical term *well-posed problem* stems from a definition given by twentieth-century French mathematician Jacques Hadamard. He believed that mathematical models of physical phenomena should have the properties that:

1. A solution exists.
2. The solution is unique.
3. The solution's behavior changes continuously with the initial and boundary conditions.

Examples of well-posed problems include the Dirichlet problem for Laplace's equation and the heat equation with given initial conditions. A simple example is the initial boundary value problem for a *first-order hyperbolic equation*:

$$\begin{aligned} a) \quad & \frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = 0, \quad 0 < x < 1, \quad t > 0, \quad a > 0 \\ b) \quad & u(x, 0) = f(x), \quad 0 < x < 1 \\ c) \quad & u(0, t) = g(t), \quad t > 0. \end{aligned} \quad (12.6)$$

Since $a > 0$ we know that a boundary condition must be prescribed at $x = 0$, and since the highest-order derivative in x is one, then only one boundary condition may be given. In other words, the solution of (12.6) at $x = 1$ is computed and not given. This fact will be used when we discuss PDEs for Asian option pricing in Chapter 24.

If we multiply (12.6)(a) by u , integrate on the interval $(0, 1)$ and use the boundary condition (12.6)(c) we get:

$$\int_0^1 \frac{\partial u}{\partial t} u dx = -a \int_0^1 \frac{\partial u}{\partial t} \cdot u dx = -\frac{a}{2} \int_0^1 \frac{\partial}{\partial x} u^2 dx$$

and

$$\frac{1}{2} \frac{d}{dt} \int_0^1 u^2 dx = \frac{-a}{2}(u^2(1, t) - u^2(0, t)) < \frac{a}{2} u^2(0, t) = \frac{a}{2} g^2(t).$$

Integrating this equation in the time interval $[0, \xi]$ gives:

$$\|u(\cdot, \xi)\|_2^2 \leq \|u(\cdot, 0)\|_2^2 + a \int_0^\xi g^2(t) dt, \quad (12.7)$$

where in general for a function $\varphi(x, t)$ we define $\|\varphi(\cdot, \xi)\|_2^2 = \int_0^1 |\varphi(x, \xi)|^2 dx$.

The formula (12.7) is called an *energy inequality*, and it shows that the solution of (12.6) depends continuously on its data, in this case the initial and boundary conditions. It proves that if a solution is bounded by its data, then it is unique. Thus the problem is well-posed.

An *ill-posed problem* is one that is not well-posed. An example is the *inverse heat equation* in which we deduce a previous distribution of temperature from final data. The solution is highly sensitive to changes in the final data. Ill-posed problems cannot be solved in a computer by stable algorithms. They must be reformulated for numerical treatment, a process called *regularisation*. In general, we regularise a problem by adding some kind of penalty term to it, for example. For some applications, see Tikhonov and Arsenin (1977).

12.4.1 Examples of an ill-posed Problem

Let us take the *Backward Heat Equation* in which we wish to find a solution to the following problem based on a *terminal condition* rather than an initial condition:

$$\begin{aligned} \frac{\partial u(x, t)}{\partial t} &= \frac{\partial^2 u(x, t)}{\partial x^2}, \quad 0 < x < 1, \quad 0 < t < T \\ u(x, T) &= f(x), \quad 0 < x < 1 \\ u(0, t) &= u(1, t) = 0, \quad 0 < t < T. \end{aligned} \quad (12.8)$$

The problem is to find previous states $u(x, t)$ for $t < T$. We claim that this problem has no solution for arbitrary $f(x)$. Let us take an example in which:

$$f(x) = \frac{\sin N\pi x}{N} \quad (N = \pm 1, \pm 2, \dots) \quad (12.9)$$

and the unique solution to (12.8) is given by:

$$u(x, t) = \frac{1}{N} e^{N^2 \pi^2 (T-t)} \sin N\pi x, \quad 0 < x < 1, \quad 0 < t < T. \quad (12.10)$$

For large N , the quantity $|f(x)|$ becomes uniformly small; that is, the initial condition differs by as little as we wish from $f \equiv 0$, and this corresponds to the solution $u \equiv 0$. On the other hand, the solution (12.10) grows with N ; that is, it does not remain close to $u \equiv 0$. We conclude that there is no continuity of dependence on the data.

It is worth noting *Fredholm integral equations* of the first kind are ill-posed (Tikhonov and Arsenin (1977), Tricomi (1985)):

$$\int_a^b K(x,y)u(y)dy = g(x), \quad a < x < b. \quad (12.11)$$

In this case $K(x,y)$ is a known kernel function, $u(y)$ is the unknown function and $g(x)$ is a known function.

Small variations in the function g in (12.11) can lead to instabilities in the solution u . Integral equations occur when we discuss *partial integro-differential equations* (PIDE) for option pricing problems in the presence of jumps. In particular, integral equations of the type:

$$\int_0^\infty K(x,y)u(y)dy = g(x), \quad 0 < x < \infty \quad (12.12)$$

occur in such applications. We can transform (12.12) to an integral equation on the unit interval by the (by now) well-known transformations (as was suggested in Tricomi (1985), p. 151):

$$\begin{aligned} \xi &= \frac{x}{1+x}, & x &= \frac{\xi}{1-\xi} \\ \eta &= \frac{y}{1+y}, & y &= \frac{\eta}{1-\eta} \end{aligned}$$

leading to a new integral equation:

$$\int_0^1 \tilde{K}(\xi, \eta)\tilde{u}(\eta)d\eta = \tilde{g}(\xi), \quad 0 < \xi < 1. \quad (12.13)$$

The kernel of the new integral equation may have some singularities, but this should not lead to major issues. This could be a useful trick for PIDEs.

It might come as a surprise that summation of Fourier series whose coefficients are approximately known in the Euclidean norm is ill-formed. To see this, define the two series:

$$\begin{aligned} f_1(t) &= \sum_{n=0}^{\infty} a_n \cos nt \\ f_2(t) &= \sum_{n=0}^{\infty} b_n \cos nt, \text{ where } b_n = a_n + \varepsilon/n, \quad b_0 = a_0. \end{aligned}$$

The difference in norm between these series is:

$$\varepsilon_1 = \left(\sum_{n=0}^{\infty} (b_n - a_n)^2 \right)^{1/2} = \varepsilon \left(\sum_{n=1}^{\infty} \frac{1}{n^2} \right)^{1/2} = \varepsilon \sqrt{\frac{\pi^2}{6}}$$

and we can make this quantity as small as we wish by an appropriate choice of ε . On the other hand the difference:

$$f_2(t) - f_1(t) = \varepsilon \sum_{n=1}^{\infty} \frac{1}{n} \cos nt$$

can be made as large as we wish because this series is divergent.

As final example, we take the following Cauchy problem for the two-dimensional Laplace equation:

$$\begin{aligned} \Delta u &= \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad -\infty < x < \infty, \quad y > 0 \\ u(x, 0) &= f(x), \quad -\infty < x < \infty \\ \frac{\partial u}{\partial y}(x, 0) &= g(x), \quad -\infty < x < \infty \end{aligned} \tag{12.14}$$

(f(x) and g(x) are given continuous functions).

Consider the two special cases and their solutions:

$$\begin{aligned} f_1(x) &\equiv 0, \quad g(x) = \frac{1}{a} \sin ax, \quad a > 0 \\ u_1(x, y) &= \frac{1}{a^2} \sin ax \sinh ay, \\ \text{and} \\ f_2(x) &= g_2(x) \equiv 0 \\ u_2(x, y) &\equiv 0. \end{aligned} \tag{12.15}$$

Noting that:

$$\begin{aligned} \sup_x |f_1(x) - f_2(x)| &= 0 \\ \sup_x |g_1(x) - g_2(x)| &= \frac{1}{a}. \end{aligned}$$

This latter quantity can be made as small as we wish if we make a large enough. However, the resulting solutions diverge:

$$\sup_x |u_1(x, y) - u_2(x, y)| = \sup_x \left| \frac{1}{a^2} \sin ax \sinh ay \right| = \frac{1}{a^2} \sinh ay \quad \forall y > 0$$

which has no upper limit.

12.4.2 The Importance of Proving that Problems Are Well-Posed

The energy inequality (12.7) and its generalisations are important when we wish to prove that an initial boundary value problem is well-posed: in other words the growth of a solution in some norm is bounded by the associated inhomogeneous terms and

by its boundary and initial conditions, also in some norm. In particular, in many PDEs in finance no boundary conditions are needed or allowed, in particular after having performed domain transformation to map a PDE on an unbounded domain to one on a finite domain. In these cases, the solution will be bounded by the inhomogeneous terms (if present) and by the initial conditions, and hence this proves uniqueness of a solution. For two-factor PDEs we use techniques such as Green's function, two-dimensional integration by parts and the Divergence Theorem to prove well-posedness of problems. In a sense 'all the bad, non-essential data' disappears on the boundaries, leaving only the known data.

The topic of well-posedness has not received much exposure in computational finance. This is possibly because it is not well-known and that ad hoc approaches (such as *domain truncation*) are used without paying attention to existence and uniqueness issues.

12.5 VARIATIONS ON INITIAL BOUNDARY VALUE PROBLEM FOR THE HEAT EQUATION

We give a short overview of the two formulations relating to the heat equations on unbounded and bounded domains. We encounter similar problems when modelling PDEs for finance:

A. Well-Posed Initial Value Problem (Cauchy problem):

$$\begin{aligned} \frac{\partial u(x, t)}{\partial t} &= \Delta u(x, t), \quad x \in \mathbb{R}^n, t > 0 \\ u(x, 0) &= f(x), \quad x \in \mathbb{R}^n \\ |u(x, t)| &< M, \quad x \in \mathbb{R}^n, t > 0. \end{aligned} \tag{12.16}$$

The boundedness condition at infinity is independent of the spatial dimension, but it is too general to be used in practice. We either use domain truncation or domain transformation in combination with numerical boundary conditions.

B. Well-Posed Initial Boundary Value Problem:

$$\begin{aligned} \frac{\partial u}{\partial t}(x, t) &= \Delta u(x, t), \quad x \in \Omega, t > 0 \\ u(x, 0) &= f(x), \quad x \in \Omega \\ \alpha(x)u(x, t) + \beta(x)\frac{\partial u}{\partial x}(x, t) &= g(x, t), \quad x \in \Gamma, t > 0 \quad (\alpha\beta > 0). \end{aligned} \tag{12.17}$$

We are using Robin (mixed) boundary conditions. Special values of α and β lead to Dirichlet or Neumann boundary conditions. For non-bounded domains (for example, a *half-space*) special additional conditions at infinity may be needed.

12.5.1 Smoothness and Compatibility Conditions

In general, the solution of an initial boundary value problem for a parabolic PDE is smooth for all $t > C$ for some given value C (normally it has the value zero) and all x inside the domain. If we wish to have a smooth solution on the boundary and

at $t = C$, it is then necessary to impose extra conditions. To this end, let us take a very useful model example:

$$\begin{aligned} \frac{\partial u(x,t)}{\partial t} &= \frac{\partial^2 u(x,t)}{\partial x^2}, \quad 0 < x < L, \quad 0 < t < T \\ u(x, 0) &= f(x), \quad 0 < x < L \\ u(0, t) &= u(L, t) = 0, \quad 0 < t < T. \end{aligned} \quad (12.18)$$

We wish to extend the solution to $0 \leq x \leq L, t \geq 0$, in which case we must have *compatibility conditions*:

$$f(0) = g(0), \quad f(L) = h(0) \quad (12.19)$$

at the ‘corners’ $(0, 0)$ and $(L, 0)$. If these conditions are not satisfied, then the solution may satisfy the initial and boundary conditions in a mean-square sense only but not in a pointwise sense. In these cases there is a discontinuity in the solution, and finite difference approximations will be poor. However, the approximation at $x = 0$ improves as t increases. The same remark holds for discontinuous initial conditions; the discontinuities in the solution are smoothed as t increases.

As a fix for *incompatibility* at $x = 0, t = 0$, we could take the average value at the corner $x = 0, t = 0$ (Smith (1978)):

$$u(0, 0) = \left(\lim_{t \rightarrow 0} u(0, t) + \lim_{x \rightarrow 0} u(x, 0) \right) / 2 = \left(\lim_{t \rightarrow 0} g(t) + \lim_{x \rightarrow 0} f(x) \right) / 2. \quad (12.20)$$

An important follow-on consequence of this little piece of analysis is to choose the boundary conditions so that the compatibility conditions (12.20) hold. Thus, given the initial condition we can then find the boundary conditions for a given problem. In the case of finance, one choice for numerical boundary conditions will be found from the values of the payoff function on the boundary of the domain. A good application of this principle can be found in Chapter 23, where we discuss multi-asset option pricing (spread options).

12.6 MAXIMUM-MINIMUM PRINCIPLES FOR PARABOLIC PDEs

We now discuss the maximum-minimum principles that we introduced in Chapter 10 but now we do it for the heat equation.

Let Ω be a bounded region in \mathbb{R}^3 whose boundary Γ is a smooth closed surface. Let $u(x, y, z, t)$ be continuous for $(x, y, z) \in \bar{\Omega}$ and $0 \leq t \leq T$ and let:

$$\begin{aligned} M_\Gamma &= \max \{u(x, y, z, t) : (x, y, z) \in \Gamma, 0 \leq t \leq T\} \\ M_0 &= \max \{u(x, y, z, t) : (x, y, z) \in \bar{\Omega}, t = 0\} \\ M &= \max \{M_\Gamma, M_0\}. \end{aligned}$$

In the same vein, let m_Γ, m_0 and m denote the corresponding minimum values for u .

Theorem 12.1 (Maximum-minimum principles for the heat equation)

If $u(x, y, z, t)$ is continuous in $\bar{\Omega} \times (0, T)$, then with $Q = \Omega \times (0, T)$ we have:

- i) If $\frac{\partial u}{\partial t} - \Delta u \leq 0$ in Q then $u \leq M$ in \bar{Q} .
- ii) If $\frac{\partial u}{\partial t} - \Delta u \geq 0$ in Q then $u \geq m$ in \bar{Q} .
- iii) If $\frac{\partial u}{\partial t} - \Delta u = 0$ in Q then $m \leq u \leq M$ in \bar{Q} where $\bar{Q} = \bar{\Omega} \times [0, T]$.

12.7 PARABOLIC EQUATIONS WITH TIME-DEPENDENT BOUNDARIES

We discuss an initial boundary value problem for one-dimensional convection-diffusion-reaction equations with time-dependent boundaries. Such problems occur in many kinds of applications, for example barrier options with time-dependent barriers (Tavella and Randall (2000)) and PDE models for option pricing with early exercise features. In the former case the boundaries are time-dependent and known, while in the latter case they are time-dependent and unknown. Some examples of time-dependent barriers are found in Haug (2007) and Ikeda and Kunimoto (1992):

- Flat boundaries (no time dependence).
- An exponentially growing lower boundary and an exponentially decaying upper boundary.
- A convex downward lower boundary and a convex upward upper boundary.

In this section we examine a second-order parabolic partial differential equation in one space dimension. In other words, this is a specialisation of the equations in previous sections for the case $n = 1$. To this end, we examine the equation:

$$-\frac{\partial u}{\partial t} + \sigma(x, t) \frac{\partial^2 u}{\partial x^2} + \mu(x, t) \frac{\partial u}{\partial x} + b(x, t)u = f(x, t) \quad (12.21)$$

in the domain defined by:

$$D = \{(x, t) : 0 \leq t \leq T, s_1(t) < x < s_2(t)\}$$

subject to the conditions:

$$\begin{aligned} s_1(0) &= 0; \quad s_2(0) = 1 \\ s_1(t) &< s_2(t), \quad 0 \leq t \leq T. \end{aligned}$$

Furthermore, we augment PDE (12.21) with boundary conditions (see Figure 12.1). Here, ψ_1, ψ_2 and φ are given functions.

$$\left. \begin{aligned} u(s_1(t), t) &= \psi_1(t) \\ u(s_2(t), t) &= \psi_2(t) \end{aligned} \right\} \quad 0 \leq t \leq T$$

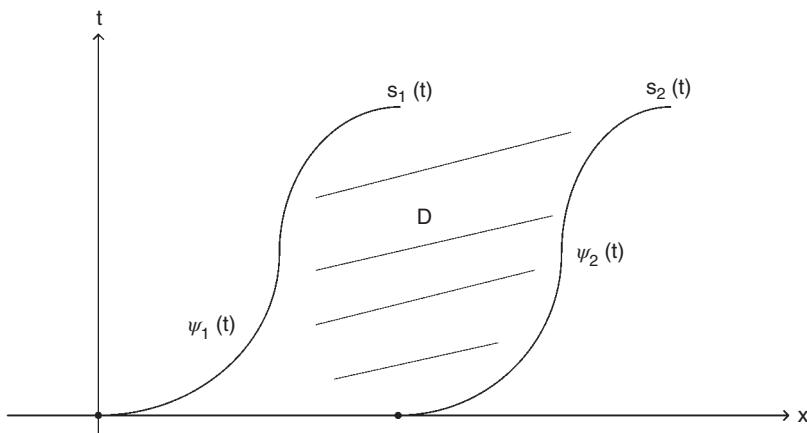


FIGURE 12.1 Region of integration D .

and initial conditions:

$$u(x, 0) = \varphi(x), \quad s_1(t) < x < s_2(t).$$

Finally, we assume the *compatibility conditions*:

$$\begin{aligned}\psi_1(0) &= \varphi(0) \\ \psi_2(0) &= \varphi(1).\end{aligned}$$

The domain D is somewhat irregular because it is a function of time. We can map D onto the unit square by a change of variables:

$$z = \frac{x - s_1(t)}{s_2(t) - s_1(t)}, \quad \tau = t, \quad s_1(t) \neq s_2(t). \quad (12.22)$$

In this case, the point (x, t) is mapped to the point (z, τ) . We then get a convection-diffusion PDE in (z, τ) (see Bobisud (1967)).

The above problem is a model for many one-factor Black–Scholes equations. For example, standard European options can be formulated in this way having constant boundaries while barrier options fit into this model as well because in practice the boundaries are time dependent. For example, a down-and-out/up-and-out call barrier option is described by the following model (Tavella and Randall (2000), p. 183):

$$\begin{aligned}\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r - D_0)S \frac{\partial V}{\partial S} - rV &= 0 \\ V(S \leq L(t), t) &= 0 \\ V(S \geq U(t), t) &= 0 \\ V(S, T) &= \max(S - K, 0).\end{aligned} \quad (12.23)$$

In general, the barrier functions $L(t)$ and $U(t)$ can be analytic functions, but they could also be the solution of ordinary differential equations or even the solution of other PDEs. (For example, $U(t)$ could be a forward swap rate.)

We note that the transformation $(x, t) \rightarrow (z, \tau)$ in Equation (12.22) leads to a PDE in two new variables, and to achieve this end we need to use the chain rule for differentiation *in two variables* as discussed in Chapter 1. Nothing else will work, and many students and practitioners miss the point when encountering this problem for the first time.

We can now approximate the PDE with fixed boundaries in the new coordinate system (z, τ) using your favourite finite difference method. Then the option values in the original variables by the inverse transformation of (12.22).

12.8 UNIQUENESS THEOREMS FOR BOUNDARY VALUE PROBLEMS IN TWO DIMENSIONS

When studying initial boundary value problems for partial differential equations we need to answer various questions:

- How many solutions are there (none, one, a finite number of solutions, and an infinite number of solutions)?
- Prove that a solution exists and determine its smoothness (for example, is it continuously differentiable?).

We now take two examples.

12.8.1 Laplace Equation

To prove uniqueness, we need to prove that two functions that satisfy Laplace's equation and that satisfy the same boundary conditions are in fact equal. Since this is a linear problem, this is equivalent to proving that zero data produces a zero solution for the following problem:

$$\begin{aligned} \Delta u &= 0 \text{ in } \Omega \\ u &= 0 \text{ on } \Gamma. \end{aligned} \tag{12.24}$$

Recalling Green's first identity from Chapter 11 (Section 11.7.3), namely:

$$\int_{\Omega} u \Delta v dx = \int_{\Gamma} u \frac{\partial v}{\partial n} ds - \int_{\Omega} \nabla u \cdot \nabla v dx$$

and we set $u = v$ to give and also noting that $u = 0$ on Γ :

$$\begin{aligned} \int_{\Omega} \nabla u \cdot \nabla u dx &= 0 \\ \text{or} \\ \int_{\Omega} \left[\left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2 \right] dx &\equiv \int_{\Omega} \|\nabla u\|^2 dx = 0 \end{aligned} \tag{12.25}$$

Then:

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial y} = 0 \Rightarrow u = \text{constant in } \Omega$$

since $u|_{\Gamma} = 0$ we conclude $u = 0$ in $\overline{\Omega}$.

12.8.2 Heat Equation

In the spirit of the previous subsection we examine:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \Delta u \text{ in } \Omega \times (0, T) \text{ (a)} \\ u &= 0 \text{ on } \Gamma \\ u(x, 0) &= 0, \quad x \in \Omega. \end{aligned} \tag{12.26}$$

Multiplying both sides of equation (12.26)(a) by u and using Green's first identity as in Section 12.8.1 we get the ODE:

$$\frac{d}{dt} \int_{\Omega} u^2 dx = - \int_{\Omega} \|\nabla u\|^2 dx \leq 0. \tag{12.27}$$

Integrating (12.27) gives:

$$\int_{\Omega} u^2(x, t) dx \leq \int_{\Omega} u^2(x, 0) dx = 0. \tag{12.28}$$

Hence $u \equiv 0$ in $\Omega \times (0, T)$.

12.9 SUMMARY AND CONCLUSIONS

In this chapter we introduced initial boundary value problems for parabolic partial differential equations and for the two-dimensional heat equation. We discussed several techniques to give us more insights into understanding the use cases associated with these kinds of equations and that will allow us to progress to more complex time-dependent PDEs and their numerical solutions, in particular Black–Scholes-style equations and their numerical approximation.

CHAPTER 13

Stochastics Representations of PDEs and Applications

Young man, in mathematics you don't understand things. You just get used to them.

John von Neumann

13.1 INTRODUCTION AND OBJECTIVES

The first twelve chapters of this book focused on ordinary and partial differential equations, in particular on formulating initial boundary value problems, producing analytical solutions and introducing fundamentals of the finite difference method. In this chapter we discuss the close connection between statistics, stochastic processes and Monte Carlo simulation on the one hand and PDE theory on the other hand. In a sense there a one-to-one correspondence between the solution of a PDE and its stochastic representation (a standard reference is Friedman (1976)). Furthermore, each representation can be approximated by numerical methods, for example by Monte Carlo simulation of stochastic differential equations and by the finite difference method. The advantages are clear; we can choose the representation that is most appropriate in a given context. Furthermore, we discuss the relationship between probability theory and partial differential equation theory in the form of the *Kolmogorov* and *Fokker–Planck equations* which are used when calibrating volatility models.

The following topics are discussed in this chapter:

- An introduction to stochastic differential equations (SDEs) and their numerical approximation.
- Some popular finite difference schemes for SDEs and applications to the Heston model and two-factor spread options.
- Option pricing using Monte Carlo simulation and C++ code.

- Fundamental results and theorems; *Feynman–Kac formula*, *Kormogorov backward and forward (Fokker–Planck) equations* and their applications.
- Initial boundary value problems for the Fokker–Planck equation.
- First-exit time problems.

More could be written, but a detailed discussion is outside the scope of this book.

13.2 BACKGROUND, REQUIREMENTS AND PROBLEM STATEMENT

Some of the new topics in this chapter had not yet been discussed in this book, in particular the area of stochastic differential equations (SDEs) and their numerical approximation. We refer the interested reader to Kloeden and Platen (1995), Kloeden, Platen and Schurz (1997), Duffy and Kienitz (2009) and Øksendal (1998) for further discussions. From a PDE perspective, the results in this chapter can be seen as a continuation of the other chapters in this book and familiar and standard notation is used.

13.3 AN OVERVIEW OF STOCHASTIC DIFFERENTIAL EQUATIONS (SDEs)

We include a short discussion of SDEs for completeness. A *random process* is a family of random variables defined on some probability space and indexed by the parameter t where t belongs to some *index set*. A random process is in fact a function of two variables:

$$\{\xi(t, x) : t \in T, x \in S\}$$

where T is the index set and S is the *sample space*. For a fixed value of t , the random process becomes a random variable, while for a fixed sample point x in S the random process is a real-valued function of t called a *sample function* or a *realisation* of the process. It is also sometimes called a *path*. The totality of all sample paths is called an *ensemble*.

The index set T is called the *parameter set*, and the values assumed by $\xi(t, \omega)$ are called the *states*; finally, the set of all possible states is called the *state space* of the random process.

The index set T can be discrete or continuous; if T is discrete then the process is called a *discrete-parameter* or *discrete-time process* (also known as a *random sequence*). If T is continuous then we say that the random process is called *continuous-parameter* or *continuous-time*.

We can also consider the situation where the state is discrete or continuous. We then say that the random process is called *discrete-state (chain)* or *continuous-state*, respectively.

13.4 AN INTRODUCTION TO ONE-DIMENSIONAL RANDOM PROCESSES

In this section we introduce random processes described by SDEs of the form:

$$d\xi(t) = \mu(t, \xi(t)) dt + \sigma(t, \xi(t)) dW(t) \quad (13.1)$$

where:

- $\xi(t) \equiv$ random process
- $\mu(t, \xi(t)) \equiv$ transition (drift) coefficient
- $\sigma(t, \xi(t)) \equiv$ diffusion coefficient
- $W(t) \equiv$ Brownian process
- $\xi(0) =$ given initial condition

defined in the interval $[0, T]$. We assume for the moment that the process takes values on the real line. We know that this SDE can be written in the equivalent integral form:

$$\xi(t) = \xi(0) + \int_0^t \mu(s, \xi(s)) ds + \int_0^t \sigma(s, \xi(s)) dW(s). \quad (13.2)$$

This is a non-linear equation because the unknown random process appears on both sides of the equation and it cannot be expressed in a closed form in general.

We know that the second integral:

$$\int_0^t \sigma(s, \xi(s)) dW(s)$$

is a continuous process (with probability 1) provided $\sigma(s, \xi(s))$ is a bounded process. In particular, we restrict the scope to those functions for which:

$$\sup_{|x| \leq C} (|\mu(s, x)| + |\sigma(s, x)|) < \infty, \quad t \in (0, T] \text{ and for every } C > 0.$$

Using this fact, we shall see that the solution of Equation (13.2) is bounded and continuous with probability 1.

We now discuss existence and uniqueness theorems. First, we define some conditions on the coefficients in Equation (13.2):

- C1: $\exists K > 0$ such that $\mu(s, x)^2 + \sigma(s, x)^2 \leq K(1 + x^2)$.
- C2: $\forall c > 0, \exists L$ such that $|\mu(s, x) - \mu(s, y)| + |\sigma(s, x) - \sigma(s, y)| \leq L|x - y|$ for $|x| \leq C$, $|y| \leq C$.
- C3: $\mu(s, x)$ and $\sigma(s, x)$ are defined and measurable with respect to their variables where $s \in (0, T], x \in (-\infty, \infty)$.
- C4: $\mu(s, x)$ and $\sigma(s, x)$ are continuous with respect to their variables for $s \in (0, T]$, $x \in (-\infty, \infty)$.

Condition C2 is called a *Lipschitz condition* in the second variable, while condition C1 constrains the growth of the coefficients in Equation (13.2). We assume throughout that the random variable $\xi(0)$ is independent of $W(t)$.

Theorem 13.1 Assume conditions C1, C2 and C3 hold. Then Equation (13.2) has a unique continuous solution with probability 1 for any initial condition $\xi(0)$.

Theorem 13.2 Assume that conditions C1 and C4 hold. Then Equation (13.2) has a continuous solution with probability 1 for any initial condition $\xi(0)$.

We note the difference between the two theorems: the condition C2 is what makes the solution unique.

We now define another condition on the diffusion coefficient in Equation (13.2).

- C5 $\sigma(t, x) > 0$ and for every $C > 0$, there exists an $L > 0$ and $\alpha > \frac{1}{2}$ such that:
 $|\sigma(t, x) - \sigma(t, y)| \leq L|x - y|^\alpha$ for $|x| \leq C, |y| \leq C$.

Theorem 13.3 Assume conditions C4, C1 and C5 hold. Then Equation (13.2) has a continuous solution with probability 1 for any initial condition $x(0)$.

For proofs of these theorems, see Skorokhod (1982), for example.

In some cases it is possible to find a closed-form solution of Equation (13.1) (or equivalently, Equation (13.2)). When the drift and diffusion coefficients are constant we see that the exact solution is given by the formula:

$$\xi(t) = \xi(0) \exp((\mu - \frac{1}{2}\sigma^2)t + \sigma W(t)). \quad (13.3)$$

Knowing the exact solution is useful because we can test the accuracy of finite difference schemes against it, and this gives us some insights into how well these schemes work for a range of parameters.

It is useful to know how the solution of Equation (13.3) behaves for large values of time; the answer depends on the relationship between the drift and diffusion parameters:

- i) $\mu > \frac{1}{2}\sigma^2$, $\lim_{t \rightarrow \infty} \xi(t) = \infty$ almost surely
- ii) $\mu < \frac{1}{2}\sigma^2$, $\lim_{t \rightarrow \infty} \xi(t) = 0$ almost surely
- iii) $\mu = \frac{1}{2}\sigma^2$, $\xi(t)$ fluctuates between arbitrary large and arbitrary small values as $t \rightarrow \infty$ almost surely. (13.4)

In general it is not possible to find an exact solution, and in these cases we resort to numerical approximation techniques.

Equation (13.2) is a *one-factor equation* because there is only one dependent variable to be modelled, namely $\xi(t)$. It is possible to define equations with several dependent variables. The prototypical non-linear stochastic differential equation in that case is given by the system:

$$dX = \mu dt + \sigma dW(t) \quad (13.5)$$

where

$$\begin{aligned} \mu &: [0, T] \times \mathbb{R}^n \rightarrow \mathbb{R}^n \text{ (vector)} \\ \sigma &: [0, T] \times \mathbb{R}^n \rightarrow \mathbb{R}^{n \times m} \text{ (matrix)} \\ X &: [0, T] \rightarrow \mathbb{R}^n \text{ (vector)} \\ W &: [0, T] \rightarrow \mathbb{R}^m \text{ (vector).} \end{aligned}$$

In general the drift and diffusion are non-linear:

$$\begin{aligned}\mu &= \mu(t, X) \\ \sigma &= \sigma(t, X).\end{aligned}$$

This is a generalisation of Equation (13.2). Thus, instead of scalars this system employs vectors.

Existence and uniqueness theorems for the solution of the SDE system (13.5) are similar to those in the one-factor case. For example, theorem 5.2.1 in Øksendal (1998) addresses these issues.

13.5 AN INTRODUCTION TO THE NUMERICAL APPROXIMATION OF SDES

We give a short overview of some finite difference schemes for one-factor SDEs. They can be generalised to two-factor and n -factor SDEs (see Duffy and Kienitz (2009), where robust schemes in C++ are also discussed with applications to option pricing).

13.5.1 Euler–Maruyama Method

In this section we introduce the finite difference method (FDM), and we apply it to finding approximate solutions of SDEs.

The first step is to replace continuous time by discrete time. To this end, we divide the interval $[0, T]$ (where T is the expiration date) into a number of subintervals as shown in Figure 13.1. We define $N + 1$ *mesh points* as follows:

$$0 = t_0 < t_1 < \dots < t_n < t_{n+1} < \dots < t_N = T.$$

In this case we define a set of *subintervals* (t_n, t_{n+1}) of size $\Delta t_n \equiv t_{n+1} - t_n$, $0 \leq n \leq N - 1$. In general, we speak of a *non-uniform mesh* when the sizes of the subintervals are not necessarily the same. However, we consider a large class of finite difference schemes where the N subintervals have the same size (we then speak of a *uniform mesh*), namely $\Delta t = T/N$.

Having defined how to subdivide $[0, T]$ into subintervals, we are now ready to motivate the finite difference schemes; for the moment we examine the scalar linear SDE with constant coefficients:

$$\begin{aligned}dX &= aXdt + bXdW, \quad a, b \text{ constant} \\ X(0) &= A.\end{aligned}\tag{13.6}$$

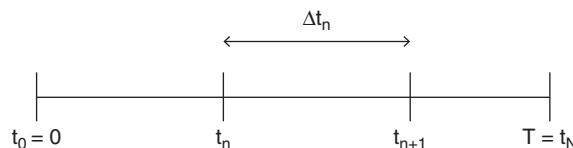


FIGURE 13.1 Mesh generation.

Regarding notation, we do not show the dependence of variable X on t , and when we wish to show this dependence, we prefer to write $X = X(t)$ instead of the form X_t .

We write Equation (13.6) as a *stochastic integral equation* between the (arbitrary) times s and t as follows:

$$X(t) = X(s) + \int_s^t aX(y)dy + \int_s^t bX(y)dW(y), \quad s < t. \quad (13.7)$$

We evaluate Equation (13.7) at two consecutive mesh points, and using the fact that the factors a and b in Equation (13.6) are constant, we get the exact identity:

$$\begin{aligned} X(t_{n+1}) &= X(t_n) + \int_{t_n}^{t_{n+1}} aX(y)dy + \int_{t_n}^{t_{n+1}} bX(y)dW(y) \\ &= X(t_n) + a \int_{t_n}^{t_{n+1}} X(y)dy + b \int_{t_n}^{t_{n+1}} X(y)dW(y). \end{aligned} \quad (13.8)$$

We now approximate Equation (13.8), and in this case we replace the solution X of Equation (13.8) by a new discrete function Y (that is, one that is defined at mesh points) by assuming that it is constant on each subinterval; we then arrive at the discrete equation:

$$Y_{n+1} = Y_n + aY_n \int_{t_n}^{t_{n+1}} dy + bY_n \int_{t_n}^{t_{n+1}} dW(y) = Y_n + aY_n \Delta t_n + bY_n \Delta W_n, \quad (13.9)$$

where:

$$\Delta W_n = W(t_{n+1}) - W(t_n), \quad 0 \leq n \leq N - 1.$$

This is called the (explicit) *Euler–Maruyama scheme*, and it is a popular method when approximating the solution of SDEs. In some cases we write the solution in terms of a discrete function X_n if there is no confusion between it and the solution of Equations (13.6) or (13.7). In other words, we can write the discrete Equation (13.9) in the equivalent form:

$$\begin{cases} X_{n+1} = X_n + aX_n \Delta t_n + bX_n \Delta W_n \\ X_0 = A. \end{cases} \quad (13.10)$$

In many examples the mesh size is constant, and furthermore the Wiener increments are well-known computable quantities:

$$\begin{cases} \Delta t_n = \Delta t = T/N, \quad 0 \leq n \leq N - 1 \\ \Delta W_n = \sqrt{\Delta t} z_n, \text{ where } z_n \sim N(0, 1). \end{cases}$$

Incidentally, we generate increments of the Wiener process by a random number generator for independent Gaussian pseudo-random numbers, for example Mersenne Twister or lagged Fibonacci generator methods. C++ support these and other generators.

13.5.2 Milstein Method

The Milstein scheme is a variation of the Euler–Maruyama scheme. In this case we add an extra term that depends on the derivative of the diffusion term with respect to the (unknown) dependent variable. Let us consider the general one-factor non-linear SDE:

$$dX = \mu(t, X)dt + \sigma(t, X)dW$$

In order to use the Milstein scheme we define the term:

$$\frac{1}{2}\sigma\sigma' \{ (\Delta W_n)^2 - \Delta t \} \text{ where } \sigma' = \frac{\partial\sigma}{\partial X}. \quad (13.11)$$

In general, a derivative needs to be calculated. In the special constant coefficient case of Equation (13.6) this quantity is just the constant volatility term. There are two forms of the Milstein scheme; first, the *Ito–Milstein* scheme is:

$$X_{n+1} = X_n + \mu X_n \Delta t + \sigma X_n \Delta W_n + \frac{1}{2}\sigma\sigma' \{ (\Delta W_n)^2 - \Delta t \} \quad (13.12)$$

while the *Stratonovich–Milstein* scheme is:

$$X_{n+1} = X_n + \tilde{\mu} X_n \Delta t + \sigma X_n \Delta W_n + \frac{1}{2}\sigma\sigma' (\Delta W_n)^2 \quad (13.13)$$

where:

$$\tilde{\mu} = \mu - \frac{1}{2}\sigma\sigma'.$$

It can be shown that the Milstein method is strongly consistent of strong order 1.0.

It is possible to define a generalised Milstein method by approximating the drift term by a combination of explicit and implicit terms. This is called the *implicit Milstein scheme*:

$$X_{n+1} = X_n + \mu \{ \alpha X_n + (1 - \alpha) X_{n+1} \} \Delta t + \sigma X_n \Delta W_n + \frac{1}{2}\sigma\sigma' \{ (\Delta W_n)^2 - \Delta t \} \quad (13.14)$$

where $0 \leq \alpha \leq 1$.

13.5.3 Predictor-Corrector Method

Predictor-corrector schemes are suitable for non-linear equations because they allow us to linearise these non-linear equations. It is a generalisation of similar methods for ODEs. This means that we can solve the problem without having to use non-linear solvers such as the Newton–Raphson method, for example. We consider system (13.5) again with $\mu \equiv a$ and $\sigma \equiv b$. The idea is simple; first, we define a *predictor* value that is the solution of an explicit scheme at level n , for example Euler–Maruyama:

$$\tilde{X}_{n+1} = X_n + \mu X_n \Delta t + \sigma X_n \Delta \hat{W}_n \quad (13.15)$$

where $\Delta \widehat{W}_n$ is a *two point distributed random variable* with $P(\Delta \widehat{W}_n = \pm \sqrt{\Delta t}) = 1/2$, $n = 0, \dots, N - 1$.

We then use this predictor to calculate the solution (called the *corrector*) at time level $n + 1$, for example using the *modified trapezoidal method*:

$$X_{n+1} = X_n + \mu \{ \alpha X_n + (1 - \alpha) \tilde{X}_{n+1} \} \Delta t + \sigma X_n \Delta W_n \quad (13.16)$$

where $0 \leq \alpha \leq 1$.

It is possible to define other kinds of predictor-corrector schemes for non-linear SDEs; see Kloeden, Platen and Schurz (1997) for more details.

13.5.4 Drift-Adjusted Predictor-Corrector Method

We discuss a more general robust scheme that is being used for one-factor and multi-factor SDEs in finance. It is called the predictor-corrector method and is a generalisation of a similar scheme that is used to solve ordinary differential equations (Lambert (1991)). In order to apply it to SDEs, we examine the scalar non-linear problem given in Section 13.4. If we discretise this SDE using the trapezoidal rule, for example, we will get a non-linear system of equations at each time level that we need to solve using Newton's or Steffensen's method:

$$\begin{aligned} X_{n+1} &= X_n + \{ \alpha \mu(X_{n+1}) + (1 - \alpha) \mu(X_n) \} \Delta t \\ &\quad + \{ \beta \sigma(X_{n+1}) + (1 - \beta) \sigma(X_n) \} \Delta W_n, \quad n \geq 0 \end{aligned} \quad (13.17)$$

where $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$.

Instead of solving this equation, we attempt to linearise it by replacing the offending unknown solution at time level $n + 1$ by another known quantity (called the *predictor*) that is hopefully close to it. To this end, we compute the predictor as:

$$\tilde{X}_{n+1} = X_n + \mu(X_n) \Delta t + \sigma(X_n) \Delta W_n, \quad n \geq 0 \quad (13.18)$$

that we subsequently use in the corrector equation:

$$\begin{aligned} X_{n+1} &= X_n + \{ \alpha \mu(\tilde{X}_{n+1}) + (1 - \alpha) \mu(X_n) \} \Delta t \\ &\quad + \{ \beta \sigma(\tilde{X}_{n+1}) + (1 - \beta) \sigma(X_n) \} \Delta W_n, \quad n \geq 0. \end{aligned} \quad (13.19)$$

However, we modify Equation (13.19) by using the so-called *corrector drift function*:

$$\bar{\mu}_\beta(x) = \mu(x) - \beta \sigma(x) \frac{\partial \sigma}{\partial x}(x). \quad (13.20)$$

The new corrector equation is given by:

$$\begin{aligned} X_{n+1} &= X_n + \{ \alpha \bar{\mu}_\beta(\tilde{X}_{n+1}) + (1 - \alpha) \bar{\mu}_\beta(X_n) \} \Delta t \\ &\quad + \{ \beta \sigma(\tilde{X}_{n+1}) + (1 - \beta) \sigma(X_n) \} \Delta W_n, \quad n \geq 0. \end{aligned} \quad (13.21)$$

The solution of this scheme can be solved without the need to use a non-linear solver. Furthermore, we can customise the scheme to support different levels of implicitness and explicitness in the drift and diffusion terms:

- A. Fully explicit ($\alpha = \beta = 0$)
- B. Fully implicit ($\alpha = \beta = 1$)
- C. Implicit in drift explicit in diffusion ($\alpha = 1, \beta = 0$)
- D. Symmetric ($\alpha = \beta = 1/2$).

We need to determine which combination of parameters results in stable schemes. We have seen with the schemes in this chapter that reducing the time steps (which is the same as increasing the number of subdivisions of the interval $[0, T]$) does not necessarily increase accuracy and may actually lead to serious inaccuracies. This phenomenon is not common when discretising deterministic equations and can come as a shock if you are not prepared. We have experimented with a number of test cases, and the general conclusion is that the most stable schemes are those that have the same level of implicitness in drift and diffusion, for example the options A, B and D in Equation (13.19). In particular, the symmetric case D is a good all-rounder in the sense that it gives good results for a range of SDEs.

13.6 PATH EVOLUTION AND MONTE CARLO OPTION PRICING

We now turn our attention to the n -factor case, and we examine the system of equations representing a multidimensional geometric Brownian motion:

$$dS_j(t) = \mu_j S_j(t)dt + \sigma_j S_j(t)dW_j(t), \quad j = 1, \dots, M \quad (13.22)$$

where W_j is a standard one-dimensional Brownian motion; furthermore, W_i and W_j have correlation ρ_{ij} .

The individual solutions have the representation:

$$S_j(t) = S_j(0) \exp \left(\left(\mu_j - \frac{\sigma_j^2}{2} \right) t + \sigma_j W_j(t) \right) \quad j = 1, \dots, M. \quad (13.23)$$

Now, define the $M \times M$ covariance matrix $\Sigma = (\Sigma_{ij})$ as follows (Glasserman (2004), p. 104):

$$\begin{aligned} \Sigma_{ij} &= \sigma_i \sigma_j \rho_{ij} \text{ for } i \neq j, \quad i, j = 1, \dots, M \\ \Sigma_{ii} &= \sigma_i^2, \quad i = 1, \dots, M \end{aligned}$$

where ρ_{ij} is the correlation between assets i and j .

We know that the covariance matrix can be written in the form $\Sigma = AA^\top$ (because it is symmetric and positive definite) where A is a matrix and A^\top is the transpose of A . From this, we can compute the values at given mesh points as follows:

$$S_j(t_{n+1}) = S_j(t_n) \exp \left(\left(\mu_j - \frac{\sigma_j^2}{2} \right) \Delta t_n + \sqrt{\Delta t_n} \sum_{i=1}^M A_{ji} Z_{n+1,i} \right)$$

$$j = 1, \dots, M, \quad n = 0, \dots, N-1 \quad (13.24)$$

where:

$$\Delta t_n = t_{n+1} - t_n$$

and:

$Z_n = (Z_{n,1}, \dots, Z_{n,M}) \sim N(0, I)$ where $Z_{n,1}, \dots, Z_{n,M}$ are independent and I is the $M \times M$ identity matrix.

13.6.1 Monte Carlo Option Pricing

We describe the steps to price a one-factor option using the Monte Carlo method.

We recall the SDE that describes the behaviour of a dividend-paying stock. The SDE is given by:

$$dS_t = (r - D)S_t dt + \sigma S_t dW_t \quad (13.25)$$

where:

- S_t = underlying stock.
- r = (constant) interest rate.
- D = constant dividend.
- σ = constant volatility.
- dW_t = increments of the Wiener process.

For the current problem it is possible to transform the SDE to a simpler one. To this end, define the variable:

$$x_t = \log(S_t).$$

Then the modified SDE is described as:

$$dx_t = vdt + \sigma dW_t, \quad v \equiv r - D - \frac{1}{2}\sigma^2.$$

We now discretise this equation by replacing the continuously defined quantities dx_t , dt and dW_t by their discrete analogues. This gives the following *discrete stochastic equation*:

$$\Delta x_t = v\Delta t + \sigma\Delta W_t \quad (13.26)$$

or

$$x_{t+\Delta t} = x_t + v\Delta t + \sigma(W_{t+\Delta t} - W_t).$$

Since $x_t = \log(S_t)$ we can see that Equation (13.26) is equivalent to a discrete equation in the stock price S_t :

$$S_{t+\Delta t} = S_t \exp(v\Delta t + \sigma(W_{t+\Delta t} - W_t)). \quad (13.27)$$

Formulae (13.26) and (13.27) constitutes the basic *path-generation algorithm*: we calculate the value in Equation (13.27) at a set of discrete *mesh points*:

$$0 = t_0 < t_1 < \dots < t_{N-1} < t_N = T$$

$$t_n = n\Delta t, \quad n = 0, \dots, N, \quad \Delta t = N/T.$$

Then Equation (13.27) takes the more computationally attractive form:

$$x_{t_n} = x_{t_{n-1}} + v\Delta t + \sigma\sqrt{\Delta t}\epsilon_n, \quad n = 1, \dots, N \text{ where} \quad (13.28)$$

ϵ_n is a sample from $N(0, 1)$ and $S_{t_n} = \exp(x_{t_n})$.

The next step is to run the algorithm in Equation (13.28) M times (M is called the number of *simulations* or *draws*); for each price of the stock at $t = T$ we calculate the pay-off function for a call option, namely:

$$\text{payoff} = \max(0, S_T - K).$$

This formula gives the value of the pay-off, and we are done, almost. Finally, we calculate the average call price over all call prices (evaluated at $t = T$), and we take the discounted average of these *simulated paths*. Summarising, the basic algorithm for a call option is given by:

For each $j = 1, \dots, M$ calculate

$$C_{T,j} = \max(0, S_{T,j} - K)$$

and:

$$\hat{C} = \exp(-rT) \frac{1}{M} \sum_{j=1}^M \max(0, S_{T,j} - K)$$

Then \hat{C} is the desired call price.

13.6.2 Some C++ Code

We present the full C++ (from version 11) code that approximates a general one-factor SDE by the Euler–Maruyama method. An interesting exercise is to modify the code to support the drift-adjusted predictor-corrector method (13.21). The code computes call option price by the Monte Carlo method:

```
// Encapsulate all data in one place
struct OptionData
{ // Option data + behaviour

    double K;
    double T;
    double r;
    double sig;

    // Extra data
    double D;           // dividend

    int type;          // 1 == call, -1 == put

    explicit constexpr OptionData(double strike, double expiration,
        double interestRate, double volatility, double dividend, int PC)
        : K(strike), T(expiration), r(interestRate), sig(volatility),
        D(dividend), type(PC)
    {}

    double myPayOffFunction(double S)
    { // Payoff function

        if (type == 1)
        { // Call

            return std::max(S - K, 0.0);
        }
        else
        { // Put

            return std::max (K - S, 0.0);
        }
    }
};

// (C) Datasim Education BV 2008-2018
//

#include "OptionData.hpp" // in local directory
#include <random>
#include <memory>
#include <cmath>
#include <iostream>
#include "StopWatch.cpp"
```

```
class SDE
{ // Defines drift + diffusion + data
private:
    std::shared_ptr<OptionData> data;      // The data for the option
public:
    SDE(const OptionData& optionData) : data(new OptionData(optionData)) {}

    double drift(double t, double S)
    { // Drift term

        return (data->r - data->D)*S; // r - D
    }

    double diffusion(double t, double S)
    { // Diffusion term

        return data->sig * S;
    }

};

int main()
{
    std::cout << "1 factor MC with explicit Euler\n";

    // Init via arg positon

    OptionData myOption{ 65.0, 0.25, 0.08, 0.3, 0.0, -1 };

    SDE sde(myOption);

    // Initial value of SDE
    double S_0 = 60;

    // Variables for underlying stock
    double x;
    double VOld = S_0;
    double VNew;

    long NT = 100;
    std::cout << "Number of time steps: ";
    std::cin >> NT;

    // V2 mediator stuff
    long NSIM = 50000;
    std::cout << "Number of simulations: ";
    std::cin >> NSIM;
    double M = static_cast<double>(NSIM);
```

```
double dt = myOption.T / static_cast<double> (NT);
double sqrt = std::sqrt(dt);

// Normal random number
double dW;
double price = 0.0;      // Option price
double payoffT;
double avgPayoffT = 0.0;
double squaredPayoff = 0.0;
double sumPriceT = 0.0;

// #include <random>
// Normal (0,1) rng
std::default_random_engine dre;
std::normal_distribution<double> nor(0.0, 1.0);

// Create a random number
dW = nor(dre);

StopWatch<> sw;
sw.Start();

for (long i = 1; i <= M; ++i)
{ // Calculate a path at each iteration

    if ((i/100'000) * 100'000 == i)
    { // Give status after each 10000th iteration

        std::cout << i << ", ";
    }

    VOld = S_0;
    x = 0.0;
    for (long index = 0; index <= NT; ++index)
    {

        // Create a random number
        dW = nor(dre);

        // The FDM (in this case explicit Euler)
        VNew = VOld + (dt * sde.drift(x, VOld)) + (sqrt * sde.diffusion(x, VOld) * dW);

        VOld = VNew;
        x += dt;
    }

    // Assemble quantities (postprocessing)
    payoffT = myOption.myPayOffFunction(VNew);
```

```

        sumPriceT += payoffT;
        avgPayoffT += payoffT/M;
        avgPayoffT *= avgPayoffT;

        squaredPayoff += (payoffT*payoffT);
    }

    // Finally, discounting the average price
    price = std::exp(-myOption.r * myOption.T) * sumPriceT/M;

    std::cout << "Price, after discounting: " << price;

    double SD = std::sqrt((squaredPayoff / M) - sumPriceT*sumPriceT/(M*M));
    std::cout << "Standard Deviation: " << SD << ", " << std::endl;

    double SE = SD / std::sqrt(M);
    std::cout << "Standard Error: " << SE << ", " << std::endl;

    sw.Stop();
    std::cout << "Elapsed time in seconds: " << sw.GetTime() << '\n';

    return 0;
}

```

13.7 TWO-FACTOR PROBLEMS

We give two specific examples of two-factor SDEs and their numerical approximation using finite differences.

13.7.1 Spread Options with Stochastic Volatility

In this section we examine a two-factor model. The two underlying assets have stochastic volatility, and in these cases we shall have four loosely coupled SDEs. Thus, we model both the assets and their volatilities, and we approximated it using an Euler–Maruyama scheme in Duffy and Kienitz (2009); first, we simulate the variances and then we use their updated values in the numerical scheme that calculates asset prices at each time level.

In order to scope the problem, we model a two-factor spread option. By definition, a *spread call option* between two assets is one whose payoff at expiry T is given by:

$$\max(0, [S_1(T) - S_2(T)] - K)$$

where:

- S_1 = first asset value
- S_2 = second asset value
- K = strike price.

Similarly, a put option is defined as follows:

$$\max(0, K - [S_1(T) - S_2(T)]).$$

The most general case is when we have a quantity α of asset 1 and a quantity β of asset 2. In this case, the payoff function for a call option is given by:

$$\max(0, [\alpha S_1(T) - \beta S_2(T)] - K)$$

where $\alpha, \beta > 0$ are constants.

There are many examples of such options (see Geman (2005)), for example:

- *Crack spread option*: an option on the spread between heating oil and crude oil futures.
- *Spark spread*: the difference between the price of electricity and the price of the corresponding quantity of primary fuel. For example, we could have a spread between electricity and natural gas.

The volatilities may be modeled as deterministic functions of the underlying assets as we do in Chapter 23. We discuss the case where the volatilities are stochastic, and in particular we examine a European spread option when the underlying prices follow GBM, and the variance of returns follow mean-reverting square-root processes). In this case we have a system of SDEs defined by:

$$\begin{aligned} dS_1 &= rS_1 dt + \sigma_1 S_1 dW_1 \\ dS_2 &= rS_2 dt + \sigma_2 S_2 dW_2 \\ dV_1 &= \alpha_1(\mu_1 - V_1)dt + \xi_1 \sqrt{V_1} dW_3 \\ dV_2 &= \alpha_2(\mu_2 - V_2)dt + \xi_2 \sqrt{V_2} dW_4 \end{aligned}$$

where:

- S_1 = first underlying asset
- S_2 = second underlying asset
- $V_1 = \sigma_1^2$, σ_1 is volatility of S_1
- $V_2 = \sigma_2^2$, σ_2 is volatility of S_2

and:

- α_j = rate of mean reversion of variance V_j , $j = 1, 2$
- ξ_j = volatility of variance V_j , $j = 1, 2$
- W_j = Wiener process, $j = 1, \dots, 4$
- μ_j = mean reversion factor, $j = 1, 2$.

In this case we have an SDE comprising four state variables. It can be approximated by generalisations of the finite difference methods presented in this chapter. We discuss spread option pricing using PDE models in Chapter 23.

13.7.2 Heston Stochastic Volatility Model

We discuss the Heston stochastic volatility model. We are mainly interested in applying a number of finite difference schemes to its SDE model. It is an interesting problem to study for a number of reasons:

- It is popular in finance because of its ability to model option pricing problems with stochastic volatility. It allows correlation between volatility and spot-asset returns, which explains the skewness and strike-price biases in the Black–Scholes model (Heston (1993)).
- It is an example of a two-factor model with a square root non-linearity in the SDE component for the variance. It is thus a good benchmark example to test a range of finite difference schemes on. We can then apply the same schemes to other kinds of problems.

The SDE for the Heston model is given by:

$$\begin{aligned} dS &= (r - d)Sdt + \sqrt{v} SdW_1 \\ dv &= \kappa(\theta - v)dt + \xi\sqrt{v} dW_2 \\ dW_1 dW_2 &= \rho dt \end{aligned} \tag{13.29}$$

where:

- S = asset price
- v = variance
- r = interest rate (domestic rate, usually)
- d = dividend (or foreign interest rate)
- κ = strength of mean reversion
- θ = long-term average
- ξ = volatility of variance
- ρ = correlation
- $S(0)$ = initial asset price
- $v(0)$ = initial variance price and $dW_1 dW_2 = \rho dt$.

This system is mean-reverting in the variance while there is a non-linear term in both diffusion terms. Furthermore, the Wiener process increments are correlated, and we need to take this fact into account when generating the corresponding random numbers for the finite difference schemes.

From a numerical viewpoint, we see that the drift term is linear in the asset price and in the variance while the diffusion terms are non-linear. This means that some

methods cannot be used (for example, the implicit Euler method) precisely because of this non-linearity, but on the other hand we can use implicit methods for the drift terms and explicit methods for the diffusion terms. This realisation leads us to a number of schemes that we use, and we introduce the following finite difference schemes to approximate the solution of the system (13.29):

- The explicit Euler scheme.
- The Milstein scheme.
- The semi-implicit Euler scheme.
- The implicit Milstein scheme.
- The weak predictor-corrector scheme.
- The transformed semi-implicit Euler scheme.
- The transformed explicit Euler scheme.

These one-step schemes advance the solution (a two-dimensional vector) from time-level n to time-level $n + 1$ by first calculating the variance (it does not depend on the asset) and then using this value in the finite difference scheme for the asset. Furthermore, we can write the Heston SDE in three equivalent forms, namely:

- The original form (13.29) as already discussed.
- By defining untransformed, correlated Wiener increments and incorporating the correlation as an explicit parameter in the new SDE:

$$\begin{aligned} dS &= (r - d)Sdt + \sqrt{v}S dW_1 \\ dv &= \kappa(\theta - v)dt + \xi\sqrt{v} (\rho dW_1 + \sqrt{1 - \rho^2}dW_2) \\ dW_1 dW_2 &= 0. \end{aligned} \tag{13.30}$$

- By using a logarithmic transformation:

$$\begin{aligned} dX &= \left(r - d - \frac{1}{2}v \right) dt + \sqrt{v}dW_1 \\ dv &= \kappa(\theta - v)dt + \xi\sqrt{v}dW_2 \end{aligned} \tag{13.31}$$

where $X = \log(S)$, $S = e^X$.

We now have three equivalent formulations for the Heston model, and we propose several finite difference schemes to solve them. It is obvious that we wish to produce accurate and stable algorithms. At this stage, we assume that the parameters in Equation (13.29) are known. We now enumerate the schemes for the current problem based on Equation (13.29):

Explicit Euler Scheme:

$$\begin{aligned} S_{n+1} &= S_n + (r - d)S_n \Delta t + \sqrt{V_n}S_n \Delta W_{1,n} \\ V_{n+1} &= V_n + \kappa(\theta - V_n)\Delta t + \xi\sqrt{V_n} \Delta W_{2,n} \quad n = 0, \dots, N - 1 \\ S_0 &= S(0), \quad V_0 = v(0) \end{aligned} \tag{13.32}$$

where:

$$\Delta W_{j,n} = \sqrt{\Delta t} N_j, \quad N_j \sim N(0, 1) \quad j = 1, 2, \quad n = 0, \dots, N - 1$$

and W_1 and W_2 have correlation ρ .

Rearranging, we can write this scheme in the computational form:

$$\begin{aligned} S_{n+1} &= S_n(1 + (r - d)\Delta t + \sqrt{V_n}\Delta W_{1,n}) \\ V_{n+1} &= V_n(1 - \kappa\Delta t) + \kappa\theta\Delta t + \xi\sqrt{V_n}\Delta W_{2,n} \\ n &= 0, \dots, N - 1. \end{aligned} \quad (13.33)$$

Milstein Scheme

We now introduce the Milstein scheme. In the interest of readability, we define some notation for operators that act on the mesh functions:

$$\begin{aligned} L_1^{\Delta t}(S_n, V_n) &\equiv S_n(1 + (r - d)\Delta t + \sqrt{V_n}\Delta W_{1,n}) \\ L_2^{\Delta t}(S_n, V_n) &\equiv V_n(1 - \kappa\Delta t) + \kappa\theta\Delta t + \xi\sqrt{V_n}\Delta W_{2,n}. \end{aligned} \quad (13.34)$$

Then the Milstein scheme is defined by:

$$\begin{aligned} S_{n+1} &\equiv L_1^{\Delta t}(S_n, V_n) + \frac{1}{2}S_n V_n (\Delta W_{1,n} * \Delta W_{1,n} - \Delta t) \\ V_{n+1} &\equiv L_2^{\Delta t}(S_n, V_n) + \frac{\xi^2}{4}(\Delta W_{2,n} * \Delta W_{2,n} - \Delta t), \quad n = 0, \dots, N - 1 \end{aligned} \quad (13.35)$$

with:

$$S_0 = S(0), \quad V_0 = v(0).$$

Semi-Implicit Euler Scheme:

$$\begin{aligned} S_{n+1} &= S_n + (r - d)S_{n+1}\Delta t + \sqrt{V_n}S_n\Delta W_{1,n} \\ V_{n+1} &= V_n + \kappa(\theta - V_{n+1})\Delta t + \xi\sqrt{V_n}\Delta W_{2,n}, \quad n = 0, \dots, N - 1 \end{aligned} \quad (13.36)$$

with:

$$S_0 = S(0), \quad V_0 = v(0).$$

In this case we evaluate the drift terms at time level $n + 1$ while we evaluate the diffusion terms at time level n . We rewrite Equation (13.36) in the computational form:

$$\begin{aligned} S_{n+1}(1 - (r - d)\Delta t) &= S_n + \sqrt{V_n}S_n\Delta W_{1,n} = S_n(1 + \sqrt{V_n}\Delta W_{1,n}) \\ V_{n+1}(1 + \kappa\Delta t) &= V_n + \kappa\theta\Delta t + \xi\sqrt{V_n}\Delta W_{2,n} \quad n = 0, \dots, N - 1 \end{aligned} \quad (13.37)$$

Implicit Milstein Scheme:

$$\begin{aligned} S_{n+1} &= S_n + (r - d)S_{n+1}\Delta t + \sqrt{V_n}S_n\Delta W_{1,n} + \frac{1}{2}S_nV_n(\Delta W_{1,n} * \Delta W_{1,n} - \Delta t) \\ V_{n+1} &= V_n + \kappa(\theta - V_{n+1})\Delta t + \xi\sqrt{V_n}\Delta W_{2,n} + \frac{\xi^2}{4}(\Delta W_{2,n} * \Delta W_{2,n} - \Delta t), \quad n = 0, \dots, N-1 \end{aligned} \quad (13.38)$$

with:

$$S_0 = S(0), \quad V_0 = v(0).$$

Weak Predictor-Corrector Scheme:

Predictor

$$\begin{aligned} \bar{S}_{n+1} &= S_n + (r - d)S_n\Delta t + \sqrt{V_n}S_n\Delta W_{1,n} \\ \bar{V}_{n+1} &= \kappa(\theta - V_n)\Delta t + \xi\sqrt{V_n}\Delta W_{2,n}, \quad n = 0, \dots, N-1 \end{aligned} \quad (13.39)$$

Corrector

$$\begin{aligned} S_{n+1} &= S_n + (r - d)\bar{S}_{n+1}\Delta t + \sqrt{V_n}S_n\Delta W_{1,n} \\ V_{n+1} &= V_n + \kappa(\theta - \bar{V}_{n+1})\Delta t + \xi\sqrt{V_n}\Delta W_{2,n}, \quad n = 0, \dots, N-1 \end{aligned} \quad (13.40)$$

with:

$$S_0 = S(0), \quad V_0 = v(0).$$

Transformed Semi-Implicit Euler Method (using Equation (13.31)):

$$\begin{aligned} X_{n+1} &= X_n + \left(r - d - \frac{1}{2}V_{n+1}\right)\Delta t + \sqrt{V_n}\Delta W_{1,n} \\ V_{n+1} &= V_n + \kappa(\theta - V_{n+1})\Delta t + \xi\sqrt{V_n}\Delta W_{2,n} \\ X_n &\equiv \log(S_n), \quad n = 0, \dots, N-1 \end{aligned} \quad (13.41)$$

with:

$$S_0 = S(0), \quad V_0 = v(0).$$

Transformed Explicit Euler Scheme (using Equation (13.31)):

$$\begin{aligned} X_{n+1} &= X_n + \left(r - d - \frac{1}{2}V_n\right)\Delta t + \sqrt{V_n}\Delta W_{1,n} \\ V_{n+1} &= V_n + \kappa(\theta - V_n)\Delta t + \xi\sqrt{V_n}\Delta W_{2,n} \quad n = 0, \dots, N-1 \end{aligned} \quad (13.42)$$

with:

$$S_0 = S(0), \quad V_0 = v(0).$$

13.8 THE ITO FORMULA

We introduce the *Ito formula* and it can be used for one-factor and multi-factors SDEs that are driven by a range of processes.

Theorem 13.4 (One-dimensional Ito formula)

Let X be an *Ito process* defined by:

$$dX = \mu(t, X)dt + \sigma(t, X)dW.$$

Let $g(t, x) \in C^2([0, \infty] \times \mathbb{R})$, i.e g is twice continuously differentiable in x and t . Then $y(t) \equiv g(t, x(t))$ is also an Ito process satisfying the SDE:

$$dY = \frac{\partial g}{\partial t}dt + \frac{\partial g}{\partial x}dX + \frac{1}{2} \frac{\partial^2 g}{\partial x^2}(dX)^2$$

where $(dX)^2 = dX \cdot dX$ is computed using $dt \cdot dt = dt \cdot dW = dW \cdot dt = 0$ and $dW \cdot dW = dt$.

This formula is useful because we can transform an SDE into a simpler one that we can solve. To show how to use the formula, we examine the SDE:

$$dS = rSdt + \sigma SdW.$$

Using the transformation $Y = f(t, S) = \log(S)$, checking that $\frac{\partial f}{\partial t} = 0$, $\frac{\partial f}{\partial S} = \frac{1}{S}$, $\frac{\partial^2 f}{\partial S^2} = -\frac{1}{S^2}$ we can deduce (by a bit of algebra) that:

$$df = \left(r - \frac{1}{2}\sigma^2 \right) dt + \sigma dW.$$

We take another example. The SDE for an underlying commodity is defined by $dS = \mu Sdt + \sigma SdW$ and let the forward price F be defined by $F = Se^{\tau(T-t)}$. Then F satisfies the SDE :

$$dF = (\mu - r)F + \sigma FdW.$$

13.9 STOCHASTICS MEETS PDEs

In this section we show that there is close relationship between statistics, stochastic processes and PDEs.

13.9.1 A Statistics Refresher

The *joint cumulative distribution function* (or joint cdf) of two continuous random variables X and Y is the function defined by:

$$F_{XY}(x, y) = P(X \leq x, Y \leq y).$$

Then the *joint probability density function* (joint pdf) is defined by what is actually a PDE:

$$f_{XY}(x, y) = \frac{\partial^2 F_{XY}(x, y)}{\partial x \partial y}. \quad (13.43)$$

We can integrate this equation to get the following relation:

$$F_{XY}(x, y) = \int_{-\infty}^x \int_{-\infty}^y f_{XY}(\xi, n) d\xi d\eta$$

The *marginal distribution* functions are defined by:

$$\begin{aligned} f_X(x) &= \int_{-\infty}^{\infty} f_{XY}(x, y) dy \\ f_Y(y) &= \int_{-\infty}^{\infty} f_{XY}(x, y) dx. \end{aligned}$$

The conditional pdf of Y given that $X = x$ is defined by:

$$\begin{aligned} f_{Y|X}(y|x) &= \frac{f_{XY}(x, y)}{f_X(x)}, \quad f_X(x) > 0. \\ f_{X|Y}(x|y) &= \frac{f_{XY}(x, y)}{f_Y(y)}, \quad f_Y(y) > 0. \end{aligned}$$

Finally, the *conditional expectation* of Y given $X = x$ is defined by:

$$\begin{aligned} \mu_{Y|x} &= E(Y|X) = \int_{-\infty}^{\infty} y f_{Y|x}(y|x) dy \\ &= \int_{-\infty}^{\infty} y \frac{f_{XY}(x, y)}{f_X(x)} dy \\ &= \frac{\int_{-\infty}^{\infty} y f_{XY}(x, y) dy}{\int_{-\infty}^{\infty} f_{XY}(x, y) dy}. \end{aligned}$$

We discuss an example, namely the *bivariate normal distribution* whose pdf is given by:

$$f(x, y) = \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)}(x^2 - 2\rho xy + y^2)\right) \quad (x, y \in \mathbb{R}, -1 < \rho < 1). \quad (13.44)$$

This function and its cdf can be used to compute two-factor option prices, as we shall see in Chapter 23. Furthermore, in Duffy (2018) we computed the cdf by solving the hyperbolic *Goursat PDE* (13.43) using the finite difference method, which is a novel and generic way to solve such problems in our opinion.

13.9.2 The Feynman–Kac Formula

This theorem offers a method to solve certain PDEs by simulating random paths of a stochastic process. It also states that expectations of random processes can be computed by deterministic methods. To this end, consider the PDE:

$$\begin{aligned} \frac{\partial u(x, t)}{\partial t} + \mu(x, t) \frac{\partial u(x, t)}{\partial x} + \frac{1}{2} \sigma^2(x, t) \frac{\partial^2 u(x, t)}{\partial x^2} - V(x, t)u(x, t) + f(x, t) = 0 \\ (x \in \mathbb{R}, \quad 0 < t < T) \end{aligned} \quad (13.45)$$

with terminal condition:

$$u(x, T) = \psi \quad (\mu, \sigma, \psi, V, f \text{ are known functions}). \quad (13.46)$$

Then the solution of the *Cauchy problem* (13.45), (13.46) can be written as a conditional expectation:

$$u(x, t) = E^Q \left[\int_0^t d(r) f(X_r, r) dr + d(r) \psi(X_r) | X_t = x \right] \quad (13.47)$$

where we use the shorthand notation

$$d(r) = \exp \left(- \int_t^\tau V(X_\tau, \tau) d\tau \right).$$

Under probability measure Q , X is an Ito process driven by the SDE:

$$\begin{aligned} dX = \mu(X, t) dt + \sigma(X, t) dW^Q \\ X(0) = x \end{aligned} \quad (13.48)$$

with $W^Q(t)$ being a Wiener process (Brownian motion) under Q .

The expectation (13.47) can be computed using Monte Carlo methods (as we have discussed in this chapter) or by quasi-Monte Carlo methods.

The formula (13.45) is also valid for n -dimensional Ito diffusions where the corresponding PDE is given by:

$$\frac{\partial u}{\partial t} + \sum_{i=1}^N \mu_i(x, t) + \frac{1}{2} \sum_{i=j}^N \sum_{j=1}^N \gamma_{ij}(x, t) \frac{\partial^2 u}{\partial x_i \partial x_j} - r(x, t)u = f(x, t), \quad (13.49)$$

where:

$$\gamma_{ij}(x, t) = \sum_{k=1}^n \sigma_{ik}(x, t) \sigma_{jk}(x, t), \quad (\gamma = \sigma \sigma^\top, \text{ where } \sigma^\top \text{ denotes the transpose of matrix } \sigma).$$

13.9.3 Kolmogorov Equations

Kolmogorov equations characterise stochastic processes and they describe the probability that a stochastic process is in a certain state as a function of time. Special cases are *Kolmogorov forward equation* (also known as the *Fokker–Planck equation* (FPE)) and *Kolmogorov backward equation* (KBE).

Both equations model the *transition probability density function*:

$$P(a < y < b \text{ at time } t' | y \text{ at time } t) = \int_a^b p(y, t; y', t') dy'. \quad (13.50)$$

The function $p(y, t; y', t')$ represents the probability that the random variable y is in the interval (a, b) at time t' in the future, given that its state at time t is $y(t < t')$.

13.9.4 Kolmogorov Forward (Fokker–Planck (FPE)) Equation

We assume that the state of a system at time t is given by the probability distribution $p_t(x)$. We wish to know the probability distribution of the state at a later time $s > t$. In this case $p_t(x)$ serves as the initial condition and the FPE PDE is integrated forward in time. In many cases the initial state is known exactly in the form of a *Dirac delta function* centred on the initial state. The PDE is:

$$\frac{\partial p(x, s)}{\partial s} = -\frac{\partial}{\partial x}[\mu(x, s)p(x, s)] + \frac{1}{2}\frac{\partial^2}{\partial x^2}[\sigma^2(x, s)p(x, s)], \quad s \geq t \quad (13.51)$$

$$p(x, t) = p_t(x) \text{ (initial condition)}$$

where we assume that the system state X evolves according to the SDE in Equation (13.48).

In general, (13.51) does not have an analytical solution, and we solve it using numerical methods, such as finite differences or finite elements and we have to pay attention to the imposition of appropriate boundary and initial conditions, the latter being a delta function, for example.

In some very simple cases we are able to find analytical solutions. To this end, we take the standard scalar Wiener process that is generated by the SDE which is a special case of SDE (13.1):

$$dX = dW \quad (13.52)$$

In this case the Fokker–Planck equation becomes ($\mu = 0, \sigma = 1$):

$$\begin{aligned} \frac{\partial p(x, t)}{\partial t} &= \frac{1}{2} \frac{\partial^2 p(x, t)}{\partial x^2}, \quad -\infty < x < \infty \\ p(x, 0) &= \delta(x) \text{ (delta function)} \end{aligned} \quad (13.53)$$

which is a simple diffusion equation. Given a delta function as initial condition $p(x, 0) = \delta(x)$ we get the solution:

$$p(x, t) = \frac{1}{\sqrt{2\pi t}} e^{-x^2/2t}. \quad (13.54)$$

We can find the solution (13.53) by Fourier methods, for example. Another example is the *Ornstein–Uhlenbeck process* defined by:

$$dX = -aXdt + \sigma dW. \quad (13.55)$$

In this case the Fokker–Planck equation becomes ($\mu = -ax$, $\sigma = \text{constant}$):

$$\frac{\partial p(x, t)}{\partial t} = a \frac{\partial}{\partial x}(xp(x, t)) + \frac{\sigma^2}{2} \frac{\partial^2 p(x, t)}{\partial x^2}. \quad (13.56)$$

13.9.5 Multi-Dimensional Problems and Boundary Conditions

The multi-dimensional Fokker–Planck equation is given by:

$$\frac{\partial p(x, t)}{\partial t} = - \sum_{i=1}^n \frac{\partial}{\partial x_i} [\mu_i(x, t)p(x, t)] + \sum_{i,j=1}^n \frac{\partial^2}{\partial x_i \partial x_j} [D_{ij}(x, t)p(x, t)] \quad (13.57)$$

where:

$$D_{ij}(x, t) = \frac{1}{2} \sum_{k=1}^n \sigma_{ik}(x, t)\sigma_{jk}(x, t) \text{ (diffusion tensor)} \quad i, j = 1, \dots, n.$$

In general, this equation needs to be approximated by numerical methods: for example, the finite difference methods that we introduce in Part D of this book. We need to specify the domain in which (13.57) holds as well as boundary conditions or asymptotic values of the solution.

We are now ready to consider different kinds of boundary conditions. We assume that (13.57) is satisfied in a domain Ω with boundary Γ .

1. Reflecting barrier

The process cannot leave the region Ω , hence there is zero net flow of probability across Γ .

2. Absorbing barrier

The process is removed from the system at the moment that it reaches Γ . In other words, the *barrier absorbs*:

$$p(x, t) = 0, \quad x \in \Gamma, \quad t > 0.$$

3. Natural boundaries

The natural boundary condition is applied when the range of the random variable $X(t)$ is infinite or semi-infinite. In this case we have $\lim_{x \rightarrow \pm\infty} p(x, t) = 0$ with the decay to zero being sufficient fast to ensure the *normalisation integral* is unity:

$$\int_{-\infty}^{\infty} p(x, t)dx = 1, \quad x \in \Gamma, \quad t > 0.$$

In one dimension, this condition requires that $p(x, t)$ tends to zero faster than $|x|^{-1}$ as $|x| \rightarrow \infty$.

4. Boundaries at infinity

The first three kinds of boundaries can be considered at infinity provided that we can simultaneously guarantee:

$$\lim_{x \rightarrow \infty} p(x, t) = 0$$

$$\lim_{x \rightarrow \infty} \frac{\partial p}{\partial x}(x, t) = 0, \quad x \in \Gamma, \quad t > 0.$$

In applications, we must decide which boundary conditions are most appropriate. For example, a discussion for the CIR, Heston and LSV models is given in Lucic (2012). Further discussion is given in Feller (1951) and Glasserman (2004). In particular, the transition density for the CIR process is known analytically; it is a non-central chi-squared distribution.

The CIR process is (that we discuss in Chapter 25):

$$dv = \kappa(\theta - v)dt + \sigma\sqrt{v}dW, \quad v_0 = v(0) > 0 \quad (13.58)$$

and the corresponding FPE is:

$$\frac{\partial p}{\partial t} = \frac{\partial^2}{2 \partial v^2} (vp) - \frac{\partial}{\partial v} (\kappa(\theta - v)p)$$

$$p(v, 0) = \delta(v - v_0) \text{ (delta function)}. \quad (13.59)$$

An interesting (open?) question is to investigate the applicability of the Fichera theory to the FPE equation. In particular, what is the behaviour at $v = 0$? (See Lewis (2016)).

13.9.6 Kolmogorov Backward Equation (KBE)

This equation is useful when we are interested in knowing whether the system will be in a given subset of states B (the *target set*) at a future time s from current time t . The target is described by a given function:

$$\frac{\partial p(x, t)}{\partial t} + \sum_{i=1}^n \frac{\partial F_i}{\partial x_i}(x, t) = 0 \quad (13.60)$$

where:

$$F_i(x, t) = \mu_i(x, t)p(x, t) - \frac{1}{2} \sum_{j=1}^n \frac{\partial}{\partial x_j} (a_{ij}(x, t)p(x, t)) \quad i = 1, \dots, n. \quad (13.61)$$

The objective is to know what the so-called *hit probability* is of ending up in the target set at time s for every state x at time t .

13.10 FIRST EXIT-TIME PROBLEMS

We consider a random process, and we wish to answer the following question: When does the process first encounter a threshold, barrier, boundary or specified state of the system? By definition, the *first hitting time* (also known as *first passage time*) is the first time that the process encounters the threshold. In other words, it is the time taken for the process to reach a certain value.

We can compute the probability of the first exit time from a bounded domain for multi-dimensional distributions. To this end, let $(\Omega, (F_t)_{t \geq 0}, P)$ be a filtered probability space, and $X := (X^1, \dots, X^d)$ (with $x := (x_1, \dots, x_d)$ as initial condition) be the solution to the following system of stochastic differential equations:

$$dX_t^i = b_i(X_t)dt + \sum_{j=1}^d \sigma_{ij}(X_t)dW_t^j, \quad X_0 = x_i \in \mathbb{R}, \quad 1 \leq i \leq d$$

where $W := (W^1, \dots, W^d)^\top$ is a d -dimensional standard Brownian motion with $d \geq 1$ and the coefficients b_i and σ_{ij} are smooth for $1 \leq i, j \leq d$.

We denote the *infinitesimal generator* of X which has the form:

$$Lf(x) = \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d a_{ij}(x) \frac{\partial^2 f(x)}{\partial x_i \partial x_j} + \sum_{i=1}^d b_i(x) \frac{\partial f(x)}{\partial x_i}$$

where $a = \sigma^\top \sigma$ and $f \in C_K^2(\mathbb{R}^d)$, the space of twice continuously differentiable functions on \mathbb{R}^d with compact support K . In other words, these functions are identically zero outside a closed bounded subset K of \mathbb{R}^n .

Let A be an open bounded Borel subset of \mathbb{R}^d with boundary ∂A . We define the *stopping time*:

$$\tau_A = \inf\{u \geq 0; X_u \notin A\}.$$

This is the first exit time of X from domain A where we assume that ∂A is smooth. Let us denote by $Q(t, x)$ the probability that X starting from x did not exit the domain A before t , that is:

$$Q(t, x) = 1 - P_x(\tau_A < t).$$

Then Q coincides with the solution of the following *backward Kolmogorov equation*:

$$\frac{\partial u}{\partial t}(t, x) = Lu(t, x) \text{ on } (t, x) \in \mathbb{R}^+ \times A \quad (13.62)$$

associated to the process X with initial condition and boundary conditions:

$$\begin{aligned} u(0, x) &= 1, \quad x \in A \\ u(t, x) &= 0, \quad x \in \partial A, \quad t > 0. \end{aligned} \quad (13.63)$$

For a detailed numerical discussion of this problem, see Patie and Winter (2008). These authors use Monte Carlo simulation and finite element method. Of course, the same problem can be approximated by any one of the finite difference methods in this book. See also Ren, Madan and Qian (2007), where the authors use splitting methods to solve the two-factor Fokker–Planck equation in the context of calibration.

13.11 SUMMARY AND CONCLUSIONS

In this chapter we gave an overview of the connection between PDEs and stochastic analysis. Both approaches are used in applications, and to this end we discuss several useful techniques that can be used for option pricing and calibration of stochastic volatility models.

The added value of this chapter in our opinion is that it forms a bridge between the world of stochastic calculus, probability and SDEs on the one hand and the world of PDEs on the other hand. It is now possible to view them as two sides of the same coin. Both worlds are amenable to qualitative (existence and uniqueness theorems) and quantitative (simulation, numerical analysis) analysis. This results in deeper insights than just working in either an ‘SDE or PDE silo’.

The Foundations of the Finite Difference Method (FDM)

Mathematical and Numerical Foundations of the Finite Difference Method, Part I

How far my efforts agree with those of other philosophers I will not decide. Indeed what I have here written makes no claim to novelty in points of detail; and therefore I give no sources, because it is indifferent to me what I have thought has already been thought before me by another.

Ludwig Wittgenstein

14.1 INTRODUCTION AND OBJECTIVES

In general, it is not possible to find analytical solutions to partial differential equations, initial value problems and initial boundary value problems, especially for non-linear problems. More commonly, *closed* solutions can be found in terms of series of special functions, for example Bessel functions or orthogonal polynomials (Abramowitz and Stegun (1965), Chihara (1978)). A discussion of these important methods is unfortunately outside the scope of this book. Instead, we *discretise* a PDE by replacing its derivatives in space and time by divided differences based on function values that we evaluate at discrete mesh points. We replace an infinite-dimensional problem by a finite-dimensional one that we can solve in a finite number of steps. We approximate the solution of the original problem P1 by the solution of an approximate problem P2. We may have a number of general concerns before we even consider which specific finite difference scheme to use:

- Does P2 preserve certain *qualitative properties* of P1, such as positivity, monotonicity and conservation of energy? Does it satisfy a *discrete maximum principle*?
- How good is P2 as approximation to P1 as a function of the mesh sizes in the space and time variables? In particular, are we interested in first-order, second-order or even higher order schemes?

- Is P2 *well-posed* in the sense that small changes in the input data (for example, initial conditions or boundary conditions) lead to small changes in the approximate solution?
- Computational efficiency: Can P2 be solved in a reasonable amount of time and memory resources? In other words, does the algorithm that implements P2 have good run-time performance?

We make these requirements more precise by interpreting them in the light of finite difference theory. In particular, we are interested in the following defining properties of a finite difference scheme that approximate a time-dependent PDE:

- *Consistency*: Does P2 ‘become’ P1 as the mesh sizes in space and time tend to zero?
- *Stability*: Is the norm of the solution of P2 at each time level bounded by norms of the input data?
- *Convergence*: Is the norm of the difference between the solutions of P2 and P1 at each time level bounded by some polynomial power of the mesh sizes?

The standard techniques for proving stability are *Fourier analysis* (and the related von Neumann criterion for stability) and using the theory of matrices. The method is strictly speaking only valid for initial values for PDEs with constant coefficients.

We focus on model time-dependent convection, diffusion and convection-diffusion partial differential equations, in *particular initial-boundary value problems* for these kinds of PDEs. A good discussion is Thomas (1995).

14.2 NOTATION AND PREREQUISITES

In Part A of the book, we introduced the finite difference method, and we focused mainly on examples to motivate its application to ordinary differential equations and relatively simple one-dimensional partial differential equations. No attention was paid to whether the schemes were a ‘good’ approximation to the differential equation or whether they satisfied a discrete form of the maximum principle. We now formalise discussions in this chapter and the next three chapters by a detailed study based on the following use cases:

- U1: Standardised notation as far as possible. We wish to become familiar with the symbols and then use them in a consistent manner as we progress in the book.
- U2: Derive finite difference schemes from first principles by taking concrete examples (for example, the one-dimensional heat equation) to learn the fundamentals before proceeding to more complex cases. To quote Paul Halmos again: ‘it is frequent in mathematics that every instance of a concept of seemingly great generality is in essence the same as a small and concrete special case’.
- U3: Define a number of finite difference schemes for a given problem and compare them with regards to their stability, accuracy and performance metrics.
- U4: Take simpler problems and understand them before moving to bigger problems.

In Part C we concentrate on one-dimensional diffusion, convection and convection-diffusion-reaction equations, as they are the building blocks for both one-factor and two-factor PDE models in finance. We take a building-block approach by composing low-dimensional and simple schemes into more complex ones.

We try to standardise notation in Part C by taking model PDEs, and we adhere to this notation as we progress in the book. In general, we are looking for efficient difference schemes that are easy to implement and that approximate the solution of a PDE to a desired accuracy.

14.3 WHAT IS THE FINITE DIFFERENCE METHOD, REALLY?

In this section we give a high-level overview of what the finite difference method is, its relationship to partial differential equations and the major steps that need to be executed in order to design and implement algorithms that realise these finite difference schemes in C++, for example. It is important to know not only the fundamental principles but also how to use them effectively when designing algorithms, in particular choosing the most optimal scheme for a given problem.

We paraphrase the steps to be executed when approximating the solution of an initial boundary value problem (IBVP) by the finite difference method. We note that the specification of the IBVP is *qualitative* in the sense that it is an unambiguous description, but it does not tell us how to actually find the solution. The solution is defined in all of space and for all time. Mapping the IBVP to a more quantitative formulation involves a number of well-defined steps:

- S1: Preprocess the IBVP to make it suitable as input to the finite difference method.
- S2: Do we need to pay special attention to issues such as convection dominance, non-linearities, discontinuities and other problems that the proposed finite difference scheme should address? We may need special schemes for special problems.
- S3: Create a discrete domain in space and time, choosing between uniform, non-uniform and adaptive meshes.
- S4: Approximate the PDE, in other words approximate derivatives by divided differences, numerical boundary conditions (many viable alternatives that satisfy the requirements), employing one-step and multi-step methods for time discretisation.
- S5: Assemble and solve the non-linear/linear discrete system of equations. At this stage we can use standard non-linear and linear solvers.

14.4 FOURIER ANALYSIS OF LINEAR PDEs

Fourier analysis is often used to obtain solutions to initial value problems for PDEs or to perform some theoretical analysis. Specifically, we apply the *Fourier transform* to a PDE on an infinite interval to give an ordinary differential equation (ODE) whose solution is readily found. We can then apply the *inverse Fourier transform* to the ode to recover the solution of the original PDE. This *transform technique* is common in mathematics: popular transforms are Laplace, Mellin, Hankel and Weierstrass (which uses the Green's function for the heat equation). For a detailed discussion, see Zeemanian (1987).

We shall also see that Fourier analysis is important in the study of finite difference schemes for linear initial value problems. In particular, we introduce the von Neumann stability analysis to prove stability of finite difference schemes for linear initial value problems.

In the following discussion we are assuming that functions are defined in the Hilbert space of *square-integrable functions* on the real line:

$$L_2[-\infty, \infty] = \left\{ f : (-\infty, \infty) \rightarrow \mathbb{R}, \|f\|_2 \equiv \left(\int_{-\infty}^{\infty} |f(x)|^2 dx \right)^{1/2} < \infty \right\} \quad (14.1)$$

with associated *inner product* and *metric*:

$$\begin{aligned} (f, g) &= \int_{-\infty}^{\infty} f(x)g(x)dx \\ d(f, g) &= \|f - g\|_2 \end{aligned} \quad \left. \right\} \quad f, g \in L_2[-\infty, \infty]. \quad (14.2)$$

We need two important definitions. First, the *Fourier transform* of an integrable function f is defined by:

$$F[f](\omega) \equiv \hat{f}(\omega) \equiv \int_{-\infty}^{\infty} f(x)e^{-i\omega x} dx, \quad (i = \sqrt{-1}) \quad (14.3)$$

and the *inverse Fourier transform* is defined by:

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\omega)e^{i\omega x} d\omega. \quad (14.4)$$

An important relationship between the Fourier transform and its inverse is *Parseval's theorem* that preserves norms:

$$\int_{-\infty}^{\infty} |f(x)|^2 dx = \int_{-\infty}^{\infty} |\hat{f}(t)|^2 dt. \quad (14.5)$$

As a simple example, we examine the rectangular pulse signal:

$$f(x) = \begin{cases} 1, & |x| < a \\ 0, & |x| > a \end{cases} \quad (14.6)$$

whose Fourier transform is given by:

$$\begin{aligned} \hat{f}(\omega) &\equiv \int_{-\infty}^{\infty} f(x)e^{-i\omega x} dx = \int_{-a}^a e^{-i\omega x} dx \\ &= -\frac{1}{i\omega} (e^{i\omega a} - e^{-i\omega a}) = 2 \frac{\sin \omega a}{\omega}. \end{aligned} \quad (14.7)$$

It is possible to compute the inverse Fourier transform of (14.7) to recover the original pulse function (14.6).

14.4.1 Fourier Transform for Advection Equation

The initial value problem (IVP) for the one-dimensional advection (convection) equation is given by:

$$\begin{aligned} \frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} &= 0, \quad -\infty < x < \infty \quad t > 0 \\ u(x, 0) &= f(x). \end{aligned} \tag{14.8}$$

Here the parameter a is a constant (it can be positive or negative), and we can verify that the solution is given by $u(x, t) = f(x - at)$. IVP (14.8) plays an important role in applications because it corresponds to the drift component in the Black-Scholes equation, for example. In general we can approximate (14.8) and its generalisations using finite differences, and we must take care of how we do this, especially at *downstream boundaries* (see Vichnevetsky and Bowles (1982), pp. 97–102; see also Chapter 24 of the current book).

In this subsection we reproduce the analytical solution of (14.8) using Fourier analysis. By definition, the Fourier transform and its inverse are:

$$\begin{aligned} \hat{u}(\omega, t) &= \int_{-\infty}^{\infty} u(x, t) e^{-i\omega x} dx \\ u(x, t) &= \int_{-\infty}^{\infty} \hat{u}(\omega, t) e^{i\omega x} d\omega. \end{aligned} \tag{14.9}$$

Differentiating the second equation in (14.9) with respect to x and t (we can take differentiation inside the integral term) leads to:

$$\begin{aligned} \frac{\partial u}{\partial t}(x, t) &= \int_{-\infty}^{\infty} \frac{d\hat{u}}{dt}(\omega, t) e^{i\omega x} d\omega \\ \frac{\partial u}{\partial x}(x, t) &= \int_{-\infty}^{\infty} \hat{u}(\omega, t) i\omega e^{i\omega x} d\omega. \end{aligned} \tag{14.10}$$

Using these results in (14.8) leads to the ODE:

$$\begin{aligned} \frac{d\hat{u}}{dt}(\omega, t) + ai\omega\hat{u}(\omega, t) &= 0 \\ \hat{u}(\omega, 0) = \hat{f}(\omega) &= \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx \end{aligned} \tag{14.11}$$

with solution $\hat{u}(\omega, t) = e^{-i\omega at}\hat{f}(\omega)$. Then from the second equation in (14.9) we finally get:

$$u(x, t) = \int_{-\infty}^{\infty} e^{-i\omega at} \hat{f}(\omega) e^{i\omega x} dx = \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega(x-at)} dx = f(x - at). \tag{14.12}$$

which is the desired answer.

Some remarks on the solution are:

- Differentiation with respect to x becomes multiplication by $i\omega$ after Fourier transforming. Then the PDE in (14.8) becomes an ODE (14.11) in the variable t .
- We have an infinite number of uncoupled ODEs in (14.11), one for each value of ω .
- Another way to analyse (14.8) is to apply *Method of Lines* (MOL) by using centred finite difference scheme in space, resulting in a system of ODEs:

$$\frac{du_j}{dt} + a \frac{u_{j+1} - u_{j-1}}{2h} = 0, \quad j = 0, \pm 1, \pm 2, \dots \quad (14.13)$$

$$u_j = u_j(t).$$

We discuss MOL in Chapter 20.

We finally note that (14.8) is an IVP on an infinite interval, and hence no boundary conditions enter the discussion. In later chapters we shall discuss initial boundary value problems for first-order hyperbolic partial differential equations.

14.4.2 Fourier Transform for Diffusion Equation

We now consider the important case of heat flow in an infinite rod:

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2}, \quad -\infty < x < \infty, \quad t > 0 \quad (14.14)$$

$$u(x, 0) = f(x), \quad -\infty < x < \infty.$$

In this case there are no boundary conditions, but we do place some restrictions on the solution, for example that u and its derivative with respect to the variable x should tend to zero at plus and minus infinity. Again, we can apply the Separation of Variables technique, but in contrast to a finite rod (where the eigenvalues are discrete), the eigenvalues vary continuously in this case. After a lengthy analysis (Tolstov (1962)) we produce a solution to problem (14.14):

$$u(x, t) = \frac{1}{2a\sqrt{\pi t}} \int_{-\infty}^{\infty} f(s) e^{-\frac{(x-s)^2}{4a^2 t}} ds.$$

From this equation we can see that the temperature approaches zero for very large x (the heat spreads out). We can also see how the initial temperature $f(x)$ influences the subsequent evolution of the temperature in the rod.

Incidentally, the function:

$$G(x, t; \xi, 0) \equiv \frac{1}{2a\sqrt{\pi t}} e^{-\frac{(x-\xi)^2}{4a^2 t}}$$

is called the *Gauss–Weierstrass kernel* or *influence* function, and it is important in stochastic calculus and Brownian motion applications; see Zeemanian (1987). Furthermore, this function has the following properties:

$$\int G(x, t; 0, 0) dx = 1 \quad \forall t > 0$$

$$\lim_{t \rightarrow 0+} \int G(x, t; 0, 0) f(x) dx = f(0).$$

We see that this function is essentially a *transition density for Brownian motion*. Consider again the *Cauchy problem* on an infinite interval:

$$\begin{aligned} \frac{\partial u}{\partial t} &= a^2 \frac{\partial^2 u}{\partial x^2}, \quad -\infty < x < \infty, \quad t > 0 \\ u(x, 0) &= f(x), \quad -\infty < x < \infty \\ u(x, t), \quad \frac{\partial u}{\partial x}(x, t) &\rightarrow 0 \text{ as } x \rightarrow \pm\infty, \quad t > 0. \end{aligned} \tag{14.15}$$

We now apply the Fourier transform to the initial value problem (14.15) to produce an ordinary differential equation in the *transformed domain*:

$$\begin{aligned} U'(\omega, t) + a^2 \omega^2 U(\omega, t) &= 0, \quad t > 0 \\ U(\omega, 0) &= F(\omega) \end{aligned}$$

where

$$\begin{aligned} \mathcal{F}[u](\omega, t) &= U(\omega, t) \\ \mathcal{F}[f](\omega) &= F(\omega). \end{aligned} \tag{14.16}$$

In order to recover the original solution, we apply the inverse Fourier transform defined by:

$$\mathcal{F}^{-1}[F](x) = f(x) = \int_{-\infty}^{\infty} F(\omega) e^{i\omega x} d\omega.$$

We find that the solution of (14.16) is given by:

$$U(\omega, t) = F(\omega) e^{-a^2 \omega^2 t}.$$

The inverse Fourier transform applied to $U(\omega, t)$ produces the solution of (14.15).

14.5 DISCRETE FOURIER TRANSFORM

In this section we introduce the discrete analogue of the Fourier transform, and we apply it to study the stability of finite difference schemes for initial value problems.

14.5.1 Finite and Infinite Dimensional Sequences and Their Norms

We introduce the concept of a *norm* in discrete finite-dimensional and discrete infinite-dimensional space. A norm measures the size of a finite or infinite sequence of real or complex-valued numbers. We focus on one-dimensional sequences in this chapter.

A norm is a precise mathematical concept (see Bronson (1989)). First, a *semi-norm* for vector x is a mapping $p(x)$ such that:

- (1) $p(x) \geq 0$
 - (2) $p(\lambda x) = |\lambda| p(x), \lambda \in \mathbb{R}$
 - (3) $p(x + y) \leq p(x) + p(y)$
- (14.17)

for any vectors x and y . If, in addition the following holds:

$$(4) \quad p(x) = 0 \iff x = 0 \quad (14.18)$$

then we say that p is a *norm*.

We now turn our attention to defining specific norms for finite-dimensional vectors. We define the following four vector norms:

Euclidean (l_2) norm

$$\|x\|_2 = \left(\sum_{j=1}^n x_j^2 \right)^{\frac{1}{2}}$$

l_1 norm

$$\|x\|_1 = \sum_{j=1}^n |x_j| \quad (14.19)$$

l_∞ norm

$$\|x\|_\infty = \max_{1 \leq j \leq n} |x_j|$$

l_p norm

$$\|x\|_p = \left(\sum_{j=1}^n |x_j|^p \right)^{\frac{1}{p}}.$$

We now turn our attention to the infinite dimensional case. We define the following spaces, each endowed with its own norm:

$$l_2 = \left\{ u = (\dots, u_{-1}, u_0, u_1 \dots)^\top : \sum_{j=-\infty}^{\infty} |u_j|^2 < \infty \right\}$$

$$\|u\|_2 = \sqrt{\sum_{j=-\infty}^{\infty} |u_j|^2} \quad (14.20)$$

$$\|u\|_{2,\Delta x} = \sqrt{\sum_{j=-\infty}^{\infty} |u_j|^2 \Delta x} \quad (\text{energy norm } l_{2,\Delta x}).$$

$$l_\infty = \left\{ u = (\dots, u_{-1}, u_0, u_1, \dots)^\top : \sup_{-\infty < j < \infty} |u_j| < \infty \right\}$$

$$\|u\|_\infty = \sup_{-\infty < j < \infty} |u_j|. \quad (14.21)$$

This is sufficient notation to allow us to proceed.

14.5.2 Discrete Fourier Transform (DFT)

We now introduce the discrete variant of the continuous Fourier transform. We apply the DFT to a finite difference scheme that will allow us to prove that the scheme is stable (or otherwise).

Let $u = (\dots, u_{-1}, u_0, u_1, \dots)^\top$ be an infinite sequence of values.

Then the DFT (given in Thomas (1995)) is defined as:

$$\hat{u}(\xi) = \frac{1}{\sqrt{2\pi}} \sum_{n=-\infty}^{\infty} e^{-in\xi} u_n. \quad (14.22)$$

Using this definition, we can apply it to the study of finite difference schemes for linear initial value problems. In particular, we use it to transform an arbitrary finite difference scheme to a simpler form.

We take the example of the explicit Euler scheme for the heat equation that we write in the form:

$$\frac{u_j^{n+1} - u_j^n}{k} = \frac{1}{h^2} (u_{j+1}^n - 2u_j^n + u_{j-1}^n) \quad (a)$$

$$u_j^{n+1} = \lambda u_{j-1}^n + (1 - 2\lambda)u_j^n + \lambda u_{j+1}^n \quad (b)$$

where ($\lambda \equiv a^2 k / h^2$).

Applying the DFT (14.22) to both sides of (14.23) (b) gives the following sequence of results:

$$\begin{aligned}
 & \frac{1}{\sqrt{2\pi}} \sum_{j=-\infty}^{\infty} e^{-ij\xi} u_j^{n+1} \equiv \hat{u}^{n+1}(\xi) \\
 &= \frac{1}{\sqrt{2\pi}} \left\{ \sum_{j=-\infty}^{\infty} e^{-ij\xi} (\lambda u_{j-1}^n + (1-2\lambda)u_j^n + \lambda u_{j+1}^n) \right\} \\
 &= \frac{\lambda}{\sqrt{2\pi}} \sum_{j=-\infty}^{\infty} e^{-ij\xi} u_{j-1}^n + \frac{(1-2\lambda)}{\sqrt{2\pi}} \sum_{j=-\infty}^{\infty} e^{-ij\xi} u_j^n + \frac{\lambda}{\sqrt{2\pi}} \sum_{j=-\infty}^{\infty} e^{-ij\xi} u_{j+1}^n.
 \end{aligned} \tag{14.24}$$

By making a change of variables we can easily prove that:

$$\frac{1}{\sqrt{2\pi}} \sum_{j=-\infty}^{\infty} e^{-ij\xi} u_{j\pm 1}^n = \frac{e^{\pm i\xi}}{\sqrt{2\pi}} \sum_{m=-\infty}^{\infty} e^{-im\xi} u_m^n = e^{\pm i\xi} \hat{u}^n(\xi) \quad (m = j \pm 1).$$

Using this result in (14.24) we see that after having done some arithmetic:

$$\begin{aligned}
 \hat{u}^{n+1}(\xi) &= \lambda e^{-i\xi} \hat{u}^n(\xi) + (1-2\lambda) \hat{u}^n(\xi) + \lambda e^{i\xi} \hat{u}^n(\xi) \\
 &= \{\lambda e^{-i\xi} + (1-2\lambda) + \lambda e^{i\xi}\} \hat{u}^n(\xi) = \{2\lambda \cos \xi + (1-2\lambda)\} \hat{u}^n(\xi) \\
 &= (1 - 4\lambda^2 \sin^2 \xi / 2) \hat{u}^n(\xi) \equiv \rho(\xi) \hat{u}^n(\xi).
 \end{aligned}$$

We thus eliminate the x dependency, and this greatly simplifies matters. Continuing, we define the *symbol* of the difference scheme (14.23) by:

$$\rho(\xi) = 1 - 4\lambda^2 \sin^2 \xi / 2.$$

Applying this formula $n+1$ times gives:

$$\hat{u}^{n+1}(\xi) = \rho(\xi)^{n+1} \hat{u}^0(\xi). \tag{14.25}$$

From Thomas (1995) the *symbol* or *amplification factor* needs to be less than one in absolute value, and some tedious but simple arithmetic shows that a sufficient condition is:

$$|\rho(\xi)| \leq 1 \text{ or } \lambda \leq 1/2.$$

This technique can be applied to more general finite difference schemes.

The symbol will either be always less than one in absolute value (in which case the difference scheme is said to be *unconditionally stable*) or it will be equal to one in absolute value depending on the mesh sizes (in which case the difference scheme is said to be *conditionally stable*). For example, the explicit Euler scheme (14.23) for the diffusion equation (14.14) is conditionally stable, because we must satisfy $\lambda = a^2 k / h^2 \leq 1/2$ or $k \leq \frac{h^2}{2a^2}$ where h and k are the step sizes in the space and time directions, respectively.

14.5.3 Discrete von Neumann Stability Criterion

The von Neumann stability analysis (also known as *Fourier stability analysis*) is a procedure to check the stability of finite difference schemes when applied to linear partial differential equations. The analysis is based on the Fourier decomposition of numerical error. The analysis rests on representing the approximate solution in the form:

$$u_j^n = \gamma^n e^{ij\alpha h}, \quad i = \sqrt{-1}, \quad (\alpha \text{ is a number}) \quad (14.26)$$

where γ is called the *symbol* or *amplification factor* of the difference scheme in question. In general, we plug this formula into the difference scheme and solve for γ . For stability to hold, we must have $|\gamma| \leq 1$. Let us take an example as in scheme (14.23). The steps are:

$$\begin{aligned} u_j^{n+1} &= \gamma^{n+1} e^{ij\alpha h} \\ &= \lambda u_{j-1}^n + (1 - 2\lambda)u_j^n + \lambda u_{j+1}^n \\ &= \dots \\ &= \gamma^n e^{ij\alpha h}(\lambda e^{-i\alpha h} + (1 - 2\lambda) + \lambda e^{i\alpha h}). \end{aligned} \quad (14.27)$$

Then the symbol is $\gamma = 1 - 2\lambda \sin^2 \frac{\alpha h}{2}$ and then $|\gamma| \leq 1$ if we wish to have stability. From (14.27) we see that von Neumann stability is equivalent to the steps in Section 14.5.2.

In general, the von Neumann criterion is a necessary condition for stability, and the approach is, strictly speaking, only applicable to initial value problems with constant coefficients in an infinite domain. It was not built to cater for initial boundary value problems (IBVPs). (We discuss stability of schemes for IBVPs in Chapter 15.) Nonetheless, the method gives necessary conditions for stability.

14.5.4 Some More Examples

We consider the heat equation for convenience. Some arithmetic shows that its symbol for Crank–Nicolson is given by:

$$\rho(\xi) = \frac{1 - 2\lambda \sin^2 \xi/2}{1 + 2\lambda \sin^2 \xi/2}. \quad (14.28)$$

The implicit Euler scheme has the symbol:

$$\lambda(\xi) = \frac{1}{1 + 4\lambda \sin^2 \xi/2}. \quad (14.29)$$

Both symbols have absolute value less than one, and we conclude that the schemes are unconditionally stable.

As a counterexample, we give an example of a scheme that is not stable. Consider the PDE:

$$\frac{\partial u}{\partial t} - \frac{\partial u}{\partial x} = 0. \quad (14.30)$$

This is a *wave equation* with the wave travelling in the negative x direction with speed equal to one. We propose upwinding in space, explicit in time scheme:

$$\frac{u_j^{n+1} - u_j^n}{k} - \frac{u_j^n - u_{j-1}^n}{h} = 0. \quad (14.31)$$

Again, some arithmetic shows that:

$$\begin{aligned} \rho(\xi) &= 1 + \lambda - \lambda e^{-i\xi}, \quad \lambda = \frac{k}{h} \\ |\rho(\xi)| &\leq 1 \text{ never satisfied.} \end{aligned} \quad (14.32)$$

Thus (14.31) is an unconditionally unstable scheme! This is because the finite difference scheme is modelling the flow in the ‘wrong direction’ as it were! An interesting exercise is to use (correct) downwinding instead and then compute the symbol.

Finally, we consider the *convection-diffusion equation* with constant coefficients:

$$\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2} \quad (14.33)$$

with corresponding difference scheme (centred differences in x , explicit in time):

$$\frac{u_j^{n+1} - u_j^n}{k} + a \frac{u_{j+1}^n - u_{j-1}^n}{2h} = \nu D_+ D_- u_j^n.$$

The symbol is given by:

$$\rho(\xi) = (1 - 2\lambda) + 2\lambda \cos \xi - iR \sin \xi$$

and $\lambda = \nu k / h^2$ and $R = ak/h$.

Again, some long-winded arithmetic shows that the symbol has absolute value less than one if:

$$\frac{R^2}{2} \leq \lambda \leq 1/2.$$

See, for example, Thomas (1995).

Finally, Equation (14.33) serves as a very good model problem to study for a number of reasons:

- It is a convection-diffusion equation, and it is a component of the one-factor and multi-factor Black–Scholes equations.

- It is a good model to use as input to finite difference methods. Many undergraduate and graduate textbooks avoid this model, with the consequence that they miss certain key properties such as convection-dominance, upwinding and monotone schemes. (See Duffy (1980), Duffy (2006) and Duffy (2018) for detailed treatments of these problems.)
- Splitting and ADI methods consist essentially of sequential application of methods for one-dimensional convection-diffusion problems applied to two-dimensional problems. In general terms, we construct schemes in higher dimensions from lower-dimensional building blocks. This promotes reusability and reliability of algorithms and code.
- Model (14.33) with Dirichlet boundary conditions is a good mid-sized problem to implement by novices (and even experienced developers) before tackling the Black–Scholes equation. For example, you could also integrate exponential fitting into the schemes to resolve convection dominance problems.

14.6 THEORETICAL CONSIDERATIONS

We now discuss what it means for the solution of a finite difference equation to converge to the solution of a partial differential equation that it approximates. The most popular (but not necessarily the most general) approach is based on the *Lax Equivalence Theorem* that allows us to prove convergence of a scheme by proving that it is consistent and stable.

In Chapter 15 we continue with a stability analysis based on the discrete analogue of the maximum principle, M-matrices and monotone finite difference schemes.

14.6.1 Consistency

Let us consider the initial value problem:

$$(a) \frac{\partial u}{\partial t} + Lu = F, \quad -\infty < x < \infty, \quad t > 0 \\ (b) u(x, 0) = f(x), \quad -\infty < x < \infty \tag{14.34}$$

where L is a differential operator and consider some finite difference scheme to approximate (14.34):

$$(a) L_h^k u_j^n = G_j^n \\ (b) u_j^0 = f(x_j), \quad j = \dots, -2, -1, 0, 1, 2, \dots, n > 0 \tag{14.35}$$

where G_j^n is some approximation to $F(x_j, t_n)$ and L_h^k is a discrete approximation to the operator L .

Definition 14.1 The finite difference scheme (14.35)(a) is *pointwise consistent* with the partial differential equation in (14.34)(a) if for any function $v = v(x, t)$ the following relationship holds:

$$\left(\frac{\partial v}{\partial t} + Lv - F \right)_j^n - [L_h^k v(x_j, t_n) - G_j^n] \rightarrow 0 \quad (14.36)$$

as $h, k \rightarrow 0$ and $(x_j, t_{n+1}) \rightarrow (x, t)$.

This definition essentially tells us how well the differential equation approximates the finite difference scheme. We can write (14.36) in the equivalent form:

$$\left(\frac{\partial}{\partial t} + L - L_h^k \right) v(x_j, t_n) + G_j^n - F_j^n = 0. \quad (14.37)$$

Thus the scheme is consistent (or compatible) with the initial value problem if the terms in (14.37) approach zero as h and k tend to zero. The second term G_j^n represents approximates the source term F in Equation (14.37) and this tends to zero. It only remains to prove that the first term in (14.37) also tends to zero in general.

Let us take the example of scheme (14.23) that approximates the heat equation (notice that $F = 0$ in this case). Then for the scheme (14.23) we get:

$$\begin{aligned} \left(\frac{\partial}{\partial t} + L - L_h^k \right) u(x_j, t_n) &= \frac{\partial u(x_j, t_n)}{\partial t} - \frac{u(x_j, t_{n+1}) - u(x_j, t_n)}{k} \\ &\quad - a^2 \left(\frac{\partial^2 u(x_j, t_n)}{\partial x^2} - D_+ D_- u(x_j, t_n) \right). \end{aligned} \quad (14.38)$$

Here, scheme (14.23) is the explicit Euler approximation to the heat equation. The same analysis can be applied to other schemes.

Then, by applying Taylor's theorem around the point (x_j, t_n) with exact remainder, we can show that this term is bounded by:

$$M(h^2 + k)$$

where M depends on the derivatives of u with respect to x and t but is independent of k and h . Thus, scheme (14.23) is *consistent with* the heat equation.

14.6.2 Stability

We now investigate the concept of stability of finite difference schemes. For the moment, let us take a scheme whose inhomogeneous term is zero. We write a general *one-step scheme* for an initial value problem in the vector form:

$$\begin{aligned} u^{n+1} &= Qu^n, \quad n \geq 0 \\ u^n &= (\dots, u_{-1}^n, u_0^n, u_1^n, \dots)^\top \end{aligned} \quad (14.39)$$

where Q is an operator.

Definition 14.2 The difference scheme (14.39) is said to be *stable* with respect to the norm $\|\cdot\|$ if there exist positive constants k_0 and h_0 and two non-negative constants K and β such that:

$$\|u^{n+1}\| \leq Ke^{\beta t} \|u^n\|$$

for $0 \leq t = t_{n+1}$, $0 < h \leq h_0$ and $0 < k \leq k_0$.

We now generalise (14.39) to include an inhomogeneous term

$$\begin{aligned} u^{n+1} &= Qu^n + kG^n \\ G^n &\equiv (\dots, G_{-1}^n, G_0^n, G_1^n, \dots)^\top. \end{aligned} \tag{14.40}$$

Definition 14.3 The difference scheme (14.40) is *consistent* with the partial differential equation (14.34)(a) if the solution v of (14.34)(a) satisfies:

$$v^{n+1} = Qv^n + kG^n + k\tau^n \text{ and } \|\tau^n\| \rightarrow 0$$

as $h, k \rightarrow 0$ where v^n denotes the vector whose j^{th} component is $v(x_j, t_n)$, $j = 1, \dots, J$. (We use the symbol v to avoid confusion with the solution u of (14.40)).

Definition 14.4 The difference scheme (14.40) is said to be *accurate of order* (p, q) to the given partial differential equation if:

$$\|\tau^n\| = O(h^p) + O(k^q).$$

We refer to τ^n as the *truncation error*.

14.6.3 Convergence

We now discuss the fundamental relationship between consistency and stability.

Theorem 14.1 (The Lax Equivalence Theorem) A consistent two-level scheme of the form (14.40) for a well-posed linear initial value problem is convergent if and only if it is stable.

As long as we have a consistent scheme, convergence is synonymous with stability. In short, all we need to prove is that a scheme is consistent (use Taylor's theorem) and stable.

14.7 FIRST-ORDER PARTIAL DIFFERENTIAL EQUATIONS

There is a vast literature on first-order hyperbolic equations. Much effort has gone into devising robust approximate schemes in application areas such as gas and fluid dynamics, chemical reactor theory and wave phenomena. We consider first-order partial

differential equations in two independent variables x and t . The first variable x is typically space, and the second variable t usually represents time.

The first model problem is an initial value problem (IVP) on an infinite interval:

$$\begin{aligned} \frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} &= 0, \quad -\infty < x < \infty, \quad t > 0 \\ u(x, 0) &= f(x), \quad -\infty < x < \infty. \end{aligned} \tag{14.41}$$

In this equation the constant a can be positive or negative, and $f = f(x)$ is some given function that we call the initial condition. System (14.41) is a model for wave propagation in homogeneous media. For example, the solution $u(x, t)$ could represent the concentration of a reactant in a chemical process, and a is the linear velocity of the reactant mixture. Another example models problems related to multiphase flow in porous media in reservoir engineering (Peaceman (1977)). In this case $u(x, t)$ is the saturation variable, and a is the positive velocity and represents flow in the direction of increasing x .

We shall later discuss examples from financial engineering in which the variables x , t and u will take on specific roles. At this moment we view (14.41) from a generic perspective.

The second model problem is the *initial boundary value problem* (IBVP) defined as follows:

$$\begin{aligned} \frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} &= 0, \quad 0 < x < 1, \quad t > 0 \\ u(x, 0) &= f(x), \quad 0 \leq x \leq 1 \\ u(0, t) &= g(t), \quad t \geq 0. \end{aligned} \tag{14.42}$$

In this case we assume that $a > 0$ and that a boundary condition (BC) $g(t)$ is given when $x = 0$. This is the correct boundary condition because information is travelling from left to right. In the case when $a < 0$, the IBVP is formulated as follows:

$$\begin{aligned} \frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} &= 0, \quad 0 < x < 1, \quad t > 0 \\ u(x, 0) &= f(x), \quad 0 \leq x \leq 1 \\ u(1, t) &= g(t), \quad t \geq 0. \end{aligned} \tag{14.43}$$

The main difference between the IVP (14.41) and IBVP (14.42) or IBVP (14.43) is the presence of a boundary condition. This latter condition is needed in many situations. For example, the simplest form of heat exchanger consists of a tube immersed in a bath held at a constant temperature K . If the temperature of the fluid flowing through the tube is $u(x, t)$ at some point from the inlet at $x = 0$, then the IBVP for this case is given by:

$$\begin{aligned} \frac{\partial u}{\partial t} + V \frac{\partial u}{\partial x} &= H(K - u), \quad 0 < x < 1, \quad t > 0 \\ u(x, 0) &= f(x), \quad 0 \leq x \leq 1 \\ u(0, t) &= g(t), \quad t \geq 0 \end{aligned} \tag{14.44}$$

where V is the (positive) velocity of the fluid, H is some constant, $f(x)$ is the initial temperature distribution and $g(t)$ is the inlet boundary condition.

In general, for first-order IBVP we place the boundary condition at $x = 0$ when $a > 0$ (as in Equation (14.42)) or at $x = 1$ when $a < 0$ (as in Equation (14.43)).

We conclude this section with some examples from financial engineering. The first example is a very simple PDE for an Asian option (Ingersoll (1987), p. 377):

$$\begin{aligned} -\frac{\partial F}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 F}{\partial S^2} + rS \frac{\partial F}{\partial S} + S \frac{\partial F}{\partial A} - rF &= 0 \\ 0 < S < \infty, \quad 0 < A < \infty, \quad 0 < t < T \\ F(0, A, t) &= 0, \quad t \geq 0 \\ F(\infty, A, T) &= 1, \quad 0 \leq A < \infty \\ F(S, \infty, t) &= 0, \quad t \geq 0 \\ F(S, A, T) &= \max\left(S - \frac{A}{T}, 0\right) \end{aligned}$$

where the average A is defined by:

$$A = A(T) \equiv \int_0^T S(t)dt$$

and F represents the Asian option price.

We see that the equation in the A direction is first order (there is no diffusion term) and thus only one boundary condition (at most) needs to be given. We can convince ourselves that the condition at infinity is the right one (in fact, it is similar to Equation (14.43)). We note that these boundary conditions have been heuristically motivated. We formalise them in Chapter 24.

The second example is the Black–Scholes equation:

$$-\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r - D)S \frac{\partial V}{\partial S} - rV = 0 \quad (14.45)$$

We mention the *linearity boundary condition*:

$$\frac{\partial^2 V}{\partial S^2} = 0 \text{ when } S = S_{max}$$

where S_{max} is the position of the so-called *far field*.

In this case the Equation (14.45) at $S = 0$ degenerates into the *ordinary differential equation*

$$-\frac{dV}{dt} - rV = 0$$

and it is possible to solve this equation analytically.

The final example is concerned with the pricing of a *zero coupon bond* under a Cox-Ingersoll-Ross (CIR) interest-rate model. The pricing equation is given by the parabolic PDE:

$$-\frac{\partial B}{\partial t} + \frac{1}{2}\sigma^2 r \frac{\partial^2 B}{\partial r^2} + (a - br)\frac{\partial B}{\partial r} - rB = 0.$$

In this case, B is the bond price.

If we let the PDE ‘degenerate’ to $r = 0$ (when the *Feller condition* is satisfied), we get the following boundary condition:

$$-\frac{\partial B}{\partial t} + a\frac{\partial B}{\partial r} = 0.$$

Thus, on the boundary $r = 0$ we must solve a first-order hyperbolic equation that can be solved numerically, for example. We discuss the CIR model in detail in Chapter 25.

14.7.1 Why First-Order Equations are Different: Essential Difficulties

Hyperbolic partial differential equations model many kinds of phenomena in the real world: for example, aerodynamics, atmospheric flow, fluid flow in porous media and more. Hyperbolic equations can be more difficult to model than parabolic and elliptic equations. In particular, finding good schemes for non-linear systems of equations is a non-trivial task.

We first take a look at the model initial value problem (14.41). In this case we can conveniently ignore boundary conditions because there are none. The reader can check that the solution of (14.41) is given by:

$$u(x, t) = f(x - at), \quad -\infty < x < \infty, \quad t > 0.$$

Thus, we know what the solution is, and we also know that it is constant along the *characteristic curve* $x - at = \text{constant}$. The *family of characteristics* completely determines the solution at any point (x, t) . Furthermore, the form of the solution $u(x, t)$ is the same as that of $f(x)$, except that the form is translated to the right in the case $a > 0$ and to the left in the case $a < 0$.

A special property of the solution of (14.41) is that there is no *dissipation* in it. This means that the Fourier modes neither decay nor grow with time. A major challenge when designing finite difference schemes for hyperbolic equations is to design them to be stable while at the same time ensuring that they do not dampen out the solution.

Another challenge is to develop schemes that take the *speed of propagation* of the solution u into account. It is intuitively obvious that the numerical schemes should give good approximations to the speed of propagation of the wave forms from the analytic solution.

Finally, *dispersion* is concerned with how the numerical solution loses its form in time. A good discussion of these topics is given in Vichnevetsky and Bowles (1982).

As stated in Thomas (1999), *the solution will only be as smooth as the initial condition*. This is in contrast to parabolic equations, where the solution becomes smooth after a certain time even if the initial condition is discontinuous. A simple example is given by defining the initial condition:

$$f(x) = \begin{cases} 1 & \text{if } x \leq 0 \\ 0 & \text{if } x > 0. \end{cases}$$

Using the exact formula $u(x, t) = f(x - at)$ we see that the solution $u(x, t)$ will be discontinuous along the lines $x - at = 0$. The solution in this case is given by:

$$u(x, t) = \begin{cases} 1, & x \leq at \\ 0, & x > at. \end{cases}$$

We conclude that the solution cannot satisfy (14.41) in the classical sense, and in this case we must resort to finding a so-called *weak solution*. For a detailed discussion of this topic, see Thomas (1999). This topic is outside the scope of this book.

14.7.2 A Simple Explicit Scheme

In this section we introduce a simple finite difference scheme. To this end, we partition (x, t) space into a uniform rectangular mesh, and we define the constants h and k to be the mesh sizes in the x and t directions, respectively. In general, we employ one-step methods in the t direction, and we choose between one-sided and centred differencing in the x direction. We depict the mesh in Figure 14.1.

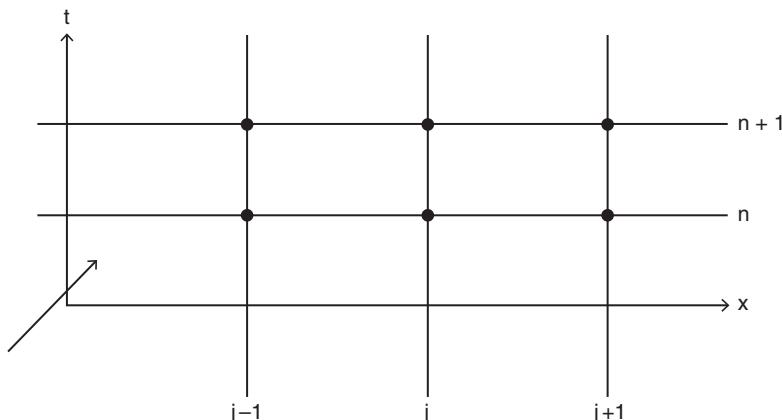


FIGURE 14.1 Mesh in (x, t) space.

We examine IVP (14.41) again.

The first scheme is called *Forward in Time, Backward in Space* (FTBS) is defined by:

$$\frac{u_j^{n+1} - u_j^n}{k} + a \frac{u_j^n - u_{j-1}^n}{h} = 0, \quad a > 0$$

or

$$u_j^{n+1} = (1 - \lambda)u_j^n + \lambda u_{j-1}^n, \quad n \geq 0, \quad \lambda \equiv \frac{ak}{h}. \quad (14.46)$$

Thus, the value at time level $n + 1$ is computed directly from the value at time level n . However, if the parameter λ is greater than 1, the solution may oscillate boundedly or unboundedly. This is a common problem with explicit difference schemes, and we say that scheme (14.46) is *conditionally stable*. This means that the inequality:

$$|\lambda| = \left| \frac{ak}{h} \right| \leq 1 \quad (14.47)$$

must hold if we wish to have a stable and hence convergent scheme. Inequality (14.47) is called the *Courant–Friedrichs–Lowy* (CFL) condition, in honour of the mathematicians who invented it, and it is one of the most famous inequalities in numerical analysis. It can be shown that the CFL condition is necessary for convergence of the discrete solution to the analytic solution. In fact, we can replicate the CFL inequality by applying the von Neumann stability analysis by examining Fourier modes:

$$u_j^n = \gamma^n e^{i\alpha j h}, \quad i = \sqrt{-1}. \quad (14.48)$$

Using this representation in the scheme (14.46) gives an expression for the *amplification factor* as follows:

$$\gamma = (1 - \lambda + \lambda \cos \alpha h) - i \lambda \sin \alpha h = 1 - \lambda(1 - \cos \alpha h) - i \lambda \sin \alpha h.$$

Under the constraint (14.47) we can prove that:

$$|\gamma| \leq 1.$$

When the coefficient a in Equation (14.41) is negative, we can use the following *Forward in Time, Forward in Space* (FTFS) scheme:

$$\frac{u_j^{n+1} - u_j^n}{k} + a \frac{u_{j+1}^n - u_j^n}{h} = 0, \quad a < 0$$

or

$$u_j^{n+1} = (1 + \lambda)u_j^n - \lambda u_{j+1}^n = u_j^n - \lambda(u_{j+1}^n - u_j^n). \quad (14.49)$$

Again, we can calculate the amplification factor as before, and it will be less than 1 in absolute value if the CFL inequality (14.47) holds. Thus, we must be careful when constructing good schemes; the sign of the coefficient a is important.

The scheme (14.46) uses so-called *backward differencing* (with $a > 0$), while the scheme (14.49) using *forward differencing* (with $a < 0$). Unstable schemes will result if we use backward differencing with $a < 0$ or forward differencing with $a > 0$. You can convince yourself of this fact by calculating the amplification factors for the schemes. This is also important when working with more complex PDEs, such as Asian-style option PDEs in Chapter 24.

14.7.3 Some Common Schemes for Initial Value Problems

We start with a FTCS (*Forward in Time, Centred in Space*) scheme, where the derivative with respect to x is taken around the mesh points $(j-1)h$ and $(j+1)h$, and we use explicit Euler in time:

$$\frac{u_j^{n+1} - u_j^n}{k} + a \frac{u_{j+1}^n - u_{j-1}^n}{2h} = 0 \quad (14.50)$$

Then:

$$\gamma(\xi) = 1 - i \lambda \sin \xi \left(\lambda = \frac{ak}{h} \right) \text{ and } |\gamma(\xi)|^2 \geq 1 \text{ always!}$$

This scheme is thus never stable for any value of the CFL number! We say that this scheme is unconditionally unstable. This is a pity, but the situation can be improved by adding a so-called *viscosity* term to (14.50) in order to stabilise it. The result is called the *Lax–Wendroff scheme* and is given by a second-order perturbation of scheme (14.50), namely:

$$u_j^{n+1} = u_j^n - \frac{\lambda}{2}(u_{j+1}^n - u_{j-1}^n) + \frac{\lambda^2}{2}(u_{j+1}^n - 2u_j^n + u_{j-1}^n)$$

and this scheme is stable if $|\lambda| \leq 1$.

Thus, Lax–Wendroff is a conditionally stable explicit scheme. We also discuss it in Chapter 24.

We now discuss some implicit schemes. The first scheme is *Backward in Time, Backward in Space* (BTBS) and is given by:

$$\frac{u_j^{n+1} - u_j^n}{k} + a \frac{u_j^{n+1} - u_{j-1}^{n+1}}{h} = 0, \quad a > 0$$

or

$$u_j^{n+1}(1 + \lambda) = u_j^n + \lambda u_{j-1}^{n+1}.$$

This scheme is always stable. The centred difference scheme is given by:

$$\frac{u_j^{n+1} - u_j^n}{k} + a \frac{u_{j+1}^{n+1} - u_{j-1}^{n+1}}{2h} = 0.$$

We can show that the amplification factor in this case is:

$$\gamma = \frac{1}{1 + \lambda i \sin \alpha h}, \quad |\gamma| < 1$$

and hence the scheme is unconditionally stable.

We conclude this section by applying the *Crank–Nicolson scheme* to (14.41). It is an implicit scheme and uses averaging in time and centred differences in x :

$$\frac{u_j^{n+1} - u_j^n}{k} + a \frac{u_{j+1}^{n,\frac{1}{2}} - u_{j-1}^{n,\frac{1}{2}}}{2h} = 0$$

where:

$$u_j^{n,\frac{1}{2}} \equiv \frac{1}{2}(u_j^{n+1} + u_j^n).$$

After some lengthy but simple arithmetic we see that the amplification factor is given by:

$$\gamma = \frac{1 - i\beta}{1 + i\beta} \text{ where } \beta = \frac{\lambda}{2} \sin \alpha h, \left(\lambda = \frac{ak}{h} \right) \text{ and hence } |\gamma| = 1.$$

The Crank–Nicolson scheme is called *neutrally stable* because the absolute value of its amplification factor is exactly equal to one! Any perturbation (for example, due to round-off errors) could make this value greater than one. The end-result is potential instability and Gibbs-type oscillation phenomena.

Figure 14.2 is a schematic diagram of the different kinds of schemes for IVP (14.41) based on Peaceman (1977). It shows the stability levels of the different kinds of finite difference schemes for (14.41). You can use this figure as a roadmap.

14.7.4 Some Other Schemes

We give some other examples of finite difference schemes for first-order hyperbolic partial differential equations. We can use them as components or ‘building blocks’ for schemes for the Black–Scholes equation.

We consider the *three-level* ($n - 1, n, n + 1$) *leapfrog scheme* defined as:

$$\frac{u_j^{n+1} - u_j^{n-1}}{2k} + a \frac{u_{j+1}^n - u_{j-1}^n}{2h} = 0. \quad (14.51)$$

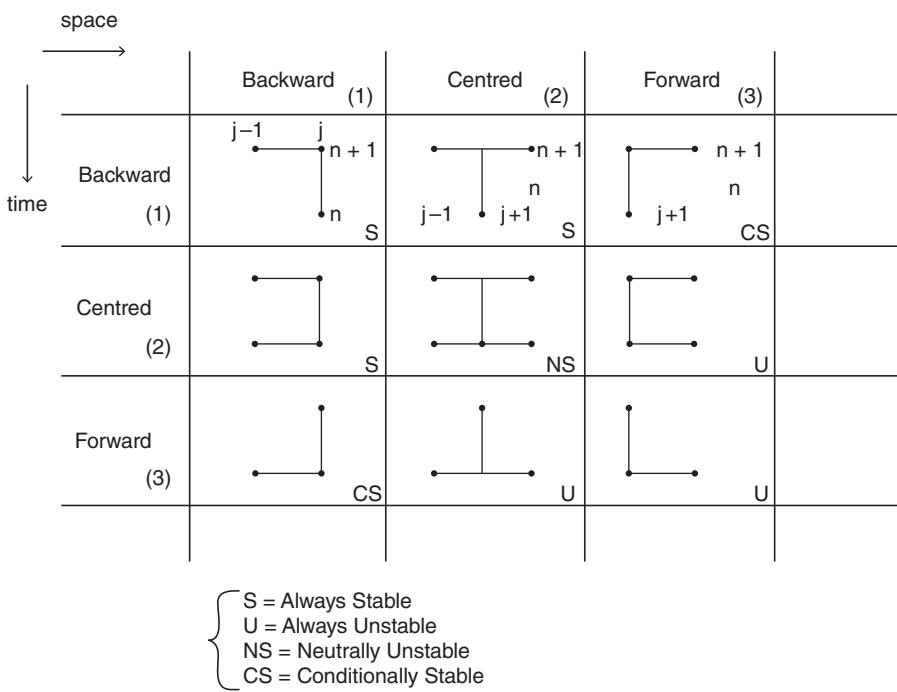


FIGURE 14.2 Special cases.

This is a second-order accurate scheme with respect to k and h , which makes the scheme appealing. However, it requires two initial values; we usually determine these values by a two-level scheme. The leapfrog scheme is stable if:

$$|a| \frac{k}{h} < 1. \quad (14.52)$$

Applying a von Neumann analysis to (14.51) shows that the leapfrog scheme is neutrally stable because the absolute value of its amplification factor is exactly one.

The following scheme is called the *Thomee or Box scheme*, and it gets its name from the fact that we take averages in the x and t directions in a box:

$$\frac{u_{j+1/2}^{n+1} - u_{j+1/2}^n}{k} + a \frac{u_{j+1}^{n+1/2} - u_j^{n+1/2}}{h} = 0$$

where

$$u_{j+\frac{1}{2}}^n \equiv \frac{1}{2}(u_{j+1}^n + u_j^n) \text{ and } u_j^{n+\frac{1}{2}} \equiv \frac{1}{2}(u_j^{n+1} + u_j^n).$$

This is a second-order scheme in k and h . What is its amplification factor?

14.7.5 General Linear Problems

The most general linear IBVP problem in this context is given by:

$$\begin{aligned}\frac{\partial u}{\partial t} + a(x, t) \frac{\partial u}{\partial x} &= R(x, t), \quad 0 < x < 1, \quad t > 0, \quad a(x, t) > 0 \\ u(x, 0) &= f(x), \quad 0 \leq x < 1 \\ u(0, t) &= g(t), \quad t \geq 0\end{aligned}$$

and the finite difference schemes in this chapter can easily be adapted to accommodate non-constant coefficients.

For example, the FTBS scheme generalisation of (14.46) applied to this problem is:

$$\begin{aligned}\frac{u_j^{n+1} - u_j^n}{k} + a(x_j, t_n) \frac{u_j^n - u_{j-1}^n}{h} &= R(x_j, t_n) \quad 1 \leq j \leq J, \quad n \geq 0 \\ u_j^0 &= f(x_j), \quad 1 \leq j \leq J-1 \\ u_0^n &= g(t_n), \quad n \geq 0.\end{aligned}$$

14.8 SUMMARY AND CONCLUSIONS

In this chapter we introduced and motivated the theoretical underpinnings of the finite difference method for one-dimensional initial value problems for partial differential equations with constant coefficients. The material is documented in many undergraduate textbooks on numerical methods, and studying it is recommended because it provides insights on understanding finite difference methods.

Mathematical and Numerical Foundations of the Finite Difference Method, Part II

Perhaps some day in the dim future it will be possible to advance the computations faster than the weather advances and at a cost less than the saving to mankind due to the information gained. But that is a dream.

Lewis Fry Richardson

15.1 INTRODUCTION AND OBJECTIVES

The main goal of this fundamental chapter is to analyse one-dimensional time-dependent convection-diffusion-reaction equations by numerical methods. We take a lifecycle approach from initial specification of the initial boundary value problem (IBVP) through choice of finite difference scheme. We focus on the following major topics:

- A1: Producing accurate and robust finite difference methods for convection-diffusion-reaction (CDR) PDEs.
- A2: Discrete maximum principle, stability analysis and monotone schemes.
- A3: Analysing IBVPs by semi-discretisation methods such as the Method of Lines (MOL). Stability analysis of semi-discrete schemes.
- A4: Defining convection-dominance and resolution of numerical difficulties using exponentially fitted schemes.
- A5: The role of matrices: their properties, eigenvalues and eigenvectors, solving matrix systems.
- A6: Padé rational approximant of the exponential.
- A7: Improving accuracy by Richardson extrapolation.

Other important topics that need to be discussed in later chapters are:

- A8: Mapping PDEs on infinite and semi-infinite intervals to bounded intervals.
- A9: Finite difference methods for adjoint and non-adjoint PDEs.

We discuss a number of new and less known results and methods. In particular, we discuss mathematically robust methods based on M-matrix theory to prove unconditional stability of monotone finite difference schemes, one of the holy grails of numerical analysis. Thus, in general we use the methods of Chapter 14 for simple initial value problems and for motivation, while the methods in this and succeeding chapters are more robust and extendible. Another important contribution in this chapter (more generally in this book) is that we introduce schemes to approximate PDEs containing both diffusion and convection terms, which is unique in the sense that most standard textbooks tend to focus on diffusion equations while other specialised textbooks for fluid and gas dynamics (Roache (1998), Tannehill, Anderson, and Pletcher (1997)) tend to focus more on convection (advection) problems. The interaction between the diffusion and convection terms leads to numerical phenomena that we need to be aware of.

This is a pivotal chapter for a number of reasons. First, it discusses a generic one-dimensional convection-diffusion-reaction equation and specifies it as an unambiguous initial boundary value problem. This model subsumes many of the PDEs that we encounter in finance. Second, we discuss a number of finite difference schemes for this class of problems, and we introduce methods to help us analyse the stability and accuracy of these schemes. Finally, we include a discussion of methods that we will use and generalise in later chapters. In a sense, we construct finite difference schemes for complex problems by decomposing them into simpler problems. This facilitates a flexible object-oriented design in C++. We discuss this issue in Duffy (2018).

15.2 A SHORT HISTORY OF NUMERICAL METHODS FOR CDR EQUATIONS

We introduce some terms and concrete examples to motivate the challenges that we encounter when approximating convection-diffusion-reaction PDEs, in particular knowing the subtle interaction between convection and diffusion. The role of mesh sizes and possible constraints between these quantities is crucial.

The issue of stability is more subtle than what first meets the eye. For example, Crank–Nicolson is *A-stable* but not *L-stable* (Lambert (1991)) and this fact went unnoticed or unmentioned in finance until the article Duffy (2004) was written. The Crank–Nicolson method in combination with centred differencing can produce oscillations at points of discontinuity (for example, the strike) of the payoff function. This phenomenon, along with its origins, is discussed in Lawson and Morris (1978). A good example for the heat equation can be found in Strang and Fix (1973), pp. 245–48. A short explanation of the root cause of the oscillation problem is that the high-frequency components $\lambda_j^h, j = 1, \dots, N$ (in contrast to the PDE case) do not decay at faster and faster rates. In the case of the Crank–Nicolson method, the amplification

factor (in fact, a (1,1) Padé rational approximant to the exponential function) is:

$$\mu_j^h = \frac{1 - \lambda_j^h \Delta t / 2}{1 + \lambda_j^h \Delta t / 2}, \quad |\mu_j^h| \leq 1.$$

As the approximate eigenvalues tend to infinity in absolute value, we see that the amplification factor tends to -1 and that the weights attached to the high frequencies change sign at each time step:

$$\lim \mu_j^h = -1 \text{ as } \lambda_j^h \rightarrow \infty.$$

On the other hand, the fully implicit (also known as the implicit Euler (0,1) Padé approximant to the exponential function) does not have this problem because:

$$\mu_j^h = \frac{1}{1 + \lambda_j^h \Delta t}$$

$$\lim \mu_j^h = 0 \text{ as } \lambda_j^h \rightarrow \infty.$$

In general, the Crank–Nicolson method is stable, but its amplification factor will become negative and start to increase in magnitude at the frequency $\lambda_j^h = 2/\Delta t$. The spurious oscillations that occur can be removed by taking $\Delta t < \frac{2h}{\pi}$ (Lawson and Morris (1978)).

The oscillation problems around the Crank–Nicolson method become more pronounced at the strike when we compute option sensitivities such as delta and gamma.

15.2.1 Temporal and Spatial Stability

In general, we use implicit methods (for example, Crank–Nicolson) in time to avoid temporal stability problems and we can avoid spatial stability problems by choosing an appropriate mesh size h in space (possibly at major computational cost) or by using exponentially fitted methods that are stable and convergent for any value of h and for any values of the diffusion and convection functions. We now discuss this problem for a time-dependent convection-diffusion equation.

We examine the stability of the *explicit finite difference method* (also known as *Forward in Time, Centred in Space (FTCS)*) applied to the model *convection-diffusion equation in non-conservative form*:

$$\frac{\partial u}{\partial t} = -\mu \frac{\partial u}{\partial x} + \sigma \frac{\partial^2 u}{\partial x^2} \tag{15.1}$$

where:

u = vorticity (or other advected (convected) quantity).

σ = diffusion = $\frac{1}{R_e}$, where R_e is the *Reynolds number*.

μ = linearised advection speed, $\mu > 0$.

The *FTCS* (explicit Euler) scheme now becomes:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -\mu \left(\frac{u_{j+1}^n - u_{j-1}^n}{2h} \right) + \sigma \left(\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2} \right) \quad (15.2)$$

or:

$$u_j^{n+1} = (\beta + \alpha) u_{j-1}^n + (1 - 2\beta) u_j^n + (\beta - \alpha) u_{j+1}^n$$

where:

$$\alpha = \frac{\mu \Delta t}{2h}, \quad \beta = \frac{\sigma \Delta t}{h^2}.$$

We see that this scheme is explicit in time and centred in space. We can examine this scheme from the viewpoint of the *maximum principle* that determines the conditions under which the solution at time level $n + 1$ is non-negative given that the solution is non-negative at time level n . We can *brainstorm* by considering the conditions under which all coefficients on the right-hand side of Equation (15.2) are non-negative and what happens when they become negative. To be precise, there are two forms of instability associated with the above *FTCS* (Roache (1998)) scheme:

- *Dynamic instability*: the scheme experiences oscillatory errors due to large step sizes in time. The condition for *no overshoot* is:

$$1 - 2\beta \geq 0 \Rightarrow \beta \leq \frac{1}{2}.$$

This inequality places a dependency relationship between the mesh sizes in the space and time dimensions.

- We resolve this problem by choosing not to use explicit schemes.
- *Static instability*: this form of instability is caused by the convection (advection) term in the PDE (15.1), especially when it is large compared to the volatility term:

$$\beta - \alpha \geq 0 \Rightarrow \frac{\mu h}{\sigma} \leq 2, \text{ or } h \leq \frac{2\sigma}{\mu}.$$

This is called the *convection dominance* phenomenon. This issue was also discussed in Chapter 6. We resolve this problem by *exponential fitting* or by upwinding. This phenomenon can also occur in time-independent problems.

We discuss an implicit scheme that does not produce dynamic instability. It is called the *Backward in Time, Centred in Space (BTCS)* scheme for PDE (15.1) defined by:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -\mu \left(\frac{u_{j+1}^{n+1} - u_{j-1}^{n+1}}{2h} \right) + \sigma \left(\frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{h^2} \right).$$

We rewrite this scheme in the following form:

$$u_j^{n+1} - u_j^n = -\alpha(u_{j+1}^{n+1} - u_{j-1}^{n+1}) + \beta(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}) \quad (15.3)$$

or:

$$-(\alpha + \beta)u_{j-1}^{n+1} + (1 + 2\beta)u_j^{n+1} + (\alpha - \beta)u_{j+1}^{n+1} = u_j^n.$$

This scheme still suffers from possible static stability problems ($\alpha - \beta$ can be positive), and we can resolve this problem by employing *upwinding schemes* which are one-sided difference schemes for the convection term:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -\mu \left(\frac{u_j^{n+1} - u_{j-1}^{n+1}}{h} \right) + \sigma \left(\frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{h^2} \right) \quad (\mu > 0) \quad (15.4)$$

or:

$$-(2\alpha + \beta)u_{j-1}^{n+1} + (1 + 2\alpha + 2\beta)u_j^{n+1} - \beta u_{j+1}^{n+1} = u_j^n.$$

The disadvantage of upwinding is that it is only first-order accurate and we must take into account that the coefficient of the convection term (for example, we can have $\mu < 0$) can be positive or negative and the correct kind of upwinding must be used in each case.

Another solution to static stability problems is to use the *exponential fitting method* as discussed in Duffy (1980) and Duffy (2006). In particular, we use a combination of von Neumann stability analysis, the maximum principle for PDEs and *M-matrix theory* to provide a general method to prove the stability for a range of finite difference schemes. In particular, it can be used to analyse the stability of schemes (15.3) and (15.4).

15.2.2 Motivating Exponential Fitting Methods

We now introduce a special class of schemes that prove to be useful in approximating the solution of the Black–Scholes PDE. In particular, these so-called *exponentially fitted schemes* are able to handle discontinuities (near a strike price and at barriers, for example). In general, a fitted scheme is similar to the standard centred difference scheme except that we introduce a new coefficient into the difference equation. In order to find this coefficient we argue as follows; consider the model IVP:

$$\begin{aligned} \frac{du}{dt} + au = 0, \quad a > 0 \text{ constant} \\ u(0) = A \\ \text{with solution } u(t) = Ae^{-at}. \end{aligned} \quad (15.5)$$

and we propose the fitted Crank–Nicolson scheme defined by:

$$\begin{aligned} \sigma \frac{U^{n+1} - U^n}{h} + a \frac{U^{n+1} + U^n}{2} = 0 \\ U^0 = A. \end{aligned} \quad (15.6)$$

We now demand that the solution of the discrete scheme (15.6) should equal the solution of (15.5) at the mesh points. This will determine what the value of σ should be, and a bit of arithmetic shows that (there is only one unknown in Equation (15.6)):

$$\sigma = \frac{ah}{2} \coth \frac{ah}{2} \quad (15.7)$$

where $\coth x = \frac{e^{2x}+1}{e^{2x}-1}$.

This is the famous fitting factor, and it has been known since the 1950s (de Allen and Southwell (1955)), elaborated upon by Soviet scientists (Il'in (1969)) and generalised to time-dependent convection-diffusion equations in Duffy (1980).

Based on the fitting factor defined in Equation (15.7), we propose the generalised finite difference scheme when the coefficient a in Equation (15.5) is variable $a = a(t)$ and non-zero right-hand side $f = f(t)$:

$$\begin{aligned} \sigma^n \frac{U^{n+1} - U^n}{h} + a^{n+\frac{1}{2}} \frac{U^{n+1} + U^n}{2} &= f^{n+1/2}, \quad n \geq 0 \\ U^0 &= u_0. \end{aligned} \quad (15.8)$$

$$\text{Then } \sigma^n \equiv \frac{a^{n+1/2}h}{2} \coth \frac{a^{n+1/2}h}{2}.$$

It is more usual to generate the fitting factor by approximating a second-order convection diffusion equation, but the above approach gives the same answer and is easier to understand. The two approaches are essentially equivalent in that they lead to the same result. We now discuss this second approach.

15.2.3 Eliminating Temporal and Spatial Stability Problems

In this section we examine a boundary value problem, and we attempt to fit the exponential terms (and these are the most difficult terms to approximate) by a specially designed finite difference scheme. To this end, let us take a homogeneous second-order ordinary differential equation with constant coefficients a and b :

$$\frac{d^2u}{dx^2} + a \frac{du}{dx} + bu = 0. \quad (15.9)$$

It is known that the general solution of (15.9) is a sum of exponentials whose coefficients are roots of the so-called *auxiliary equation*:

$$m^2 + am + b = 0 \quad (\text{roots } \alpha_1, \alpha_2).$$

Thus, depending on these roots, the general solution is given by one of the following equations:

1. Real Roots $\alpha_1 \neq \alpha_2$

$$u = c_1 e^{\alpha_1 x} + c_2 e^{\alpha_2 x}.$$

2. Real Roots $\alpha_1 = \alpha_2 = \alpha$

$$u = (c_1 + c_2 x) e^{\alpha x}. \quad (15.10)$$

3. Complex Roots $\alpha_1 = A + iB, \alpha_2 = A - iB$

$$u = e^{Ax} (c_1 \cos Bx + c_2 \sin Bx)$$

where c_1 and c_2 are (undetermined) constants.

Some authors have developed special finite difference schemes that closely approximate the general solution of (15.9) at mesh points. For example, Roscoe (1975) defines difference schemes that are in some sense the discrete analogues of the solutions in Equation (15.10), and the schemes produce accurate and oscillation-free approximations to one-dimensional and two-dimensional convection-diffusion equations. In fact, for the boundary value problem:

$$\begin{aligned} \frac{d^2u}{dx^2} - \epsilon \left(\frac{1}{2} - x \right) \frac{du}{dx} &= 0, \quad 0 < x < 1 \\ u(0) = 0, \quad u(1) = 1 \end{aligned} \quad (15.11)$$

with exact solution given by :

$$u(x) = \frac{\int_0^x e^{\frac{1}{2}\epsilon y(1-y)} dy}{\int_0^1 e^{\frac{1}{2}\epsilon y(1-y)} dy}$$

the standard schemes such as upwinding, downwinding and centred differencing give terrible answers at $x = 0.5$ for certain large values of ϵ (the point $x = 0.5$ is called a *turning point*). In fact, the solution exhibits *spurious oscillations* at these points, while the so-called Unified Difference Representation (UDR) in Roscoe (1975) does not suffer from these schemes. The scheme is given by:

$$\begin{aligned} U_{j+1} - (1 + e^{w(x_j)}) U_j + e^{w(x_j)} U_{j-1} &= 0 \\ w(x) &\equiv \epsilon \left(\frac{1}{2} - x \right). \end{aligned}$$

The reason why standard difference schemes are not good is that the convection term $w(x)$ changes sign at $x = 0.5$ and for this reason we call (15.11) a *turning-point problem*. This kind of equation can occur in financial applications when the drift term changes sign.

We now introduce another fitting difference scheme (based on Il'in (1969)) which is the foundation for a number of schemes for the Black–Scholes equation. To this end, we consider the second-order equation:

$$\sigma \frac{d^2u}{dx^2} + \mu \frac{du}{dx} = 0 \quad (15.12)$$

where σ and μ are constants.

We now define the so-called fitted centred difference equation:

$$\rho D_+ D_- U_j + \mu D_0 U_j = 0, \quad 1 \leq j \leq J-1 \quad (15.13)$$

where the fitting factor ρ is chosen in such a way that the discrete and exact solutions have the same values at mesh points. If we insert a solution of (15.12) (that is, $u(x) = e^{-(\frac{\mu}{\sigma})x}$) into Equation (15.13), we can convince ourselves by algebra that:

$$\rho \equiv \frac{\mu h}{2} \coth \frac{\mu h}{2\sigma}.$$

This scheme is a faithful representation of the exact solution. For example, let us suppose that the coefficient σ tends to zero. Then by using the limits:

$$\lim_{\sigma \rightarrow 0} \frac{\mu h}{2} \coth \frac{\mu h}{2\sigma} = \begin{cases} +\mu h/2, & \mu > 0 \\ -\mu h/2, & \mu < 0 \end{cases}$$

$$\lim_{\mu \rightarrow 0} \frac{\mu h}{2} \coth \frac{\mu h}{2\sigma} = 1$$

we then see that the ‘reduced’ difference schemes are:

$$\frac{\mu}{h}(U_{j+1} - U_j) = 0, \quad \mu > 0$$

$$\frac{\mu}{h}(U_j - U_{j-1}) = 0, \quad \mu < 0.$$

We thus see that we get the correct upwinding or downwinding depending on the sign of μ . Many standard schemes have to be explicitly modified in order to get the correct upwinding or downwinding.

What happens next? We usually have to solve boundary value problems with non-constant coefficients, as in the following case with Dirichlet boundary conditions:

$$\sigma(x) \frac{d^2 u}{dx^2} + \mu(x) \frac{du}{dx} + b(x)u = f(x), \quad a < x < b$$

$$u(a) = \alpha, \quad u(b) = \beta. \quad (15.14)$$

We now approximate the solution of (15.14) by the generalisation of the scheme (15.13), namely

$$\rho_j D_+ D_- U_j + \mu_j D_0 U_j + b_j U_j = f_j, \quad 1 \leq j \leq J-1$$

$$U_0 = \alpha, \quad U_J = \beta \quad (15.15)$$

where:

$$\rho_j \equiv \frac{\mu_j h}{2} \coth \frac{\mu_j h}{2\sigma_j}, \quad \sigma_j \equiv \sigma(x_j), \quad \mu_j \equiv \mu(x_j), \quad f_j \equiv f(x_j).$$

Theorem 15.1 (Convergence) Let u and U be the solutions of (15.14) and (15.15), respectively. Then:

$$|u(x_j) - U_j| \leq Mh, \quad j = 0, \dots, J$$

where the constant M is independent of h, μ and σ .

We say that the scheme (15.15) is *uniformly convergent* irrespective of the relative sizes of the coefficients μ and σ . In order to improve the accuracy of the scheme we can use *Richardson extrapolation*. We take two approximate solutions on mesh sizes h and $h/2$:

$$U_j \equiv U_j^h = u(x_j) + A_1 h + A_2 h^2 + \dots$$

$$U_{2j} \equiv U_{2j}^{h/2} = u(x_j) + A_1 \frac{h}{2} + A_2 \frac{h^2}{4} + \dots$$

Then the discrete scheme defined by:

$$V_{2j}^{h/2} \equiv 2U_{2j}^{h/2} - U_j^h = u(x_j) + B_2 h^2, \quad j = 0, \dots, J \quad (15.16)$$

is a second-order approximation to the solution of (15.14). This estimate is borne out in theory and in numerical experiments. So we calculate the solution on two consecutive meshes and use the extrapolated scheme (15.16).

15.3 EXPONENTIAL FITTING AND TIME-DEPENDENT CONVECTION-DIFFUSION

We now come to the central theme of this chapter. We examine an initial boundary value problem with Dirichlet boundary conditions for the one-factor Black–Scholes, written in generic form:

$$\begin{aligned} Lu &\equiv -\frac{\partial u}{\partial t} + \sigma(x, t) \frac{\partial^2 u}{\partial x^2} + \mu(x, t) \frac{\partial u}{\partial x} + b(x, t)u = f(x, t) \text{ in } D \\ u(x, 0) &= \varphi(x), \quad x \in \Omega \\ u(A, t) &= g_0(t), \quad u(B, t) = g_1(t), \quad t \in (0, T) \end{aligned} \quad (15.17)$$

where $\Omega = (A, B)$ and $D = \Omega \times (0, T)$.

We introduce and apply exponentially fitted schemes to the problem (15.17), and we discuss the stability and convergence properties using the discrete maximum principle.

We partition the space and time intervals as follows:

$$A = x_0 < x_1 < \dots < x_J = B \quad (h = x_j - x_{j-1}) \quad j = 1, \dots, J$$

$$0 = t_0 < t_1 < \dots < t_N = T \quad (k = T/N).$$

We also approximate derivatives by divided differences, and to this end we define the following discrete operators:

$$L_k^h U_j^n \equiv -\frac{U_j^{n+1} - U_j^n}{k} + \rho_j^{n+1} D_+ D_- U_j^{n+1} + \mu_j^{n+1} D_0 U_j^{n+1} + b_j^{n+1} U_j^{n+1}. \quad (15.18)$$

Here we use the notation:

$$\varphi_j^{n+1} = \varphi(x_j, t_{n+1}) \text{ for any general function } \varphi = \varphi(x, t)$$

and a similar notation for the other coefficients. Furthermore:

$$\rho_j^{n+1} \equiv \frac{\mu_j^{n+1} h}{2} \coth \frac{\mu_j^{n+1} h}{2\sigma_j^{n+1}}.$$

We are now in a position to specify the *exponentially fitted scheme*:

$$\begin{aligned} L_k^h U_j^n &= f_j^{n+1}, \quad j = 1, \dots, J-1, \quad n = 0, \dots, N-1 \\ U_0^n &= g_0(t_n), \quad U_J^n = g_1(t_n), \quad n = 0, \dots, N \\ U_j^0 &= \varphi(x_j), \quad j = 1, \dots, J-1. \end{aligned} \quad (15.19)$$

What is going on here? In the x direction we use II'in fitting, while in the time direction we use the implicit Euler method. As we shall see later, the method is first-order accurate in both k and h . The difference between (15.19) and traditional finite difference schemes is the presence of the fitting factor.

In general, the fitted scheme combines fitting in space and implicit Euler in time. It is possible to use Crank–Nicolson in time to get second-order accuracy. Alternatively, we can use Richardson extrapolation for implicit Euler in time to achieve second-order accuracy in time.

15.4 STABILITY AND CONVERGENCE ANALYSIS

In this section we examine the scheme (15.19) from a numerical analysis viewpoint. In particular, we ask the questions:

- Does the scheme always produce realistic output from input?
- Is the solution bounded by the input?
- How close is the approximate solution to the exact solution?
- How does the scheme (15.19) perform compared to the Crank–Nicolson method?

The first result states that positive input data leads to a positive solution at all space and time.

Lemma 15.1 *Let the discrete function w_j^n satisfy $L_k^h w_j^n \leq 0$ (where L_k^h is defined by (15.18)) in the interior of the mesh with $w_j^n \geq 0$ on the boundary Γ . Then:*

$$w_j^n \geq 0, \quad \forall j = 0, \dots, J, \quad n = 0, \dots, N.$$

The next result gives an estimate for the growth of the solution of (15.19) in terms of its input data.

Lemma 15.2 (Uniform Stability) *Let $\{U_j^n\}$ be the solution of scheme (15.19) and suppose that:*

$$\begin{aligned} \max_j |U_j^n| &\leq \text{for all } j \text{ and } n \\ \max_j |f_j^n| &\leq N \text{ for all } j \text{ and } n \end{aligned}$$

Then:

$$\max_j |U_j^n| \leq -\frac{N}{\beta} + m \text{ in } \bar{Q} \text{ where } b(x, t) \leq \beta < 0.$$

We have thus proved stability by application of the discrete maximum principle in combination with a discrete comparison function. See Section (10.6) (Chapter 10) for the continuous analogue. The result is general and is valid for problems with non-constant coefficients, discontinuous coefficients as well as Neumann and Robin boundary conditions.

The following result tells us how accurate our exponentially fitted scheme is (we state the essential conclusions); see Duffy (1980).

Theorem 15.2 *Let u and U_j^n be the solution of (15.17) and (15.19), respectively. Then $|u(x_j, t_n) - U_j^n| \leq M(h + k)$ where M is independent of h, k, σ and μ .*

We say that the scheme (15.19) is *uniformly convergent* because the accuracy does not depend on the relative sizes of the coefficients σ and μ in the original problem.

Some schemes are:

Implicit Euler scheme (no fitting):

$$L_k^h U_j^n = -\frac{U_j^{n+1} - U_j^n}{k} + \sigma_j^{n+1} D_+ D_- U_j^{n+1} + \mu_j^{n+1} D_0 U_j^{n+1} + b_j^{n+1} U_j^{n+1}.$$

Crank–Nicolson (no fitting):

$$L_k^h U_j^n = -\frac{U_j^{n+1} - U_j^n}{k} + \sigma_j^{n+1/2} D_+ D_- U_j^{n+1/2} + \mu_j^{n+1/2} D_0 U_j^{n+1/2} + b_j^{n+1/2} U_j^{n+1/2}$$

where :

$$\sigma_j^{n+1/2} = \sigma(x_j, t_{n+1/2})$$

$$\mu_j^{n+1/2} = \mu(x_j, t_{n+1/2})$$

$$b_j^{n+1/2} = b(x_j, t_{n+1/2})$$

$$U_j^{n+1/2} \equiv \frac{1}{2}(U_j^{n+1} + U_j^n).$$

Fitted Crank–Nicolson:

$$L_k^h U_j^n = -\frac{U_j^{n+1} - U_j^n}{k} + \rho_j^{n+1/2} D_+ D_- U_j^{n+1/2} + \mu_j^{n+1/2} D_0 U_j^{n+1/2} + b_j^{n+1/2} U_j^{n+1/2}.$$

15.5 SPECIAL LIMITING CASES

In some applications the coefficient $\sigma(x, t)$ can become very small, in which case we essentially have a first-order hyperbolic equation. The question now is: What happens if we let $\sigma(x, t)$ tend to zero? Will we get a stable scheme that is the same or similar to an upwinding or downwinding scheme? To answer this question, we use the limits. We then get the difference schemes:

$$\mu > 0, \quad -\frac{U_j^{n+1} - U_j^n}{k} + \mu_j^{n+1} \frac{(U_{j+1}^{n+1} - U_j^{n+1})}{h} + b_j^{n+1} U_j^{n+1} = f_j^{n+1}$$

$$\mu < 0, \quad -\frac{U_j^{n+1} - U_j^n}{k} + \mu_j^{n+1} \frac{(U_j^{n+1} - U_{j-1}^{n+1})}{h} + b_j^{n+1} U_j^{n+1} = f_j^{n+1}.$$

These are the standard upwind or downwind schemes. Thus, the fitting scheme degenerates into a stable upwinding/downwinding scheme for a first-order hyperbolic partial differential equation. This is reassuring news. The scheme automatically switches to the correct reduced scheme. From a programming perspective, no hard-coded ‘if-else’ statements are needed when implementing this scheme.

15.6 STABILITY FOR INITIAL BOUNDARY VALUE PROBLEMS

We now consider problems on a bounded interval, namely initial boundary value problems. In general, schemes that are unstable for an IVP will probably also be unstable for the corresponding initial boundary value problem. In order to keep things concrete

for the moment, we examine the following initial boundary value problem for the heat equation:

$$\begin{aligned}\frac{\partial u}{\partial t} &= a^2 \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < 1, \quad t > 0 \\ u(x, 0) &= f(x), \quad 0 \leq x \leq 1 \\ u(0, t) &= g(t), \quad u(1, t) = h(t), \quad t > 0\end{aligned}\tag{15.20}$$

along with the compatibility conditions:

$$f(0) = g(0), \quad f(1) = h(0).$$

We then propose the Crank–Nicolson scheme where k is the mesh size in time:

$$\begin{aligned}u_j^{n+1} - \frac{k}{2} D_+ D_- u^{n+1} &= u_j^n + \frac{k}{2} D_+ D_- u_j^n, \quad j = 1, \dots, J-1 \\ u_j^0 &= f(x_j), \quad j = 1, \dots, J-1 \\ u_0^{n+1} &= g(t_{n+1}), \quad u_j^{n+1} = h(t_{n+1}), \quad n \geq 0.\end{aligned}\tag{15.21}$$

Assembling the information in Equation (15.21), we can write (15.21) in the equivalent matrix form:

$$M u^{n+1} = Q u^n, \quad n \geq 0\tag{15.22}$$

where:

$$u^n = (u_1^n, \dots, u_{J-1}^n)^\top.$$

The question concerning system (15.22) is now: Does it have a solution and is it stable? The following discussion attempts to answer this question in general and then we can apply the results for the specific problem (15.22). To this end, we introduce a useful technique from matrix algebra.

15.6.1 Gershgorin's Circle Theorem

In the following chapters we shall develop finite difference schemes for one-factor and multi-factor Black–Scholes equations. To this end, it is important to determine if a solution of the resulting matrix system exists and that it is unique.

In the following discussions we assume that A is a complex square matrix that is, one with n rows and n columns:

$$A = (a_{ij}) \quad i, j = 1, \dots, n.$$

Definition 15.1 Let the matrix A have eigenvalues λ_j , $j = 1, \dots, n$. Then:

$$\rho(A) \equiv \max_{j=1, \dots, n} |\lambda_j|$$

is called the *spectral radius* of A .

Definition 15.2 The quantity:

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

is called the *spectral norm* of the matrix A where x is a vector.

It can be shown (Varga (1962)) that:

$$\|A\| \geq \rho(A)$$

which gives the relationship between the spectral norm and spectral radius of a matrix.

We define the quantity:

$$\Lambda_i \equiv \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}| \text{ for } 1 \leq i \leq n.$$

The following theorem describes the distribution of the eigenvalues of a matrix.

Theorem 15.3 (*Gerschgorin (1931)*): *The eigenvalues of the matrix A lie in the union of the disks:*

$$|z - a_{ii}| \leq \Lambda_i, \quad 1 \leq i \leq n.$$

Corollary 15.1 *If A is a square matrix and:*

$$\nu \equiv \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}| \text{ then } \rho(A) \leq \nu.$$

Thus, the maximum of the row sums of the moduli of the entries of the matrix A give a simple upper bound for the spectral radius of the matrix A .

Let us take an example for the scheme (15.21)–(15.22). Let:

$$M = \begin{pmatrix} 1+r & -r/2 & & 0 \\ -r/2 & \ddots & \ddots & \\ & \ddots & \ddots & -r/2 \\ 0 & & -r/2 & 1+r \end{pmatrix} \quad (15.23)$$

and define:

$$Q = \begin{pmatrix} 1-r & r/2 & & 0 \\ r/2 & \ddots & \ddots & \\ & \ddots & \ddots & r/2 \\ 0 & & r/2 & 1-r \end{pmatrix}$$

where $r = \frac{a^2 k}{h^2}$.

Then:

$$\Lambda_1 = \Lambda_n = r/2$$

$$\Lambda_j = r, \quad j = 2, \dots, n-1.$$

We then get:

$$|z - (1 + r)| \leq r/2 \text{ and } |z - (1 + r)| \leq r.$$

If the first inequality is satisfied, then the second inequality is also satisfied.

But then we get:

$$1 \leq z \leq 1 + 2r.$$

We thus see that the eigenvalues of M are always greater than or equal to one. This implies that the eigenvalues of its inverse are always less than or equal to one.

Definition 15.3 A *Toeplitz matrix* is a band matrix in which each diagonal consists of identical elements, although different diagonals may contain different elements.

We are particularly interested in *tridiagonal Toeplitz* matrices. Then the eigenvalues are known (Thomas (1999), Bronson (1989)), namely:

$$\begin{pmatrix} b & c & & 0 \\ a & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & c \\ 0 & & a & b \end{pmatrix}$$

where the eigenvalues are defined by:

$$\lambda_j = b + 2\sqrt{ac} \cos \frac{j\pi}{n+1}, \quad j = 1, 2, \dots, n. \quad (15.24)$$

This is a useful formula because some difference schemes lead to matrices whose eigenvalues are not real but complex. Oscillatory solutions will appear in such cases. This is the static instability problem from Section 15.2.

This is a well-known constraint, and the conclusion is: standard difference schemes can have matrices with complex eigenvalues. This situation occurs if a and c have different signs. Oscillations can occur if the mesh size is not chosen to be small enough.

15.7 SEMI-DISCRETISATION FOR CONVECTION-DIFFUSION PROBLEMS

In this section we investigate the *semi-discretisation* of the one-dimensional convection-diffusion equation (which includes the Black–Scholes as a special case):

$$-\frac{\partial u}{\partial t} + \sigma(x) \frac{\partial^2 u}{\partial x^2} + \mu(x) \frac{\partial u}{\partial x} + c(x)u = f(x) \quad (15.25)$$

where:

$$(\sigma(x) > 0, \mu(x) > 0, c(x) \leq 0).$$

We discretise in the space direction only, and we concentrate on the centred difference and the fitting methods. The semi-discrete scheme is:

$$-\frac{du_j}{dt} + \tilde{\sigma}_j D_+ D_- u_j + \mu_j D_0 u_j + c_j u_j = f_j, \quad 1 \leq j \leq J - 1$$

where:

$$\tilde{\sigma}_j = \begin{cases} \sigma(x_j) \equiv \sigma_j \text{ for standard centred difference scheme} \\ \frac{\mu_j h}{2} \coth \frac{\mu_j h}{2\sigma_j} \text{ for fitted scheme} \end{cases} \quad (15.26)$$

and $\mu_j = \mu(x_j)$, $c_j = c(x_j)$, $f_j = f(x_j)$ $1 \leq j \leq J - 1$. In more general cases, these functions depend on both x and t .

We can write this equation as a vector system as follows:

$$-\frac{dU}{dt} + AU = F$$

$$U(0) = U_0$$

where:

$$U = (U_1, \dots, U_{J-1})^\top$$

$$A = \begin{pmatrix} & \ddots & & 0 \\ \ddots & & C_j & \\ & B_j & & \ddots \\ & A_j & & \ddots \\ 0 & & \ddots & \end{pmatrix}$$

and:

$$\begin{aligned} A_j &= \frac{\tilde{\sigma}_j}{h^2} - \frac{\mu_j}{2h} \\ B_j &= \frac{-2\tilde{\sigma}_j}{h^2} + c_j \\ C_j &= \frac{\tilde{\sigma}_j}{h^2} + \frac{\mu_j}{2h}. \end{aligned} \quad (15.27)$$

We investigate the matrix A because this determines the behaviour of the solution of system (15.27) to a large extent. We take a special example where all the coefficients in (15.27) are constants. Then A is a Toeplitz matrix whose eigenvalues are given by Equation (15.24). You can check that the eigenvalues are:

$$\lambda_j = (-2\alpha + c) + 2\sqrt{\alpha^2 - \beta^2} \cos \frac{j\pi}{J}, \quad j = 1, \dots, J-1 \quad (15.28)$$

where $\alpha \equiv \frac{\tilde{\sigma}}{h^2}$, $\beta \equiv \frac{\mu}{2h}$.

A bit of arithmetic shows that the eigenvalues of A are real and non-positive for any range of values of the parameters for the fitted scheme. In this case we always have $\alpha > \beta$. For the non-fitted centred difference scheme, we have a different story. In this case the eigenvalue will be real if:

$$\begin{aligned} A \geq 0 &\iff \frac{\sigma}{h^2} - \frac{\mu}{2h} \geq 0 \\ &\iff h \leq \frac{2\sigma}{\mu} \end{aligned}$$

which is the same result as in Section 15.2.1.

15.7.1 Essentially Positive Matrices

We now give a mathematical discussion of the properties of the general initial value problem:

$$\begin{aligned} -\frac{dU}{dt} + AU &= F \\ U(0) &= U_0. \end{aligned} \quad (15.29)$$

We assume for convenience that the matrix A and the vector F are independent of time. We are interested in two aspects of this problem:

- Behaviour of $U(t)$ for large time behavior.
- Numerical approximation of system (15.29) and more generally, system (15.30).

In this section we discuss the first problem based on the results in Varga (1962).

We now study the stability of the system (15.29) as a function of the right-hand terms and initial condition. To this end, we examine the properties of the matrix A . We say that A is *irreducible* if its directed graph is strongly connected. An equivalent statement is that A has non-vanishing off-diagonal elements. We say that A is an M matrix (with $a_{ij} \leq 0 \forall i \neq j$) if A is non-singular and a sufficient condition for A to satisfy $A^{-1} > 0$ is that $a_{ij} \leq 0 \forall i \neq j$ and $a_{ii} > 0, i = 1, \dots, J - 1$.

Theorem 15.4 (*Limit Theorem*) Let A be an irreducible M -matrix having n rows and n columns. Then the unique solution of (15.29) is uniformly bounded in norm for all $t \geq 0$ and satisfies:

$$\lim_{t \rightarrow \infty} U(t) = A^{-1}F.$$

We are interested in determining the conditions under which spurious oscillations occur in the semi-discrete scheme (15.29). Most of the problems are caused by the eigenvalues of A .

Definition 15.4 A real matrix $Q = (q_{ij})$ is said to be *essentially positive* if $q_{ij} \geq 0$ and Q is irreducible.

The following theorems and definitions are taken from Varga (1962).

Theorem 15.5 Let Q be an essentially positive matrix. Then Q has a real eigenvalue $\lambda(Q)$ such that:

1. There exists an eigenvector $x > 0$ corresponding to $\lambda(Q)$.
2. If α is another eigenvalue of Q , then $\operatorname{Re} \alpha \leq \lambda(Q)$.
3. $\lambda(Q)$ increases when an element of Q increases.

Theorem 15.6 (*Asymptotic behaviour*) Let Q be an $n \times n$ essentially positive matrix. If $\lambda(Q)$ is the eigenvalue of Theorem 15.5, then:

$$\|\exp(tQ)\| \leq K \exp(t\lambda(Q)), \quad t \rightarrow \infty$$

where K is a positive constant independent of t .

Thus $\lambda(Q)$ dictates asymptotic behaviour of $\|\exp(tQ)\|$ for large t .

Definition 15.5 Let Q be essentially positive. Then Q is called:

- *Supercritical* if $\lambda(Q) > 0$.
- *Critical* if $\lambda(Q) = 0$.
- *Subcritical* if $\lambda(Q) < 0$.

We now consider (15.29) posed in a slightly different form (in fact, we use the same notation as in Varga (1962)):

$$\begin{aligned}\frac{dU}{dt} &= QU + r \text{ in } (0, T) \\ U(0) &= U_0.\end{aligned}\tag{15.30}$$

Theorem 15.7 (*Asymptotic behaviour of solution*): Let Q be essentially positive and non-singular. If Q is supercritical, then for a given initial vector U_0 the solution of (15.29) satisfies:

$$\lim_{t \rightarrow \infty} \|U(t)\| = \infty.$$

If Q is subcritical, then $U(t)$ is uniformly bounded in norm for all $t > 0$ and satisfies:

$$\lim_{t \rightarrow \infty} U(t) = -Q^{-1}r.$$

We thus see that it is necessary to have negative eigenvalues if we wish to ensure stable asymptotic behaviour of the solution of (15.29). We mention that if Q has no zero eigenvalues, then it is either supercritical or subcritical.

We give an example in the scalar case to motivate Theorem 15.7. Consider the simple initial value problem:

$$\begin{aligned}\frac{du}{dt} &= qu + r, \quad t > 0 \\ u(0) &= A\end{aligned}$$

where q and r are constant. By using the integrating factor method, we can show that the solution is given by:

$$u(t) = Ae^{qt} - \frac{r}{q}[1 - e^{qt}].$$

Thus, if $q < 0$ (the subcritical case) we see that:

$$\lim_{t \rightarrow \infty} u(t) = -\frac{r}{q}$$

while if $q > 0$ (the supercritical case) the solution is unbounded.

Finally, if $q \equiv 0$ the solution is given by:

$$u(t) = A + rt \text{ (linear growth).}$$

15.7.2 Fully Discrete Schemes

We now revert to system (15.29). We divide the interval $[0, T]$ into N subintervals defined by:

$$0 = t_0 < t_1 < \dots < t_N = T, \text{ with } (k = T/N).$$

We replace the continuous time derivative by divided differences. We concentrate on so-called *two-level schemes*. To this end, we approximate $\frac{dU}{dt}$ at some time level as follows:

$$\frac{dU}{dt} \cong \frac{U^{n+1} - U^n}{k}, \quad U^n \equiv U(t_n).$$

We use weighted averages defined as follows:

$$\Phi^{n,\theta} \equiv (1 - \theta)\Phi^n + \theta\Phi^{n-1} \text{ for a general function } \Phi = \Phi(t)$$

where $\theta \in [0, 1]$. The discrete scheme is now defined as follows:

$$\begin{aligned} & -\frac{U^{n+1} - U^n}{k} + AU^{n,\theta} = F \\ & U^0 = U_0. \end{aligned} \tag{15.31}$$

Some well-known special cases are now given. Assume A and F are constant for the moment.

$\theta = 0$: *The explicit Euler scheme*:

$$-\frac{U^{n+1} - U^n}{k} + AU^n = F \quad \left(U^{n,1/2} = \frac{U^n + U^{n+1}}{2} \right). \tag{15.32}$$

$\theta = \frac{1}{2}$: *The Crank–Nicolson scheme*:

$$-\frac{U^{n+1} - U^n}{k} + AU^{n,1/2} = F. \tag{15.33}$$

$\theta = 1$: *The fully implicit scheme*:

$$-\frac{U^{n+1} - U^n}{k} + AU^{n,1} = F. \tag{15.34}$$

We are interested in determining if the above schemes are stable (in some sense) and whether their solution converges to the solution of (15.29) as $k \rightarrow 0$. To this end, we write Equation (15.32) in the equivalent form:

$$U^{n+1} = CU^n + H$$

where the matrix C is given by:

$$C = (I - kA\theta)^{-1}(I + kA(1 - \theta))$$

and:

$$H = -k(I - kA\theta)^{-1}F.$$

A well-known result (see Varga (1962)) states that the solution of (15.29) is given by:

$$U(t) = A^{-1}F + \exp(tA)[U(0) - A^{-1}F].$$

So, in a sense the accuracy of the approximation (15.31) will be determined by how well the matrix C approximates the exponential of a matrix. We now discuss this problem.

Definition 15.6 The time-dependent matrix $T(t)$ is *stable* for $0 \leq t \leq T$ if $\rho(T(t)) \leq 1$ (See Definition 15.1.) It is *unconditionally stable* if $\rho(T(t)) < 1$ for all $0 \leq t \leq \infty$. We now state the main result of this section (see Varga (1962), p. 265).

Theorem 15.8 Let A be a matrix whose eigenvalues λ_j satisfy $0 < \alpha < \operatorname{Re} \lambda_j < \beta$ $\forall j = 1, \dots, n$. Then the explicit Euler scheme (15.32) is stable if:

$$0 \leq k \leq \min \left\{ \frac{2\operatorname{Re} \lambda_j}{|\lambda_j|^2} \right\}, \quad 1 \leq j \leq n$$

while the Crank–Nicolson scheme (15.33) and fully implicit scheme (15.34) are both unconditionally stable.

Definition 15.7 The matrix $T(t)$ is consistent with $\exp(-tA)$ if $T(t)$ has a matrix power development about $t = 0$ that agrees through at least linear terms with the expansion of $\exp(-tA)$.

We remark that the schemes defined by (15.32), (15.33) and (15.34) have matrices that are consistent with the exponential function.

15.8 PADÉ MATRIX APPROXIMATION

In this section we consider a class of first-order linear systems of ordinary differential equations in the independent variable t (this is usually a time dimension):

$$\frac{dV(t)}{dt} + A(t)V(t) = F(t), \quad 0 < t \leq T \quad (15.35)$$

where:

$$V(t) = (u_1(t), \dots, u_n(t))^{\top}$$

$$F(t) = (f_1(t), \dots, f_n(t))^{\top}$$

$$A(t) = (a_{ij}(t))_{1 \leq i,j \leq n}.$$

In this case the vector $F(t)$ and matrix function $A(t)$ are known quantities and the vector $U(t)$ is unknown. The system (15.35) will have a unique solution if we give an initial condition for $U(t)$ when $t = 0$:

$$V(0) = U_0, \quad U_0 = (u_{01}, \dots, u_{0n})^\top \quad (15.36)$$

where U_0 is a given constant vector.

The initial value problem (IVP) (15.35) is highly relevant to the material in this book and in particular its applications to finite difference methods for parabolic initial boundary problems.

15.8.1 Padé Matrix Approximations

Let us assume for the moment that the matrix A in Equation (15.35) is independent of t . We define (formally) the exponential of a matrix as follows:

$$\exp(A) \equiv I + A + \frac{A^2}{2!} + \dots \equiv \sum_{j=0}^{\infty} \frac{A^n}{n!} \quad (15.37)$$

where I is the identity matrix. (This is the $n \times n$ matrix with the value 1 on the main diagonal and zero everywhere else). Based on this definition, the solution of (15.35), (15.36) is given by:

$$V(t) = \exp(-At)U_0 + \exp(-At) \int_0^t \exp(A\lambda)F(\lambda)d\lambda, \quad t \geq 0. \quad (15.38)$$

In general, it is difficult or undesirable to attempt to use the form (15.38) directly in calculations. Furthermore, the matrix A can be a function of time and then formula (15.38) needs to be modified. Thus, we resort to numerical techniques. Some examples are:

- One-step and multistep finite difference method (FDM).
- Runge–Kutta methods (Stoer and Bulirsch (1980)).
- Predictor-corrector methods.

In this chapter we concentrate on one-step methods. To this end, we partition the interval $[0, T]$ into subintervals:

$$0 = t_0 < t_1 < t_2 < \dots < t_N = T$$

$$k_n = t_{n+1} - t_n, \quad n = 0, \dots, N - 1.$$

The subintervals do not necessarily have to be of the same size but for convenience we partition $[0, T]$ into N equally sized subintervals as follows:

$$k = T/N$$

$$k = t_{n+1} - t_n, \quad n = 0, \dots, N - 1.$$

Having done this, we must approximate the solution of IVP (15.35), (15.36). Some popular schemes are:

Implicit Euler scheme:

$$\begin{aligned} \frac{U^{n+1} - U^n}{k} + A^{n+1}U^{n+1} &= F^{n+1}, \quad n = 0, \dots, N-1 \\ U^0 &= U_0 \\ A^{n+1} &\equiv A(t_{n+1}), \quad F^{n+1} \equiv F(t_{n+1}). \end{aligned}$$

Explicit Euler scheme:

$$\begin{aligned} \frac{U^{n+1} - U^n}{k} + A^nU^n &= F^n, \quad n = 0, \dots, N-1 \\ U^0 &= U_0 \\ A^n &\equiv A(t_n), \quad F^n \equiv F(t_n). \end{aligned}$$

Crank–Nicolson scheme:

$$\begin{aligned} \frac{U^{n+1} - U^n}{k} + A^{n+\frac{1}{2}} \frac{U^{n+1} + U^n}{2} &= F^{n+\frac{1}{2}}, \quad n = 0, \dots, N-1 \\ U^0 &= U_0 \\ t_{n+\frac{1}{2}} &\equiv \frac{t_{n+1} + t_n}{2}, \quad A^{n+\frac{1}{2}} \equiv A\left(t_{n+\frac{1}{2}}\right), \quad F^{n+\frac{1}{2}} \equiv F\left(t_{n+\frac{1}{2}}\right). \end{aligned}$$

Noting from these schemes that data is known at time level n , we can then calculate new values at time level $n+1$. Formally, we can rewrite these schemes in a computational form. The new values are:

Implicit Euler scheme:

$$(I + kA^{n+1})U^{n+1} = U^n + kF^{n+1}. \quad (15.39)$$

Explicit Euler scheme:

$$U^{n+1} = (I - kA^n)U^n + kF^n. \quad (15.40)$$

Crank–Nicolson scheme:

$$\left(I + \frac{kA^{n+\frac{1}{2}}}{2} \right) U^{n+1} = \left(I - \frac{kA^{n+\frac{1}{2}}}{2} \right) U^n + kF^{n+\frac{1}{2}}. \quad (15.41)$$

where I is the identity matrix.

The solution at time level $n+1$ in Equation (15.40) can be found directly, while we must solve a matrix system for the Equations (15.39) and (15.41). We note that the

implicit Euler scheme is also called the *backward difference method* and the explicit Euler method is called the *forward difference method*.

Let us now take the case of $F(t) = 0$ where the matrix A is independent of time. We can then write the Equations (15.39), (15.40) and (15.41) in the equivalent forms in the case $F^n \equiv 0$ (at least formally):

$$\begin{aligned} (a) \quad U^{n+1} &= (I + kA)^{-1}U^n \\ (b) \quad U^{n+1} &= (I - kA)U^n \\ (c) \quad U^{n+1} &= \left(I + \frac{k}{2}A\right)^{-1} \left(I - \frac{k}{2}A\right)U^n. \end{aligned} \tag{15.42}$$

We can compare these solutions with the exact solution, namely:

$$W(t) = \exp(-tA)U_0. \tag{15.43}$$

We realise that the solutions in (15.42) are essentially approximations to the exponential matrix term in (15.43). To make this statement more clear, we examine at the Crank–Nicolson scheme. Let us assume that the time step k is sufficiently small. Then we can formally expand the expression for the approximate solution (15.42)(c) as follows:

$$\left(I + \frac{k}{2}A\right)^{-1} \left(I - \frac{k}{2}A\right) = I - kA + \frac{(kA)^2}{2} - \frac{(kA)^3}{4} + \dots$$

and we thus see that this series agrees with that in (15.37) to second order. Similarly it can be shown that the other numerical solutions in Equations (15.42) approximate the exponential term to first order in k . However, the scheme for the explicit Euler scheme is only conditionally stable, which means that k must be chosen to be less than some critical value. The implicit Euler and Crank–Nicolson schemes are unconditionally stable for any value of k . This means that:

$$\|U^n\| \leq M\|U_0\|, \quad n = 0, 1, \dots$$

in some norm. Here the constant M is independent of the step size k .

Theorem 15.9 (Stability of the explicit Euler scheme): *Let A be an $n \times n$ matrix whose eigenvalues λ_j satisfy $0 < \infty \leq \operatorname{Re}(\lambda_j) \leq \beta$, $1 \leq j \leq n$ (here $\operatorname{Re}(\lambda_j)$ denotes the real part of the complex number λ_j).*

Then the explicit Euler scheme approximant $I - kA$ is stable for:

$$0 \leq k \leq \min_{1 \leq j \leq n} \left\{ \frac{2\operatorname{Re}(\lambda_j)}{|\lambda_j|^2} \right\}.$$

Looking at Equations (15.42) again, we might ask ourselves how the approximations to the exponential are generated. In fact, the approximations in (15.42) are special

cases of *Padé rational approximants*. A *rational function* is a quotient of two polynomials, and we use such functions to approximate the exponential function as follows:

$$\exp(-z) = \frac{n_{p,q}(z)}{d_{p,q}(z)}$$

where n (the numerator) and d (the denominator) are polynomials of degrees q and p in z , respectively. In general, we select for each pair of non-negative integers p and q those polynomials n and d such that the Taylor's series expansion of n/d agrees with as many leading terms of Taylor expansion of $\exp(-z)$. We can thus create a so-called Padé table for $\exp(-z)$. Some of the first few terms in the table are shown in Figure 15.1.

The reader can verify that the entries in Figure 15.1 are correct. An important result concerning Padé approximations and stability of difference schemes for IVP (15.35), (15.36) is that if the eigenvalues of a matrix A are positive real numbers, then the Padé matrix approximation is unconditionally stable if and only if $p \geq q$.

The Padé matrix approximation technique can also be applied to other functions.

In this section we discuss how to bootstrap the accuracy of the implicit Euler method from first-order to second-order accuracy while avoiding spurious oscillations.

We motivate the extrapolated scheme in two ways. Let:

- U_k^n be the solution of (15.42)(a).
- W be the solution of (15.35), (15.36).

where we have introduced the subscript k in the approximate solution to denote its dependence on k . Then, we can prove (this will be discussed later) that:

$$U_k^n = W + Mk + O(k^2).$$

where the constant M does not depend on k . By now taking a scheme with a mesh of size $k/2$ we also trivially see that:

$$U_{k/2}^{2n} = W + \frac{Mk}{2} + O(k^2).$$

	$q = 0$	$q = 1$	$q = 2$
$p = 0$	1	$1 - z$	$1 - z + z^2/2$
$p = 1$	$\frac{1}{1+z}$	$\frac{2-z}{2+z}$	$\frac{6-4z+z^2}{6+2z}$
$p = 2$	$\frac{1}{1+z+z^2/2}$	$\frac{6-2z}{6+4z+z^2}$	$\frac{12-6z+z^2}{12+6z+z^2}$

FIGURE 15.1 Padé table for $\exp(-z)$.

Some arithmetic shows that:

$$V_{k/2}^{2n} \equiv 2U_{k/2}^{2n} - U_k^n = W + O(k^2).$$

Thus, we now get a second-order scheme.

We now motivate this extrapolated scheme based on Padé matrix approximations and the series form for exponential matrices (based on Lawson and Morris (1978)). We denote the approximate solution by V , and let us drop the dependence on the discrete time level t ; we just take any time value t . Applying (15.42)(a) on a mesh of size k we see that:

$$V(t+k) = (I + kA)^{-1}V(t). \quad (15.44)$$

Alternatively, we can progress from time t to time $t+k$ in two steps, namely from t to $t+k/2$ and then from $t+k/2$ to $t+k$, and this combined step gives:

$$V(t+k) = \left(I + \frac{k}{2}A\right)^{-1} \left(I + \frac{k}{2}A\right)^{-1} V(t). \quad (15.45)$$

Expanding (15.44) and (15.45) in powers of k gives:

$$V(t+k) = (I + kA + k^2A^2)V(t) + O(k^3) \quad (15.46)$$

and:

$$V(t+k) = \left(I + kA + \frac{3}{4}k^2A^2\right)V(t) + O(k^3). \quad (15.47)$$

We get:

$$V(t+k) = \left(I + kA + \frac{k^2}{2}A^2\right)V(t) + O(k^3).$$

Comparing this result with the series expansion in Equation (15.37), we then get a second-order approximation. This suggests the following algorithm:

$$\begin{aligned} V^{(1)}(t+k) &= (I + kA)^{-1}V(t) \\ V^{(2)}(t+k) &= \left(I + \frac{k}{2}A\right)^{-1} \left(I + \frac{k}{2}A\right)^{-1} V(t) \\ V(t+k) &= 2V^{(2)} - V^{(1)}. \end{aligned}$$

We thus have produced the second-order scheme. We formalise this reasoning in Chapter 22.

Extrapolation techniques in conjunction with the implicit Euler scheme have been applied to the Black–Scholes, and good results have been obtained: second-order accuracy and no spurious oscillations. They can be seen as a competitor to the Crank–Nicolson scheme.

15.9 TIME-DEPENDENT CONVECTION-DIFFUSION EQUATIONS

We now consider the time-dependent convection-diffusion problem:

$$-\frac{\partial u}{\partial t} + Lu = f(x, t), \quad a < x < b, \quad t > 0$$

where:

$$Lu(x, t) \equiv \sigma(x, t) \frac{\partial^2 u}{\partial x^2} + \mu(x, t) \frac{\partial u}{\partial x} + b(x, t)u.$$

The initial and boundary conditions are:

$$\begin{aligned} u(x, 0) &= f(x), \quad a \leq x \leq b \\ u(a, t) &= g(t), \quad u(b, t) = h(t), \quad t > 0. \end{aligned} \quad (15.49)$$

There are different ways to discretise (15.48), (15.49), namely:

- Discretise in x and t simultaneously (*Fully discrete schemes*).
- Discretise in x and keep t continuous (*Method of Lines*).
- Discretise in t and keep x continuous (*Rothe's method*).

The choice of approach will be determined by a number of factors which we will explain in later chapters.

15.9.1 Fully Discrete Schemes

We use the usual notation for meshes in the x and t directions. For example, h is the mesh size in the x direction, while k is the mesh size in the t direction. Define the operator:

$$L_k^h w_j^n \equiv \sigma_j^n D_+ D_- w_j^n + \mu_j^n D_0 w_j^n + b_j^n w_j^n \quad 1 \leq j \leq J - 1, \quad n \geq 0$$

for some mesh function w_j^n where:

$$\left. \begin{array}{l} \sigma_j^n = \sigma(x_j, t_n) \\ \mu_j^n = \mu(x_j, t_n) \\ b_j^n = b(x_j, t_n) \\ f_j^n = f(x_j, t_n) \end{array} \right\} 1 \leq j \leq J - 1, \quad n \geq 0.$$

Then, based on the first-order schemes in previous chapters, we can define a number of fully discrete schemes as follows:

- Implicit Euler scheme:

$$-\frac{u_j^{n+1} - u_j^n}{k} + L_k^h u_j^{n+1} = 0. \quad (15.50)$$

- Explicit Euler scheme:

$$-\frac{u_j^{n+1} - u_j^n}{k} + L_k^h u_j^n = 0. \quad (15.51)$$

- Crank–Nicolson scheme:

This is the average of schemes (15.50) and (15.51) (add the left and right hand sides of (15.50) and (15.51)) defined as:

$$-\frac{u_j^{n+1} - u_j^n}{k} + \frac{1}{2}(L_k^h u_j^{n+1} + L_k^h u_j^n) = 0. \quad (15.52)$$

Summarising, we note:

- The Crank–Nicolson method is second-order accurate on uniform meshes only.
- It produces spurious oscillations and possibly spikes for problems with non-smooth initial and boundary conditions, and for problems where the *compatibility conditions* between boundary and initial conditions are not satisfied.
- It reduces to a neutrally stable method when the diffusion coefficient is small. This has the implication that the accuracy of the results will be compromised due to possible rounding errors.
- It gives terrible results near the strike price for approximations to the first and second derivatives of the solution in the space direction (the greeks). In pricing applications, this translates to the statement that Crank–Nicolson gives bad approximations to the delta and gamma of the option price. We discuss these issues in Chapters 16 and 17.

15.10 SUMMARY AND CONCLUSIONS

In this chapter we discussed stability and accuracy of semi-discrete and fully discrete finite difference schemes for one-dimensional time-dependent convection-diffusion-reaction partial differential equations. It is a continuation of the methods that we discussed in Chapter 14. We also discussed the interaction between the diffusion and convection terms, how standard schemes have difficulty with convection-dominated problems and how to resolve them using exponentially fitted methods.

CHAPTER 16

Sensitivity Analysis, Option Greeks and Parameter Optimisation, Part I

The first principle is that you must not fool yourself – and you are the easiest person to fool.

Richard Feynman

16.1 INTRODUCTION AND OBJECTIVES

In this chapter we introduce a number of important concepts and methods that permeate many application areas and domains. The methods discussed here can be seen as special cases of a study called *sensitivity analysis* that is concerned with how uncertainty in the output of a mathematical model or numerical system can be ascribed to uncertainty in its inputs. In particular, we can recalculate outcomes under alternative assumptions to determine the impact of modifying the value of a variable or parameter and its influence on the solution of a given problem. Some reasons for applying sensitivity analysis in models and systems are:

- Better understanding of the relationships between input and output variables.
- Locating regions in input space for which the output attains a maximum or minimum or meets some criterion, for example in optimisation.
- Calibrating models with a large number of parameters; some parameters are more important than others. We are interested in computing how the solution of the model changes when one or more parameter values are changed. For example, an option's delta represents the change in option price as a function of the underlying stock.

- Uncertainty reduction: we need to identify model inputs that will cause significant uncertainty in the output. An example is *Uncertain Volatility Models* (UVM) in option pricing problems in which all we can say about the volatility is that its values are constrained to be in a given bounded interval.
- Computing option ‘greeks’ (Haug (2007)).

We discuss even more ways to compute sensitivities in Chapter 17.

16.2 HELICOPTER VIEW OF SENSITIVITY ANALYSIS

The computation of derivatives and gradients of scalar and vector-valued functions is a pervasive and ongoing activity in many areas of computational finance, optimisation, engineering and machine learning (ML). In general, we may need to compute derivatives as part of some algorithm or software program.

Some specific applications are:

- Computing the Black–Scholes greeks/sensitivities.
- Hedge ratios in fixed income and credit risk applications (for example, vega, duration and convexity).
- Optimisation of design shape parameters.
- Computing derivatives and gradients in neural network (NN) algorithms, for example (stochastic) gradient descent method (SGD).
- Hedge ratios for stochastic differential equations (SDEs), using automatic differentiation (AD), for example.
- Global optimisation of (Langevin) SDEs and *gradient ODE systems*.
- Applications (many) in which gradients, Jacobians and Hessians need to be calculated.
- Robust calibration and volatility surfaces.

There are several established methods to compute the derivatives of functions:

- *Analytically* (using calculus mainly).
- Using *finite differences* (or to be more precise, *divided differences*) to approximate derivatives (this is a well-known approach in PDE models and numerical differentiation).
- *Automatic (algorithmic) differentiation* (AD).
- *Continuous sensitivity equation* (CSE). In this case we view the derivative of a function (with respect to a parameter) as a discrete quantity that satisfies an ordinary or partial differential equation with auxiliary initial and boundary conditions.
- Approximating the function whose derivatives we wish to compute by an ‘easier’ or surrogate function, for example a cubic spline whose derivatives are easily computed.
- The *Complex Step Method* (CSM). This is a less known but robust method to compute derivatives using function values alone (with no need to take divided differences). The method employs complex numbers and complex arithmetic.

There is no golden rule as to which of the above methods to use in general because the choice depends on the requirements and context. In general, we wish a given method to be suitable for the problem at hand, and ideally it should be efficient and robust, and the resulting code should be easy to use and to create.

16.3 BLACK-SCHOLES-MERTON GREEKS

In order to reduce the scope we restrict our attention to calculating the price C of a European call option (put options follow a similar path). Furthermore, we wish to calculate the values of some of its sensitivities (the so-called greeks), for example:

$$\begin{aligned} \text{Delta} &= \Delta_C = \frac{\partial C}{\partial S} \\ \text{Gamma} &= \Gamma_C = \frac{\partial^2 C}{\partial S^2} = \frac{\partial \Delta_C}{\partial S} \\ \text{Vega} &= \frac{\partial C}{\partial \sigma} \\ \text{Theta} &= \Theta_C = -\frac{\partial C}{\partial t}. \end{aligned} \tag{16.1}$$

For European options we can give an exact formula for the call price and its sensitivities (Cox and Rubenstein (1985)), and we use these values as benchmarks against which to test our finite difference schemes. We have not listed all possible sensitivities in (16.1), and the interested reader can find formulae for all the major ones in Haug (2007).

We introduce the generalised Black–Scholes formula to calculate the price of a call option on some underlying asset. In general the call price is a function of five parameters (we assume zero dividend):

$$C = C(S, K, T, r, \sigma) \tag{16.2}$$

where:

S = asset price

K = strike (exercise) price

T = expiration

r = risk-free interest rate

σ = constant volatility

b = cost of carry.

We can view the call option price C as a vector function because it maps a vector of parameters into a real value. The analytic formula for C is given by:

$$C = Se^{(b-r)T}N(d_1) - Ke^{-rT}N(d_2) \quad (16.3)$$

where $N(x)$ is the standard *cumulative normal (Gaussian) distribution* function defined by:

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-y^2/2} dy \quad (16.4)$$

and:

$$\begin{aligned} d_1 &= \frac{\log(S/K) + (b + \sigma^2/2)T}{\sigma\sqrt{T}} \\ d_2 &= \frac{\log(S/K) + (b - \sigma^2/2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}. \end{aligned} \quad (16.5)$$

The *cost-of-carry* parameter b has specific values depending on the kind of security in question (Haug (2007)):

$b = r$, we have the Black and Scholes stock option model.

$b = r - q$, the Morton model with continuous dividend yield q .

$b = 0$, the Black futures option model.

$b = r - R$, the Garman and Kohlhagen currency option model, where R is the foreign risk-free interest rate.

Thus, we can find the price of a plain call option by using Formula (16.3). Furthermore, it is possible to differentiate C with respect to any one of the parameters to produce a formula for the option sensitivities. For example, some tedious differentiation allows us to prove that:

$$\begin{aligned} \Delta_C &\equiv \frac{\partial C}{\partial S} = e^{(b-r)T} N(d_1) \\ \Gamma_C &\equiv \frac{\partial^2 C}{\partial S^2} = \frac{\partial \Delta_C}{\partial S} = \frac{Kn(d_1)e^{(b-r)T}}{S\sigma\sqrt{T}} \\ Vega_C &\equiv \frac{\partial C}{\partial \sigma} = S\sqrt{T}e^{(b-r)T} n(d_1) \\ \Theta_C &\equiv -\frac{\partial C}{\partial T} = -\frac{S\sigma e^{(b-r)T} n(d_1)}{2\sqrt{T}} - (b - r)Se^{(b-r)T} N(d_1) - rKe^{-rT}N(d_2) \end{aligned} \quad (16.6)$$

where $n(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$.

Thus, not only do we have exact formulae for the call price, but we also have exact formulae for its sensitivities. We can then determine how the call price varies as a function of the change in one or more of the option's parameters. In particular, we are interested in delta and gamma for problems where there is no exact solution, and in these cases we resort to finite difference schemes, for example. Of course, we need some assurance that our approximations are accurate.

We shall work out the formula for the vega of a call option for those readers who wish to refresh their mathematics in the area of differential calculus. (It took the author two attempts to get the formula right!). For more information on calculus, see, for example, Widder (1989).

Before we embark on calculating vega, we must do some preliminary work. First, let $n(x)$ be the derivative of the normal cumulative distribution function. Then:

$$n(x) = \frac{dN(x)}{dx} = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \quad (16.7)$$

and furthermore, you can check that the following are true:

$$1. \frac{\partial N(x)}{\partial \eta} = n(x) \frac{\partial x}{\partial \eta} \text{ for } \eta = \sigma, S \text{ or } T$$

$$2. n(d_2) = n(d_1) S e^{bT} / K.$$

Then the formula for the vega is calculated using the following sequence of steps:

$$\begin{aligned} \text{Vega} &= \frac{\partial C}{\partial \sigma} \\ &= \frac{\partial}{\partial \sigma} \left\{ S e^{(b-r)T} N(d_1) - K e^{-rT} N(d_2) \right\} \\ &= S e^{(b-r)T} \frac{\partial}{\partial \sigma} N(d_1) - K e^{-rT} \frac{\partial}{\partial \sigma} N(d_2) \\ &= S e^{(b-r)T} n(d_1) \frac{\partial d_1}{\partial \sigma} - K e^{-rT} n(d_2) \frac{\partial d_2}{\partial \sigma} \\ &= S e^{(b-r)T} n(d_1) \frac{\partial d_1}{\partial \sigma} - K e^{-rT} \left[\frac{n(d_1) S e^{bT}}{K} \right] \frac{\partial d_2}{\partial \sigma} \\ &= S e^{(b-r)T} n(d_1) \left[\frac{\partial d_1}{\partial \sigma} - \frac{\partial d_2}{\partial \sigma} \right] \\ &= S e^{(b-r)T} n(d_1) \sqrt{T}. \end{aligned}$$

Here we have used the fact that:

$$\frac{\partial d_1}{\partial \sigma} = \frac{\log(S/K) + (b + \sigma^2/2)}{\sigma^2 \sqrt{T}}$$

because of the relationship $d_2 = d_1 - \sigma \sqrt{T}$.

Similar formulae for put options can be found in Haug (2007) and Cox and Rubinstein (1985).

16.3.1 Higher-Order and Mixed Greeks

In general, the greeks describe the rate of change of the option with respect to one or more of its underlying parameters. *First-order greeks* are delta, vega and theta because we take the first-order derivative of the option price formula with respect to a single parameter, while gamma is a *second-order greek* because it is the second derivative of the option price with respect to the underlying stock. Analogously, we can take *third-order* and *mixed derivatives* to produce the corresponding greeks, for example, in the case of a call option:

$$\begin{aligned} \text{Speed} &: \frac{\partial^3 C}{\partial S^3}. \\ \text{Strike gamma} &: \frac{\partial^2 C}{\partial K^2}. \\ \text{Vanna} &: \frac{\partial^2 C}{\partial S \partial \sigma}. \\ \text{Charm} &: -\frac{\partial^2 C}{\partial S \partial T}. \end{aligned} \tag{16.8}$$

for which there are analytical solutions as seen in Haug (2007). These are useful to compare against when we compute greeks by non-analytical methods.

16.4 DIVIDED DIFFERENCES

In this section we discuss numerical techniques to approximate the first-order and second-order derivatives of a function. We compute these quantities in a range of applications, such as the numerical approximation of ordinary and partial differential equations, integral equations and computing option sensitivities. We also discuss why *numerical differentiation* is an *unstable process* and how to remedy this problem.

16.4.1 Approximation to First and Second Derivatives

In this section we examine a real-valued function of a real variable:

$$y = f(x).$$

In general we are interested in finding approximations to the first-order and second-order derivatives of the function f . This is needed in this book because, in general, the form of the function f may be unknown, and it is thus impossible to calculate its derivatives analytically. Second, it may be costly to evaluate the derivatives. To this end, we resort to numerical approximations. Suppose that we wish to approximate the first-order derivative of y at some point a (see Figure 16.1) and assume that h is a (small) positive number.

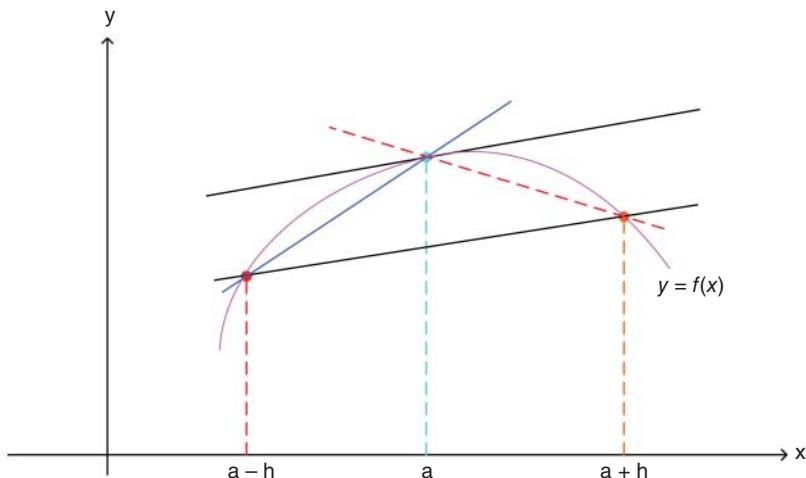


FIGURE 16.1 Motivating divided differences.

The first approximation (called the *centred-difference formula*) is given by:

$$f'(a) \approx \frac{f(a+h) - f(a-h)}{2h}.$$

Another approximation is called the *forward-difference formula* given by:

$$f'(a) \approx \frac{f(a+h) - f(a)}{h}.$$

Finally, the *backward-difference formula* is given by:

$$f'(a) \approx \frac{f(a) - f(a-h)}{h}.$$

We use the following notation throughout this book:

$$D_0 f(a) \equiv \frac{f(a+h) - f(a-h)}{2h}, \quad (16.9)(a)$$

$$D_+ f(a) \equiv \frac{f(a+h) - f(a)}{h}, \quad (16.9)(b)$$

$$D_- f(a) \equiv \frac{f(a) - f(a-h)}{h}. \quad (16.9)(c)$$

Let us examine the centred-difference case, Equation (16.9a). We use Taylor's expansion with *exact remainder* to show that:

$$\begin{cases} f(a \pm h) = f(a) \pm hf'(a) + \frac{h^2}{2!}f''(a) \pm \frac{h^3}{3!}f'''(\eta \pm) \\ \eta_- \in (a-h, a), \eta_+ \in (a, a+h) \end{cases}$$

from which we conclude that in this particular case:

$$D_0 f(a) = f'(a) + \frac{h^2}{6} \left(\frac{f'''(\eta_+) + f'''(\eta_-)}{2} \right).$$

We see that centred differencing gives a second-order approximation to the first-order derivative if h is small enough and if f has continuous derivatives up to order three. Similarly, some arithmetic shows that forward and backward differencing approximations to the first-order derivative of f at the point a are:

$$\begin{aligned} D_+ f(a) &= f'(a) + \frac{h}{2} f''(\eta_+), \quad \eta_+ \in (a, a+h) \\ D_- f(a) &= f'(a) - \frac{h}{2} f''(\eta_-), \quad \eta_- \in (a-h, a). \end{aligned}$$

We see that these one-sided schemes are only first-order accurate. On the other hand, they place low continuity constraints on the function f ; namely, we only need to assume that its second derivative is continuous.

We now discuss divided differences for the second-order derivative of f at some point a . To this end, we propose the following popular *three-point formula* (see Conte and De Boor (1981)):

$$D_+ D_- f(a) \equiv \frac{f(a-h) - 2f(a) + f(a+h)}{h^2}.$$

Thus, this divided difference is a second-order approximation to the second-order derivative of f at the point a , and we assume that this function has continuous derivatives up to and including order four. The *discretisation error* is given by:

$$D_+ D_- f(a) = f''(a) + \frac{h^2}{4!} (f^{(iv)}(\eta_+) + f^{(iv)}(\eta_-)).$$

In general, using divided differences to approximate the derivatives of stand-alone functions results in an *ill-posed problem*. The problem is caused by the step size h and by the computer! Admittedly, smaller values of h reduce the discretisation error, but then roundoff errors will kick in and results become meaningless. This is because computers have limited word length, and we experience *catastrophic subtraction cancellation* when almost equal quantities are subtracted.

Finally, we give examples of second-order and third-order accuracy divided difference approximations to the first-order derivative of a scalar function. The disadvantage is the extra number of function evaluations compared to simpler methods:

$$\begin{aligned} f'(x) &\sim \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h} \\ f'(x) &\sim \frac{3f(x) - 4f(x-h) + f(x-2h)}{2h} \end{aligned} \tag{16.10}$$

$$\begin{aligned} f'(x) &\sim \frac{2f(x+h) + 3f(x) - 6f(x-h) + f(x-2h)}{6h} \\ f'(x) &\sim \frac{-f(x+2h) + 6f(x+h) + -3f(x) - 2f(x-h)}{6h}. \end{aligned} \quad (16.11)$$

There should be compelling reasons for using (16.10) and (16.11) in applications.

16.4.2 Black–Scholes Numeric Greeks and Divided Differences

It is possible to approximate option greeks using divided differences. We give a short discussion.

The *delta* measures the absolute change in the option price with respect to a small change in the price of the underlying asset:

$$\Delta_C = \frac{\partial C}{\partial S}. \quad (16.12)$$

The delta represents the hedge ratio, the number of options to write or to buy in order to create a risk-free portfolio. The delta varies from zero for deep out-of-the-money calls to one for deep in-the-money calls. This is clear if we examine the pay-off function. However, the delta is not continuous at the strike price K because it is zero to the left of K and one to the right of K for a call option. We thus expect problems near K , and this is borne out in practice by the appearance of so-called *spurious* or *non-physical oscillations* when we use Crank–Nicolson time averaging, for example.

Approximation of the delta takes place by using divided differences. We can choose between forward, backward or centred difference schemes as introduced in Section 16.4.1. For example, we use centred differences in the interior of the domain, while we use one-sided divided differences at the boundaries. Assuming that we have an array of call prices $\{C_j^n\}$, then we can compute delta as typically seen with lattice models and naïve numerical approaches:

$$\begin{aligned} \text{Discrete Delta } D_j^n &= \frac{C_{j+1}^n - C_{j-1}^n}{2h}, \quad 1 \leq j \leq J-1 \\ &= \frac{C_1^n - C_0^n}{h} \quad (j=1) \\ &= \frac{C_J^n - C_{J-1}^n}{h} \quad (j=J). \end{aligned}$$

The *gamma* measures the change in delta:

$$\Gamma_C = \frac{\partial^2 C}{\partial S^2} = \frac{\partial \Delta_C}{\partial S}. \quad (16.13)$$

Γ_C is greatest for at-the-money options, and it is nearly zero for deep in-the-money or deep out-of-the-money calls. The gamma gives us an indication of the vulnerability of the hedge ratio.

We can approximate Formula (16.13) for the gamma by using divided differences:

$$\text{Discrete Gamma } \Gamma_j^n = \frac{D_{j+1}^n - D_{j-1}^n}{2h}, \quad 1 \leq j \leq J-1$$

where D_j^n is the discrete delta value.

Similarly, we can calculate the derivative of C with respect to r :

$$\rho_C = -\frac{\partial C}{\partial r}.$$

Exact formulae are known for this quantity. For a call option with zero and non-zero cost-of-carry, these are:

$$\begin{aligned}\rho_C &= -T_C \quad (b = 0) \\ \rho_C &= TKe^{-rT} N(d_2) \quad (b \neq 0).\end{aligned}$$

In general, when an exact option price eludes us, we can then resort to finite difference to find an approximate solution.

16.5 CUBIC SPLINE INTERPOLATION

In this section we discuss a special kind of function called the *cubic spline* (Ahlberg, Nilson and Walsh (1967)). This is a function that is defined in an interval and whose values are equal to given values at prescribed mesh points in the interior of the interval. In each subinterval, the spline is a third-order polynomial, and both it and its first and second derivatives are continuous at the interior mesh points. We construct the cubic spline by solving a tridiagonal system of equations using the Double Sweep method, for example. Cubic splines are useful in a wider context, for example:

- Interpolation of discrete data and producing a smooth function.
- Approximating continuous functions.
- Numerical integration (Stroud (1974)).
- Solving boundary value problems.

In interest rate applications, cubic splines are used in the construction of yield curves and forward curves. In general, the x -axis corresponds to time, while the y -axis can correspond to zero rates, forward rates, discount rates or some function of these rates (Duffy and Germani (2013), Ron (2000)). More generally, cubic splines are used to interpolate between values that are known at discrete mesh points. We shall see examples in later chapters when we approximate the solution of the Black–Scholes PDE by the finite difference method. Having computed the option price at discrete points,

we can then interpolate these values to produce a value for any arbitrary value of the underlying variable.

We discuss two popular interpolation methods in this section. First, consider the pair of data values (x_0, y_0) and (x_1, y_1) . We are interested in drawing a straight line between these points (*linear interpolation*). Some algebra shows that the value y corresponding to a given value x is given by:

$$y = y_0 + (x - x_0) \frac{y_1 - y_0}{x_1 - x_0}. \quad (16.14)$$

This formula can now be extended to a data set consisting of n points. It has the advantage that it never overshoots nor oscillates. However, the generated curve is only continuous as the first-order derivatives are discontinuous at the data points. Thus, the curve looks somewhat jagged.

It is important to note that linear interpolation incurs an error that is bounded by the second derivative of the function being approximated.

Second, we now introduce the *cubic spline method*. We work on the interval (a, b) and let us suppose that $\delta = \{x_j : j = 0, 1, \dots, n\}$ is a *partition* of (a, b) with mesh points $a = x_0 < x_1 < \dots < x_n = b$. We define given function values as follows:

$$Y = \{y_j : j = 0, 1, \dots, n\}$$

and the quantities:

$$h_{j+1} = x_{j+1} - x_j, \quad j = 0, 1, \dots, n - 1.$$

We are now ready to define what a cubic spline is. This is a continuous function whose derivatives up to and including order two are continuous at the interior mesh points. Furthermore, the spline is a polynomial of third degree on each subinterval.

In order to uniquely specify the spline S_δ , we need to give boundary conditions at $x = a$ and at $x = b$. The mutually exclusive options are as follows:

$$S''_\delta(Y; a) = S''_\delta(Y; b) = 0$$

and

$$S'_\delta(Y; a) = \alpha, \quad S'_\delta(Y; b) = \beta \quad (\alpha, \beta \text{ given}). \quad (16.15)$$

In this case, either the second-order derivatives of the spline function are zero at $x = a$ and at $x = b$, or the first-order derivatives of the spline function are given at $x = a$ and at $x = b$ and have values α and β , respectively. It can be shown (Stoer and Bulirsch (1980)) that:

$$S_\delta(Y; x) = M_j \frac{(x_{j+1} - x)^3}{6h_{j+1}} + M_{j+1} \frac{(x - x_j)^3}{6h_{j+1}} + A_j(x - x_j) + B_j$$

where:

$$A_j = \frac{y_{j+1} - y_j}{h_{j+1}} - \frac{h_{j+1}}{6}(M_{j+1} - M_j)$$

$$B_j = y_j - M_j \frac{h_{j+1}^2}{6}, \quad j = 0, \dots, n-1.$$

All parameters and coefficients are known with the exception of $\{M_j\}_{j=0}^n$, and we can solve for these parameters by writing the problem as a *tridiagonal matrix system*:

$$AM = d \quad (16.16)$$

where:

$$A = \begin{pmatrix} 2 & \lambda_0 & & & \\ \mu_1 & 2 & \lambda_1 & & 0 \\ & \mu_2 & \ddots & \ddots & \\ & & \ddots & \ddots & \ddots \\ 0 & & & \ddots & \ddots & \lambda_{n-1} \end{pmatrix}$$

$$M = (M_0, \dots, M_n)^\top, \quad d = (d_0, \dots, d_n)^\top$$

where some of the elements of the matrix A depend on the boundary conditions (16.15), namely zero values for the second-order derivatives (case (a) below) or given values for the first-order derivatives (case (b) below):

Case (a)

$$\lambda_0 = 0, \quad d_0 = 0, \quad \mu_n = 0, \quad d_n = 0.$$

Case (b)

$$\lambda_0 = 1, \quad d_0 = \frac{6}{h_1} \left(\frac{y_1 - y_0}{h_1} - \alpha \right)$$

$$\mu_n = 1, \quad d_n = \frac{6}{h_n} \left(\beta - \frac{y_n - y_{n-1}}{h_n} \right)$$

$$\lambda_j = \frac{h_{j+1}}{h_j + h_{j+1}}, \quad \mu_j = 1 - \lambda_j = \frac{h_j}{h_j + h_{j+1}}$$

$$d_j = \frac{6}{h_j + h_{j+1}} \left\{ \frac{y_{j+1} - y_j}{h_{j+1}} - \frac{y_j - y_{j-1}}{h_j} \right\} \quad j = 1, 2, \dots, n-1.$$

We have reduced the problem of interpolating data points by cubic splines to a known problem of solving a tridiagonal matrix system.

We now discuss the C++ code (from Duffy (2018)) that implements the algorithm for cubic spline interpolation. The main member functions are:

- Construct a cubic spline based on x (abscissa) and y (values) input arrays (discrete case).
- Construct a cubic spline based on x array (abscissa) and a function (continuous case).
- Compute a y value for a given x value.
- Compute the integral of the cubic spline function.
- Compute the first-order and second-order derivatives of the cubic spline function.

The code in Duffy (2018) supports the boundary conditions as stated in Equation (16.15); however, periodic boundary conditions are not supported in this version of the software.

The integral of the spline over the interval (a, b) results directly from its definition (Ahlberg, Nilson and Walsh (1967), p. 44):

$$\int_0^b S_\delta(x)dx = \sum_{j=1}^n \frac{f_{j-1} + f_j}{2} h_j - \sum_{j=1}^n \frac{M_{j-1} + M_j}{24} h_j^3. \quad (16.17)$$

In the case of a constant mesh size, the integral in Equation (16.17) has the simpler form:

$$\int_a^b S_\delta(x)dx = h \sum_{j=1}^N \frac{f_{j-1} + f_j}{2} - \frac{h^3}{12} \sum_{j=1}^n \frac{M_{j-1} + M_j}{2}. \quad (16.18)$$

Differentiation of the formula for the cubic spline leads to representations for its first two derivatives:

$$S''_\delta(x) = M_j \left(\frac{x_{j+1} - x}{h_{j+1}} \right) + M_{j+1} \left(\frac{x - x_j}{h_{j+1}} \right) \text{ for } x \in [x_j, x_{j+1}] \quad (16.19)$$

$$S'_\delta(x) = -M_j \left(\frac{(x_{j+1} - x)^2}{2h_{j+1}} \right) + M_{j+1} \left(\frac{(x - x_j)^2}{2h_{j+1}} \right) + A_j \text{ for } x \in [x_j, x_{j+1}]. \quad (16.20)$$

16.5.1 Caveat: Cubic Splines with Sparse Input Data

Cubic splines are well-behaved for many kinds of applications, but they can overshoot and produce spurious negative values, especially for non-equidistant and sparse data. This problem manifests itself in yield curve construction. An example is (Duffy and Germani (2013)):

```
std::vector<double> t{0.1, 1.0, 4.0, 9.0, 20.0, 30.0}; // Time in years
std::vector<double> r{0.081, 0.07, 0.044, 0.07, 0.04, 0.03}; // Interest rate
```

In these cases other methods are better, for example the Hagan–West, Akima and Hyman methods (Duffy and Germani (2013)).

We give a brief mathematical explanation of why cubic splines fail under certain circumstances. To this end, we concentrate on the constraints on the distribution of the mesh values. Let $\{\Delta_k\}, k = 0, 1, 2, \dots$ be a sequence of meshes consisting of N_k sub-intervals on the interval $[a, b]$ defined by:

$$\Delta_k : a = x_{k,0} < x_{k,1} < \dots < x_{k,N_k} = b.$$

We define the norm of Δ_k as:

$$\|\Delta_k\| = \max_{1 \leq j \leq N_k} (h_{k,j}) \text{ where } h_{k,j} = x_{k,j} - x_{k,j-1}, \quad j = 1, \dots, N_k.$$

We are interested in sequences $\{\Delta_k\}$ for which $\|\Delta_k\| \rightarrow 0$ as $k \rightarrow \infty$. We also require that:

$$R_{\Delta_k} \equiv \max_{1 \leq j \leq N_k} \frac{\|\Delta_k\|}{h_{k,j}} \leq \beta < \infty. \quad (16.21)$$

This inequality states that the ratio of the largest subinterval and the smallest subinterval is bounded. The inequality occurs in many branches of numerical analysis and its applications.

As an example, consider the mesh $\{1,2,3,3.1,5.1,6,7,8\}$ that occurs in yield curve construction. The mesh sizes are $\{1,1,0.1,2,0.9,1,1\}$ and hence $\Delta = 2, R_{\Delta} = 20$. (We have dropped the dependence on k .)

The importance of constraint (16.21) lies in the fact that cubic splines produce approximations to smooth functions. The following theorem is proved in Ahlberg, Nilson and Walsh (1967), pp. 29–34:

Theorem 16.1 Let $f(x)$ be of class $C^3[a, b]$ (f and its first three derivatives are continuous on the interval $[a, b]$). Assume that $\lim_{k \rightarrow \infty} \|\Delta_k\| = 0$ and that the mesh Δ_k satisfies constraint (16.21). Then we have $f^{(p)}(x) - S^{(p)}(x) = o(\|\Delta_k\|^{3-p})$ ($p = 0, 1, 2, 3$) (where $f^{(p)}(x)$ is the p^{th} derivative of f with respect to x).

16.5.2 Cubic Splines for Option Greeks

We give a summary of using cubic splines for option greeks (a full account and C++ code is given in Duffy (2018)).

A snapshot of the Excel is shown in Table 16.1.

Examining the accuracy of the results in Table 16.1, we see that both the Crank–Nicolson and cubic spline interpolator produce similar results for delta, while the interpolator's accuracy for gamma is less accurate than the Crank–Nicolson method. What is the reason? For smooth functions, centred divided difference approximations to the first and second derivatives result in second-order accuracy. This approach is based on Taylor expansions. Cubic splines are polynomials, and they are a third-order accurate approximation to a smooth function. The first derivative of the spline is a second-order

TABLE 16.1 Output from steps 1–5; data is
 $K = 65, T = 0.25, r = b = 0.8, v = 0.3, NS = 325, NT = 1000.$

/*	S	exact delta	CN delta	Spline delta	exact gamma	CN gamma	Spline gamma
59	0.330935233	0.330702564	0.330478248	0.040967715	0.04098453	0.041068871	
60	0.372482798	0.372232265	0.372093554	0.042043376	0.042074873	0.042161739	
61	0.41484882	0.414593361	0.414541127	0.042603904	0.042647319	0.042733408	
62	0.457519085	0.457270132	0.457302102	0.042654217	0.042706223	0.042788541	
63	0.499993235	0.499760129	0.499871257	0.042216737	0.04227377	0.042349769	
64	0.541801022	0.541590678	0.541773643	0.041328768	0.041387329	0.041455004	
65	0.582515647	0.582332471	0.582578239	0.040039354	0.040096257	0.040154187	
66	0.62176381	0.621609821	0.621908227	0.038405882	0.038458444	0.038505789	
67	0.659232357	0.659107471	0.659447781	0.036490702	0.036536856	0.03657332	
68	0.694671633	0.694574052	0.694945475	0.034357967	0.034396306	0.034422069	
69	0.727895851	0.727822509	0.728214631	0.032070838	0.032100608	0.032116243	
70	0.758780913	0.758727922	0.759131055	0.029689185	0.029710218	0.029716604	
71	0.787260172	0.787223244	0.787628682	0.027267796	0.027280425	0.027278649*/	

approximation to the exact delta, while the second derivative of the spline is a first-order approximation to the exact gamma. This theoretical result is proven mathematically in Alberg, Nilson and Walsh (1967), Theorem 3.11.1, p. 94. Cubic spline interpolators can *overshoot* in order to always create a *visually pleasing curve*, and this might account for the fact that the values for gamma are larger than the exact gamma or even the Crank–Nicolson gamma.

When good approximations to the second derivatives of a smooth function are needed, we can first approximate the function by using the values of the first derivative as input in order to compute an approximation to the second derivative. This is called *spline-on-spline* computation (Hildebrand (1974), p. 492, and Alberg, Nilson and Walsh (1967), pp. 48–50), and the improvements in accuracy are visible. In the current case, a useful exercise would be to compare the values for gamma based on this variant with the original approach taken to produce the results in Table 16.1. We expect to get a better approximation to gamma.

16.5.3 Boundary Conditions

We have seen (in Duffy (2018)) how to approximate the first- and second-order derivatives of a function using cubic splines. The mathematical theory is discussed in Ahlberg, Nilson and Walsh (1967). Spline approximation is useful for high-precision numerical differentiation when sufficiently accurate data are available. The cubic polynomial has four unknown constants in each subinterval, hence there are $4n$ constants to be evaluated in total. The following continuity conditions at the interior mesh points allow us to find $3(n - 1) + n + 1$ (in total $4n - 2$) of these constants as can be checked from the following relationships:

$$\left\{ \begin{array}{l} S(x_j^-) = S(x_j^+) \\ S'(x_j^-) = S'(x_j^+) \\ S''(x_j^-) = S''(x_j^+) \end{array} \right\} 1 \leq j \leq n - 1$$

$$S(x_j) = y_j, 0 \leq j \leq n.$$

We still need to find two extra conditions. To this end, one of the following sets of boundary conditions must be satisfied:

- a. *Clamped spline*: $S'(x_0) = y'_0$, $S'(x_n) = y'_n$.
- b. *Curved-adjusted cubic spline*: $S''(x_0) = y''_0$, $S''(x_n) = y''_n$.
- c. *Periodic spline*: $S(x_0) = S(x_n)$, $S'(x_0) = S'(x_n)$, $S''(x_0) = S''(x_n)$.

Periodic splines are not discussed in this book. Incidentally, cubic splines can be used to as a numerical integrator of functions based on Equations (16.17) and (16.18).

16.6 SOME COMPLEX FUNCTION THEORY

This section may be skipped on a first reading without loss of continuity.

In Section 16.7 of this chapter we shall introduce the *Complex Step Method* (CSM) that numerically computes the first derivative of a real-valued function by using functions of a complex variable. For this reason and because complex analysis is used in several areas of computational finance, we have decided to give a compact and precise introduction to analytic functions and some of their properties. Furthermore, most standard (undergraduate) textbooks on numerical analysis focus mainly on the approximation of real-valued functions, and this is another reason to discuss functions of complex variables.

We are mostly concerned with the class of *analytic functions* of a complex variable. Let Ω be a non-empty connected open subset of the complex plane. Then the function f is *analytic* (also known as *regular*, *holomorphic*) at the point $z_0 \in \mathbb{C}$ if the following limit exists:

$$\lim_{z \rightarrow z_0} \frac{f(z) - f(z_0)}{z - z_0} \equiv f'(z_0)$$

or alternatively:

$$\lim_{\Delta z \rightarrow 0} \frac{f(z + \Delta z) - f(z)}{\Delta z} \equiv f'(z). \quad (16.22)$$

An *entire function* is one that is analytic in the complex plane.

Some examples are:

- The functions e^z , $\sin z$, $\cos z$ are entire.
- The function $w(z) = \frac{1+z}{1-z}$, $\frac{dw}{dz} = \frac{2}{(1-z)^2}$ is analytic for $z \neq 1$ ($z = 1$ is *singular point*).
- The complex conjugate $\bar{z} = x - iy$ ($z = x + iy$) is not analytic anywhere (the limit depends on the manner in which $\Delta z \rightarrow 0$).
- Polynomials of all orders of a complex argument are analytic.

The *Cauchy–Riemann* (CR) conditions are necessary for a function $w = f(z) = u(x, y) + iv(x, y)$ ($z = x + iy$) to be analytic, that is:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}, \quad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}. \quad (16.23)$$

If these partial derivatives are continuous, then CR conditions are sufficient for f to be analytic (u and v are called *conjugate functions*).

16.6.1 Curves and Regions

A *continuous curve* or arc in the complex plane can be represented by the *parametric equation*:

$$z = z(t) = x + iy = \phi(t) + i\psi(t) \quad (16.24)$$

where $\phi(t)$ and $\psi(t)$ are real functions of the real variable t and these functions are assumed to be continuous in the interval $t_1 \leq t \leq t_2$. The curve joins the end points $a = z(t_1)$ and $b = z(t_2)$. The curve is said to be *closed* if the endpoints coincide, that is $a = b$. A *simple closed curve* is a closed curve that does not intersect itself. A smooth curve or arc is one for which $\phi(t)$ and $\psi(t)$ have continuous derivatives. A piecewise or *sectionally smooth curve* (also known as a *contour*) is composed of a finite number of smooth arcs. For example, the boundary of a rectangle is a piecewise smooth curve.

We can subdivide a curve C into n parts in much the same way as in the case of real-valued functions. We form a *partition* as follows:

$$z(t_1) = a = z_0 z_1 \dots z_{n-1} z_n = b = z(t_2). \quad (16.25)$$

We now form the sum:

$$S_n(f) \equiv \sum_{j=1}^n f(\xi_j)(z_j - z_{j-1}) = \sum_{j=1}^n f(\xi_j)\Delta z_j \quad (16.26)$$

$(z_{j-1} \leq \xi_j \leq z_j, \Delta z_j \equiv z_j - z_{j-1}$ is the *chord length*).

When this sum approaches a limit as the *chord length* approach zero (and which does not depend on the mode of subdivision), we can then denote this limit by:

$$\lim_{\Delta z_j \rightarrow 0} S_n(f) = \int_a^b f(z) dz \equiv \int_C f(z) dz \quad (16.27)$$

called the *complex line integral* or the *line integral along* the curve C . In this case we say that f is integrable along C . The special case when $\xi_j = z_j$ corresponds to the *Riemann–Stieltjes integral*. We take an example where $z_0 = \alpha$, $z_n = \beta$:

$$\int_C 1 dz = \lim \sum_{j=1}^n 1 \cdot (z_j - z_{j-1}) = z_n - z_0 = \beta - \alpha. \quad (16.28)$$

$$\int_C z dz = \lim \sum_{j=1}^n z_j (z_j - z_{j-1}) = \lim \sum_{j=1}^n z_{j-1} (z_j - z_{j-1}). \quad (16.29)$$

Adding the last two terms in (16.29) results in a *collapsing sum*, and we finally get:

$$2 \int_C z dz = \lim (z_n^2 - z_0^2) = \beta^2 - \alpha^2. \quad (16.30)$$

Generalising, we can show by forming a collapsing sum that:

$$\int_C z^n dz = \frac{\beta^{n+1} - \alpha^{n+1}}{n+1}, \quad n \geq 0. \quad (16.31)$$

Formula (16.26) is used in when approximating the inverse Fourier transform (which is the integral of a complex-valued function), for example when computing option price based on the characteristic function.

A region R in the complex plane is said to be *simply-connected* if any simple closed curve which lies in R can be shrunk to a point without leaving R . A *multiply-connected region* is one that is not simply-connected.

Finally, a *Jordan curve* divides the plane into a bounded region (the interior or inside of the curve) and an unbounded region (the exterior or outside of the curve), satisfying $|z| < M$ and $|z| > M$, respectively, for some positive M . An example of a bounded region is the open circular disc (ball) with centre a and radius r , that is $B(a; r) = \{z \in \mathbb{C} : \|z - a\| < r\}$.

16.6.2 Taylor's Theorem and Series

Taylor's theorem states that if $f(z)$ is analytic inside a circle C with centre at a , then for all z inside C we have:

$$f(z) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (z - a)^n. \quad (16.32)$$

A point z at which a function $f(z)$ fails to be analytic is called a *singular point* or *singularity* of $f(z)$. The region of convergence of the series (16.32) is given by $|z - a| < R$ where the radius of convergence R is the distance from a to the nearest singularity of the function $f(z)$. On $|z - a| = R$ the series may or may not converge. For $|z - a| > R$ the series converges. The series (16.32) converges for all z if the nearest singularity of $f(z)$ is at infinity. In general, we prove convergence using the *ratio test* or the *root test*.

A special case of (16.32) is given by:

$$f(x_0 + ih) = f(x_0) + ih f'(x_0) + O(h^2). \quad (16.33)$$

Rearranging terms we finally get the approximate identity:

$$f'(x_0) \sim \text{Im}(f(x_0 + ih)/h) \quad (16.34)$$

where Im is the imaginary part of $f(z)$.

Approximation (16.34) is the Complex Step Method *in hiding* (see Section 16.7). The current section has justified the method mathematically.

To make things concrete, we finish this section with a simple example on how to apply Taylor's theorem and compute the radius of convergence:

$$f(z) = e^{-z}, \quad a = 0.$$

$$f'(0) = -1, \quad f''(0) = 1, \quad f^{(n)}(0) = (-1)^n, \quad n \geq 1$$

Then:

$$e^{-z} = \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} z^n \equiv \sum_{n=0}^{\infty} a_n.$$

We show that the radius of convergence is infinite by applying the *ratio test* to prove that this series is absolutely convergent:

$$\begin{aligned} \lim_{n \rightarrow \infty} \left| \frac{a_{n+1}}{a_n} \right| &= \left| \frac{(-1)^{n+1} z^{n+1} n!}{(-1)^n (n+1)! z^n} \right| \\ &= \lim_{n \rightarrow \infty} \left| \frac{z}{n+1} \right| = 0. \end{aligned}$$

16.6.3 Laurent's Theorem and Series

Laurent's theorem states that if $f(z)$ is analytic inside and on the boundary of the ring-shaped ('donut') region R bounded by two concentric circles C_1 (radius r_1) and C_2 (radius r_2) ($r_1 > r_2$), then:

$$f(z) = \sum_{n=0}^{\infty} a_n (z - a)^n + \sum_{n=1}^{\infty} \frac{a_{-n}}{(z - a)^n} \quad \forall z \in \mathbb{R}$$

where:

$$\begin{aligned} a_n &= \frac{1}{2\pi i} \int_{C_1} \frac{f(\zeta)}{(\zeta - a)^{n+1}} d\zeta, \quad n = 0, 1, 2, 3, \dots \\ a_{-n} &= \frac{1}{2\pi i} \int_{C_2} \frac{f(\zeta)}{(\zeta - a)^{-n+1}} d\zeta, \quad n = 1, 2, 3, \dots \end{aligned} \tag{16.35}$$

The first part in (16.35) is called the *analytical part* of the Laurent series, while the remainder of the series consisting of negative powers of $\zeta - a$ is called the *principal part*. The Laurent series reduces to the Taylor series (16.32) when the principal part is zero.

We now take an example of the Laurent series for the function:

$$f(z) = \frac{e^{2z}}{(z-1)^3}.$$

In this case we see that the function has a *pole* of order 3 at $z = 1$, hence $\lim_{z \rightarrow 1} f(z) = \infty$. We now compute the Laurent's series for this function:

Let $z - 1 = u$, then $z = 1 + u$ and

$$\begin{aligned} \frac{e^{2z}}{(z-1)^2} &= \frac{e^{2+2u}}{u^3} = \frac{e^2}{u^3} \left\{ 1 + 2u + \frac{(2u)^2}{2!} + \dots \right\} \\ &= \frac{e^2}{(z-1)^3} + \frac{2e^2}{(z-1)^2} + \frac{2e^2}{z-1} + \dots \end{aligned}$$

We thus see that $z = 1$ is a pole of order 3, or a *triple pole*.

16.6.4 Cauchy–Goursat Theorem

Cauchy's theorem states that:

$$\int_C f(z) dz = 0 \quad (16.36)$$

if $f(z)$ is analytic and if $f'(z)$ is continuous at all points inside and on a simple closed curve C . We can prove this theorem by first writing:

$$\begin{aligned} f &= u + iv \\ z &= x + iy \Rightarrow dz = dx + idy \end{aligned}$$

and then using Green's theorem and the Cauchy–Riemann equations to get the desired result (see Figure 16.2):

$$\begin{aligned} \int_C f(z) dz &= \int_C (u + iv)(dx + idy) \\ &= \int_C u dx - v dy + i \int_C v dx + u dy \\ &= \int_R \left(-\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right) dx dy + i \int_R \left(\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} \right) dx dy = 0. \end{aligned}$$

Both integrals are zero.

We state Green's theorem for completeness:

$$\int_R \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) dx dy = \int_C (P dx + Q dy)$$

for differentiable functions $P(x, y)$ and $Q(x, y)$.

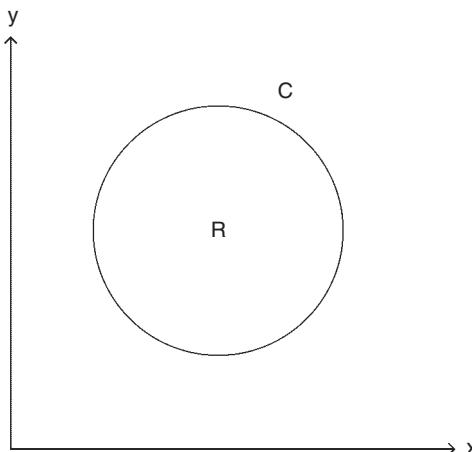


FIGURE 16.2 Region R and boundary curve C.

We can extend this result to multi-connected regions due to the fact that Green's theorem holds for such regions. The *Cauchy–Goursat* theorem removes the restriction that $f'(z)$ be continuous.

Morera's theorem states that if the above holds for every simple closed curve C in R , then $f(z)$ is analytic in R .

16.6.5 Cauchy's Integral Formula

This formula states that if $f(z)$ is analytic inside and on a simple closed curve C and z is any point inside C , then:

$$f(z) = \frac{1}{2\pi i} \int_C \frac{f(\zeta)}{\zeta - z} d\zeta. \quad (16.37)$$

We can use (16.37) to show that the derivative of an analytic function is also an analytic function. Then:

$$\frac{f(z+h) - f(z)}{h} = \frac{1}{2\pi i} \int_C \frac{f(\zeta)}{(\zeta - z)^2} d\zeta + I(h) \text{ where } \lim_{h \rightarrow 0} I(h) = 0 \quad (16.38)$$

and finally we get (Nehari (1975)):

$$f'(z) = \lim_{h \rightarrow 0} \frac{f(z+h) - f(z)}{h} = \frac{1}{2\pi i} \int_C \frac{f(\zeta)}{(\zeta - z)^2} d\zeta. \quad (16.39)$$

Similarly, we can prove that:

$$f''(z) = \frac{2}{2\pi i} \int_C \frac{f(\zeta)}{(\zeta - z)^3} d\zeta$$

and in general the *Cauchy integral formula* is:

$$f^{(n)}(z) = \frac{n!}{2\pi i} \int_C \frac{f(\zeta)}{(\zeta - z)^{n+1}} d\zeta. \quad (16.40)$$

We take an example. Consider the function $f(z) = \frac{e^z}{z-2}$. The following two results apply Cauchy's integral formula on circles of radius 3 and 1, respectively:

$$\frac{1}{2\pi i} \int_C f(z) dz = e^2 \text{ on } C = \{z : |z| = 3\} \quad (16.41)$$

$$\frac{1}{2\pi i} \int_C f(z) dz = 0 \text{ on } C = \{z : |z| = 1\}. \quad (16.42)$$

16.6.6 Cauchy's Residue Theorem

We already know from Section 16.6.3 that an analytic function $f(z)$ that is single-valued in a region R and is regular in R except at a point $z = a$ in R can be expanded the vicinity of $z = a$ into a converging Laurent series of the form (16.35). Of particular interest is the coefficient:

$$a_{-1} = \frac{1}{2\pi i} \int_C f(z) dz \quad (16.43)$$

which is called the *residue* of $f(z)$ at the singular point $z = a$. Equation (16.43) in fact states that the residue is the integral of $f(z)$ over a closed contour surrounding this singular point. This result can be generalised to the case with n singular points a_1, a_2, \dots, a_n inside C .

How do we calculate residues? Is it not necessary to compute (16.43) directly. Instead, for a pole of order k there is a simple formula:

$$a_{-1} = \lim_{z \rightarrow a} \frac{1}{(k-1)!} \frac{d^{k-1}}{dz^{k-1}} \{(z-a)^k f(z)\}. \quad (16.44)$$

For a *simple pole* ($k = 1$), the formula is particularly easy:

$$a_{-1} = \lim_{z \rightarrow a} (z-a) f(z). \quad (16.45)$$

When there are n singular points, the residue theorem is:

$$\begin{aligned} & \text{residues } a_{-1}, b_{-1}, c_{-1}, \dots \\ & \int_C f(z) dz = 2\pi i (a_{-1} + b_{-1} + c_{-1} + \dots). \end{aligned} \quad (16.46)$$

An example is:

$$\begin{aligned} f(z) &= \frac{z}{(z-1)(z+1)^2} \\ z = 1 &: \text{pole of order 1, residue } = \frac{1}{4} \\ z = -1 &: \text{pole of order 2, residue } = \frac{1}{4} \end{aligned} \quad (16.47)$$

and hence we get:

$$\int_C f(z) dz = 2\pi i \left(\frac{1}{4} - \frac{1}{4} \right) = 0. \quad (16.48)$$

We leave it to the reader to check (16.47) and (16.48) as well as (16.41) and (16.42).

16.6.7 Gauss's Mean Value Theorem

Gauss's mean value theorem states that:

$$f(a) = \frac{1}{2\pi} \int_0^{2\pi} f(a + re^{i\theta}) d\theta \quad (16.49)$$

where $f(z)$ is analytic inside and on a circle C with radius r and centre at a . In other words, the value $f(a)$ is the *mean* of the values of $f(z)$ on C . The steps in proving (16.49) are:

$$\begin{aligned} |z - a| = r \Rightarrow z = a + re^{i\theta}, \quad \frac{dz}{d\theta} = ire^{i\theta} \\ f(a) = \frac{1}{2\pi i} \int_C \frac{f(z) dz}{z - a} = \frac{1}{2\pi i} \int_0^{2\pi} \frac{f(a + re^{i\theta}) ire^{i\theta} d\theta}{re^{i\theta}} = \frac{1}{2\pi} \int_0^{2\pi} f(a + re^{i\theta}) d\theta. \end{aligned}$$

A good introduction to complex variable theory is Spiegel (1999).

16.7 THE COMPLEX STEP METHOD (CSM)

The idea of using the Cauchy integral formula (16.40) to numerically compute the derivatives of a complex-valued function dates back to Lyness and Moler (1967). Their method is essentially to approximate (16.40) using the trapezoid quadrature rule based on the analysis we performed to motivate Equation (16.26). To this end, we divide the interval $(0, 2\pi)$ into m subintervals, and the quadrature rule becomes:

$$f^{(n)}(z_0) = \frac{n!}{mr} \sum_{j=0}^{m-1} \frac{f\left(z_0 + r \exp\left(i \frac{2\pi j}{m}\right)\right)}{\exp\left(i \frac{2\pi j n}{m}\right)} \quad (16.50)$$

where we use the same notation as in Formula (16.26). In particular, we can approximate the first-order derivative as:

$$f^{(1)}(z_0) = \frac{1}{mr} \sum_{j=0}^{m-1} \frac{f\left(z_0 + \exp\left(i \frac{2\pi j}{m}\right)\right)}{\exp\left(i \frac{2\pi j}{m}\right)}. \quad (16.51)$$

We do not discuss this method further. We would require a large number of function evaluations to achieve a high accuracy when calculating the contour integral. Furthermore, the method is somewhat prone to rounding error.

A method for calculating first-order derivatives is the *Complex Step Method* (CSM) (Squire and Trapp (1998)). It is based on the fact that Taylor's theorem is valid for complex increments, as we saw from (16.33). The working formula is (16.34). It involves the evaluation of the function for a complex argument, but there is no subtractive cancellation error as witnessed when using divided differences. We do have to formulate

the problem in terms of functions of a complex variable, and this means that C++ and Python will need *complexified* versions of real-valued functions whose derivatives we wish to compute. Convergence is *quadratic*, and a very small step size h can be used, even to machine precision level. At some stage, decreasing h beyond a certain threshold value has no effect on accuracy.

It is also possible to compute higher-order derivatives using CSM, but subtractive cancellation errors can appear and thus we do not recommend these formulae in general. Some second-order formulae are (where $\text{Im } f(z)$ is the imaginary part of $f(z)$):

$$\begin{aligned}f''(x) &= \frac{2(f(x) - \text{Im } f(x - ih))}{h^2} + 0(h^2) \\f''(x) &= \frac{2(f(x) - \text{Im } f(x + ih))}{h^2} + 0(h^2).\end{aligned}\tag{16.52}$$

and an example of a fourth-order formula is:

$$f''(x) = \frac{f(x + h) + f(x - h) - 2\text{Im } f(x + ih)}{2h^2}.\tag{16.53}$$

In general, the steps to help implement CSM are:

1. Write a common template function that subsumes both the complex and double cases. Alternatively, write a separate function for the complex case and a separate function for the double case. But this is code duplication in essence, and the code can become unmaintainable.
2. Use the fact that modern C++ and Boost C++ have libraries for mathematical functions with complex arguments and return types.
3. Create a complex argument $z = x + ih$, ($i = \sqrt{-1}$); x is the value where we wish to compute the derivative and $h > 0$ is the step size.
4. Call the function and take the imaginary part of its return type.

An interesting exercise is to compare using formulae (16.52) with using (16.9)(a).

It is very easy to program formula (16.34) in both C++ and Python. We define a function type and the code for (16.34) as follows:

```
// Notation and function spaces
using value_type = double;

template <typename T>
using FunctionType = std::function<T (const T& arg)>;
using CFunctionType = FunctionType<std::complex<value_type>>;
```

and the implementation is:

```
value_type CSM(const CFunctionType& f, value_type x, value_type h)
{ // df/dx at x using the Complex step method

    std::complex<value_type> z(x, h); // x + ih, i = sqrt(-1)
    return std::imag(f(z)) / h;
}
```

We now take the standard reference example (Lyness and Moler (1967), Squire and Trapp (1998)) to test our code against. We take the derivative at $x = 1.5$, and it should be 3.62203:

$$\frac{e^x}{\sin(x)^3 + \cos(x)^3}, \quad x \in \mathbb{R}. \quad (16.54)$$

In this case we see that it would be quite laborious to compute the derivative analytically.

The code and test case is:

```
// Test case from Squire&Trapp 1998
template <typename T> T SquireTrapp(const T& t)
{
    T n1 = std::exp(t);
    T d1 = std::sin(t);
    T d2 = std::cos(t);

    return n1 / (d1*d1*d1 + d2*d2*d2);
}
// Squire Trapp
double x = 1.5;      double h = 0.1;
do
{
    std::cout << std::setprecision(12)
        << CSM(SquireTrapp<std::complex<value_type>>, x, h);
    h *= 0.1;
} while (h > 1.0e-300);
```

As a final sanity check, we examine the function:

$$f(z) = \frac{e^z}{z}, \text{ then } f'(z) = \frac{e^z(z - 1)}{z}. \quad (16.55)$$

When we apply (16.34) to this function we get:

$$\frac{\operatorname{Im}f(x + ih)}{h} = \frac{e^x \left(-\cos(h) + \frac{\sin(h)x}{h} \right)}{x^2 + h^2} \quad (z = x + ih). \quad (16.56)$$

By taking h to be 0, we see that the analytical and approximate results for the derivative coincide. To this end, we can check the limit:

$$\lim_{h \rightarrow 0} \frac{e^x \left(-\cos(h) + \frac{\sin(h)x}{h} \right)}{x^2 + h^2} = \frac{e^x(x - 1)}{x^2}.$$

16.7.1 Caveats

Ideally, introducing CSM into existing code should be as seamless as possible. This is not always possible because we must *complexify* functions of a real variable. This can introduce coding errors into the program. The combination of C++ and Boost C++ libraries should help us in writing complexified functions. However, there are situations in which we must write our own specialised functions or use some third-party library. Some useful prerequisite knowledge of complex analysis (in addition to the topics that we discussed in this chapter) are:

1. Complex numbers.
2. Functions of a complex variable; limits and continuity.
3. Complex differentiation and integration.
4. Single-valued and multiple-valued functions.
5. Elementary functions (exponential, logarithm, trigonometric, hyperbolic, inverse trigonometric and hyperbolic functions).
6. Knowing the applications in computational finance.

Complex variables and related methods are used in several areas of computational finance, for example Fourier transform theory and its application to option pricing.

As a final counterexample, let us take the following standard definition of the sine function:

$$\sin(z) = \frac{e^{iz} - e^{-iz}}{2i}. \quad (16.57)$$

If $z = x + ih$, then CSM gives:

$$Im[\sin(x + ih)] = \cos(x) \left(\frac{e^h - e^{-h}}{2} \right). \quad (16.58)$$

This representation is problematic when h is very small because subtraction of the two very small numbers e^h and e^{-h} leads to subtractive cancellation. A better representation is:

$$\sin(x + ih) = \sin(x) \cosh(h) + i \cos(x) \sinh(h) \quad (16.59)$$

that leads to:

$$\sin(x + ih) \approx \sin(x) + ih \cos(x). \quad (16.60)$$

Taking the imaginary path and using formula (16.34) gives us the answer.

16.8 SUMMARY AND CONCLUSIONS

In this chapter we introduced the important area of *sensitivity analysis* that is concerned with the rate of change of the solution of a problem in continuous space with respect to one or more of its underlying parameters. There are many applications; the main ones in this book are concerned with the computation of option sensitivities (also known as the *greeks*).

The methods that we introduced were:

- Exact formulae for the greeks.
- Computing sensitivities by divided differences.
- Computing sensitivities by piecewise cubic splines.
- Complex Step Method (CSM).

We also gave an introduction to what can best be described as a mini-course on complex variable theory and some applications that crop in computational finance.

We continue with our discussion of sensitivity analysis in Chapter 17.

CHAPTER 17

Advanced Topics in Sensitivity Analysis

A good technical writer, trying not to be obvious about it, but says everything twice: formally and informally. Or maybe three times.

Donald Knuth

17.1 INTRODUCTION AND OBJECTIVES

In this chapter we continue with our discussion on sensitivity analysis that we introduced in Chapter 16. We now focus on a method to compute sensitivity using the so-called *continuous sensitivity equation* (CSE) method as well as automatic differentiation (AD). A CSE can be an algebraic equation, an ordinary differential equation, a partial differential equation or a stochastic differential equation.

The distinguishing feature of the methods to compute sensitivities in this chapter is that we can “differentiate a PDE”, to provide us with a new PDE that describes how the particular sensitivity variable evolves in time and space. This approach can be applied to ordinary differential equations (ODEs), stochastic differential equations (SDEs) and partial differential equations (PDEs), and the formulation reduces to a study of an initial value problem or an initial boundary value problem, topics that have already been discussed in detail in this book.

We also give an introduction to *automatic differentiation* (AD) including a code example in C++.

17.2 EXAMPLES OF CSE

In this section we examine a number of specific problems in order to motivate this subject. To this end, we are interested in equations and algorithms to compute a function f that depends on state variables x and certain parameters β :

$$f(x; \beta), \quad x = (x_1, \dots, x_n), \quad \beta = (\beta_1, \dots, \beta_m), \quad x \in \mathbb{R}^n, \quad \beta \in \mathbb{R}^m. \quad (17.1)$$

We shall see that the unknown function f can be the solution of any one of the following kinds of equations:

- Ordinary differential equation (ODE)
- Stochastic differential equation (SDE)
- Partial differential equation (PDE)
- Algebraic equation.

It is well known how to formulate these equations in order to unambiguously specify f and how to approximate these equations numerically. The new challenge here, however, is to find ways to describe and approximate the first-order derivatives of f with respect to its state variables and parameters. In other words, we wish to approximate the following quantities or *sensitivities*:

$$\begin{aligned} \frac{\partial f}{\partial x_j}, \quad j = 1, \dots, n & \text{ (state variables)} \\ \frac{\partial f}{\partial \beta_j}, \quad j = 1, \dots, m & \text{ (parameters).} \end{aligned} \tag{17.2}$$

In general, there are two approaches to computing the quantities in (17.2) when using a continuous sensitivity equation (CSE) approach:

- *Differentiate-then-approximate*: We derive the CSE and we apply the solution technique, for example we differentiate the Black–Scholes PDE with respect to the stock to produce a PDE for the delta that we can then approximate by the finite difference method.
- *Approximate-then-differentiate*: This is the typically the approach with automatic differentiation (AD). For example, we apply the Euler–Maruyama method to the SDE describing geometric Brownian motion (GBM) that results in a discrete set of equations in the state variables. We can now differentiate these equations with respect to a given parameter, for example the volatility. The resulting equation can now be solved by an appropriate analytical or numerical method.

We introduce a number of motivational examples. To paraphrase Paul Halmos, ‘the source of all great mathematics is the special case, the concrete example. It is frequent in mathematics that every instance of a concept of seemingly great generality is in essence the same as a small and concrete special case’. In particular, we wish to discover patterns in these simple examples that we also see in more complex applications.

17.2.1 Simple Initial Value Problem

We take the initial value problem for the scalar ODE:

$$\frac{dy}{dt} + ay = 0, \quad t > 0, \quad y(0) = A \tag{17.3}$$

having the solution:

$$y(t) = Ae^{-at}. \quad (17.4)$$

In this case we have one state variable t and two parameters A and a , and in this case we can write the solution symbolically as $y(t; a, A)$. The sensitivities can be analytically computed as follows by differentiating (17.4) with respect to the state variable and parameters. In this case exact solutions are known.

$$\begin{aligned} u &\equiv \frac{dy}{dt} = -aAe^{-at} \\ v &\equiv \frac{dy}{da} = -tAe^{-at} \\ w &= \frac{dy}{dA} = e^{-at}. \end{aligned} \quad (17.5)$$

Let's pretend now that we are unable or unwilling to find the explicit solutions (17.5). Instead, we rather wish to find the ODEs that the sensitivities satisfy. To this end, let us first differentiate (17.3) with respect to the parameter A to give:

$$\begin{aligned} \frac{dw}{dt} + aw &= 0 \\ w(0) &= \frac{du}{dA}(0) = 1. \end{aligned} \quad (17.6)$$

Similarly, the other CSEs are:

$$\begin{aligned} \frac{du}{dt} + au &= 0 \\ u(0) &= \frac{dy}{dt}(0) = -ay(0) = -aA \end{aligned} \quad (17.7)$$

and:

$$\begin{aligned} \frac{dv}{dt} + av &= -y = -Ae^{-at} \\ v(0) &= \frac{dy}{da}(0) = 0. \end{aligned} \quad (17.8)$$

We can find analytical solutions to these equations using the integrating factor method, but this is luck because (17.3) is so simple. In more complex cases we will need to employ numerical methods. Finally, we notice the *coupling* between the variables v and y in (17.8), which will also be seen in more complex applications. It is a pattern. It then becomes even more unlikely that an analytical solution can be found in general.

In many cases the steps that we learned here will be essentially the same for all kinds of problems.

17.2.2 Population Dynamics

We now consider a slightly extended example of (17.3). ODEs can be used as simple models of population growth: for example, assuming that the rate of reproduction of

a population size P is proportional to the existing population and to the amount of available resources. The ODE is:

$$\frac{dP}{dt} = rP \left(1 - \frac{P}{K}\right), \quad P(0) = P_0 \quad (17.9)$$

where r is the *growth rate* and K is the *carrying capacity*. The initial population is P_0 . It is easy to check the following identities:

$$P(t) = \frac{K P_0 e^{rt}}{K + P_0(e^{rt} - 1)} \text{ and } \lim_{t \rightarrow \infty} P(t) = K. \quad (17.10)$$

Transformation of this equation leads to the logistic ODE:

$$\frac{dn}{dt} = n(1 - n) \quad (17.11)$$

where n is the population in units of carrying capacity ($n = P/K$) and τ measures time in units of $1/r$.

Using the same approach as in Section 17.2.1, we can produce a system of ODEs for the population and its sensitivities. First, let us define the variables:

$$u_1(t) = x(t), u_2(t) = \frac{\partial x(t)}{\partial r}, u_3 = \frac{\partial x(t)}{\partial K}, u_4 = \frac{\partial x(t)}{\partial P_0}. \quad (17.12)$$

$$\begin{aligned} \frac{du_1(t)}{dt} &= ru_1(t) \left(1 - \frac{u_1(t)}{K}\right) \\ \frac{du_2(t)}{dt} &= \left(r - \frac{2r}{K}u_1(t)\right)u_2(t) + u_1(t) - \frac{1}{K}u_1^2(t) \\ \frac{du_3(t)}{dt} &= \left(r - \frac{2r}{K}u_1(t)\right)u_3(t) + \frac{r}{K^2}u_1^2(t) \\ \frac{du_4(t)}{dt} &= \left(r - \frac{2r}{K}u_1(t)\right)u_4(t) \end{aligned} \quad (17.13)$$

with the initial conditions:

$$u_1(0) = P_0, u_2(0) = u_3(0) = 0, u_4(0) = 1. \quad (17.14)$$

We are now ready to discuss how to solve Equations (17.13)–(17.14). We are lucky that the problem has an analytical solution (17.10). Alternatively, we can complexify the code that implements formula (17.10) and then apply the Complex Step Method (CSM) to compute the sensitivities:

```
std::complex<value_type> Verhulst(std::complex<value_type>& r,
                                     std::complex<value_type>& K,
                                     std::complex<value_type>& P0,
                                     value_type t)
{
    std::complex<value_type> ert = std::exp(r*t);
    std::complex<value_type> num = K * P0*ert;
    std::complex<value_type> den = K + P0*(ert - 1.0);

    return num / den;
}
```

We see that all input arguments are complex. When computing sensitivities, we write a function for each sensitivity, and then we add the increment separately to the parameter of interest while the other parameters are kept constant (that is, real values). An example that computes the sensitivity for the growth rate r is:

```
std::complex<value_type> SensitivityR std::complex<value_type>& r,
    std::complex<value_type>& K,      std::complex<value_type>& P0, value_type t)
{
    // Not optimised for performance
    value_type h = 1.0e-40; // hard-coded
    std::complex<value_type> z1(std::real(r), h);
    std::complex<value_type> z2(std::real(K), 0);
    std::complex<value_type> z3(std::real(P0), 0);
    std::complex<value_type> result = Verhulst(z1, z2, z3, t);

    return std::imag(result) / h;
}
```

An example of use is:

```
std::complex<value_type> r(0.7, 0.0);
std::complex<value_type> K(17.5, 0.0);
std::complex<value_type> P0(0.1, 0.0);
value_type t = 10.0; value_type h = 1.0;

std::cout << "\nR\n";
h = 1.0;
do
{
    std::cout << std::setprecision(12) << SensitivityR(r, K, P0, t) << ", ";
    h /= 2.0;
} while (h > 1.0e-9);
```

The answer is 20.68275. If you are feeling energetic, you can differentiate (17.10) with respect to r analytically and check the answer. We note that we must have an analytic representation (17.10) in order to apply CSM. On the other hand, the CSE approach does not need to know what the solution is, but we only need to define an ODE for it and its sensitivities as in system (17.13)–(17.14). In general, systems of ODEs do not have an analytical solution, and we can then resort to solvers to compute an approximate solution (Lambert (1991)). In Duffy (2018) we use the Boost C++ library to solve initial value problems such as (17.13)–(17.14). In general, the dependent variables are coupled, and this is also a common pattern when we apply CSE to PDE problems. Concluding, we see that we can compute the sensitivities for this problem using CSE, CSM and analytic differentiation. Another option would be divided differences.

17.2.3 Comparing CSE and Complex Step Method (CSM)

We now discuss the relative merits of CSM and CSE with a view to deciding which one is more suitable in a given context. We compare them based on the following metrics:

- M1: Ease of implementation.
- M2: Applicability and suitability for a range of applications.
- M3: Efficiency (run-time performance).
- M4: Applications to computational finance.

These metrics have wide applicability.

17.2.3.1 CSM

This method is easy to implement compared to CSE because we do not need to derive sensitivity equations. Computation time is acceptable if the number of parameters is not large compared to the dimension of the problem. CSM converges quadratically when the step size is smaller than a given threshold value. Approximations to first-order derivatives produce no cancellation errors, but this property does not hold for higher-order derivatives (we shall see that automatic differentiation software tools do not have these issues). Finally, we need to iterate the complex-step program for as many times as the number of parameters.

17.2.3.2 CSE

In this case we do not need to have a solution in analytic form. We create a program to create a new set of equations. In this book the sensitivity equations are either PDEs or SDEs. In general, the sensitivity equations do not have analytic solutions, but we can apply the finite difference method to approximate them. We need to ensure that these equations are well-posed.

17.3 CSE AND BLACK-SCHOLES PDE

Computing option sensitivities is very important in pricing and hedging applications, and having efficient and accurate algorithms to compute them is vital. In the case of the one-factor Black–Scholes equation, we have a special instance of the function in (17.1):

$$\begin{aligned} x &= (S, t) \quad (n = 2) \\ \beta &= (T, r, \text{div}, K, \sigma) \quad (m = 5) \\ V &= V(x; \beta) \text{ (option price).} \end{aligned} \tag{17.15}$$

In the current context, sensitivities are called *greeks*, and they can be classified by their highest-order derivative. Mixed derivatives in state variables and parameters are also used:

$$\text{Second-order greeks: } \frac{\partial^2 f}{\partial x_j^2}, \frac{\partial^2 f}{\partial \beta_k^2} \tag{17.16}$$

$$\text{Third-order greeks: } \frac{\partial^3 f}{\partial x_j^3}, \frac{\partial^3 f}{\partial \beta_k^3} \quad (17.17)$$

$$\text{Mixed greeks: } \frac{\partial^2 f}{\partial x_j \partial \beta_k}, \frac{\partial^3 f}{\partial x_j^2 \partial \beta_k}$$

$$j = 1, 2, \dots, k = 1, 2, \dots \quad (17.18)$$

The index range in these three categories is $j = 1, \dots, n, k = 1, \dots, m$. Some specific greeks are (see Haug (2007) for a comprehensive list):

First-order greeks:

$$\begin{aligned} \text{Delta: } & \frac{\partial C}{\partial S} \\ \text{Theta: } & -\frac{\partial C}{\partial T} \\ \text{Vega: } & \frac{\partial C}{\partial \sigma} \\ \text{Strike Delta: } & \frac{\partial C}{\partial K}. \end{aligned} \quad (17.19)$$

Second- and third-order greeks:

$$\begin{aligned} \text{Gamma: } & \frac{\partial^2 C}{\partial S^2} \\ \text{Volga: } & \frac{\partial^2 C}{\partial \sigma^2} \\ \text{Strike Gamma: } & \frac{\partial^2 C}{\partial K^2} \\ \text{Speed: } & \frac{\partial^3 C}{\partial S^3}. \end{aligned} \quad (17.20)$$

Mixed-order greeks:

$$\begin{aligned} \text{Vanna: } & \frac{\partial^2 C}{\partial S \partial \sigma} \\ \text{Charm: } & -\frac{\partial^2 C}{\partial S \partial T} \\ \text{Zomma: } & \frac{\partial^3 C}{\partial S^2 \partial \sigma} \\ \text{Colour: } & \frac{\partial^3 C}{\partial S^2 \partial t}. \end{aligned} \quad (17.21)$$

The approach that we now describe is applicable to a wide range of problems in finance, for example to fixed income problems, as we shall see in Chapter 25 when we discuss the CIR model. See also Robinson (2019).

17.3.1 Black–Scholes Greeks: Algorithms and Design

We scope the discussion by focusing on the one-factor Black–Scholes PDE and its associated initial condition (payoff) and boundary conditions. How do we apply CSE

in this case? Basically, we first differentiate both sides of the PDE with respect to the state variable or parameter in question. In many cases the resulting PDE is one of convection-diffusion-reaction type whose coefficients may or may not be the same as those of the original PDE. But in general, the highest-order diffusion term will have the same form in all cases. The convection, reaction and inhomogeneous term will not necessarily be the same. Furthermore, the new PDE will contain an approximation to source term in general, possibly depending on other sensitivities. We thus see that the PDE will be coupled to other PDEs. We must also address the issues of finding the initial and boundary conditions associated with the new PDE. In short, we have an initial boundary value problem for the sensitivity variable (or greek).

We must be careful that differentiation of the PDE does not lead to pathological situations such as coefficient blowup, nonexistence or non-uniqueness of solutions. In general, the initial boundary value problem should be well-posed. We note that we can differentiate the Black-Scholes equation any number of times to get PDEs for any of the greeks in Equations (17.19)–(17.21). We have implemented many of them as PDEs in C++, and we compared the results in Haug (2007). A full discussion is outside the current scope. We even computed speed using this approach.

Having formed the PDE for a greek, we can analyse it like any other PDE by applying the mathematical and numerical methods from this book:

- Domain truncation or domain transformation.
- Using Fichera theory to help find boundary conditions.
- Approximating the PDE using the finite difference method.
- Ensure that approximations to inhomogeneous terms are accurate. These terms must be approximated as part of the CSE formulation (they have their own PDE) and not as divided difference approximations.

The main challenge is that the PDEs can depend on other PDEs. We will need to solve coupled PDE systems.

17.3.2 Some Specific Black-Scholes Greeks

We recall the Black-Scholes PDE:

$$\frac{\partial V}{\partial t} = \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV.$$

The PDE that we discuss is for delta $\Delta = \frac{\partial C}{\partial S}$ (using domain truncation). The initial boundary value problem for a call option is:

$$\frac{\partial \Delta}{\partial t} = \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 \Delta}{\partial S^2} + (\sigma^2 + r)S \frac{\partial \Delta}{\partial S} \quad (17.22)$$

$$\Delta(0, t) = 0, \quad \Delta(S_{\max}, t) = 1 \quad (17.23)$$

$$\Delta(S, 0) = \begin{cases} 1, & S > K \\ 0, & S < K \end{cases} \quad (\text{Heaviside function}). \quad (17.24)$$

The rationale is: differentiate the Black–Scholes PDE with respect to S to get (17.22) and differentiate the call payoff with respect to S to get (17.24), and then imposing continuity at $S = 0$ and $S = S_{\max}$ delivers the boundary conditions (17.23).

The initial boundary value problem for gamma $\Gamma = \frac{\partial^2 C}{\partial S^2} = \frac{\partial \Delta}{\partial S}$ is:

$$\frac{\partial \Gamma}{\partial t} = \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 \Gamma}{\partial S^2} + (2\sigma^2 S + rS) \frac{\partial \Gamma}{\partial S} + (\sigma^2 + r)\Gamma \quad (17.25)$$

$$\Gamma(0, t) = \Gamma(S_{\max}, t) = 0 \quad (17.26)$$

$$\Gamma(S, 0) = \delta(S - K) \approx \frac{2\lambda e^{-2\lambda(S-K)}}{(e^{-2\lambda(S-K)} + 1)^2} \quad (17.27)$$

(λ is steepness of the logistical curve).

Equation (17.25) can most easily be found by differentiating (17.22) with respect to S . We notice that the initial condition (or payoff) for gamma is a delta function. We need to have an accurate approximation to it.

Finally, the initial boundary value problem for vega $\nu = \frac{\partial V}{\partial \sigma}$ is:

$$\frac{\partial \nu}{\partial t} = \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 \nu}{\partial S^2} + rS \frac{\partial \nu}{\partial S} - r\nu + \sigma S^2 \Gamma \quad (17.28)$$

$$\nu(0, t) = \Delta(0, t), \nu(S_{\max}, t) = \Delta(S_{\max}, t) \quad (17.29)$$

$$\nu(S, 0) = 0. \quad (17.30)$$

We see that the PDE (17.28) contains a source term involving gamma. Approximating gamma by divided differences is less robust than treating it as the solution of systems (17.25)–(17.27). We also need to design good approximations to the delta function. Finally, the other greeks can be computed as solutions of initial boundary value problems. An interesting exercise for the reader is to compute speed $\frac{\partial^3 C}{\partial S^3}$ using this approach. (Hint: use Equation (17.25).)

17.4 USING OPERATOR CALCULUS TO COMPUTE GREEKS

We give a short overview of the PDE-based method in Carr (2001) for computing the greeks of path-independent claims in the Black–Scholes model. The method is based on using spatial derivatives as input to the calculation of other partial derivatives and the assumption that every derivative can be regarded as the arbitrage-free value of a certain quantoed contingent claim. The analysis relies on the application of the *operator calculus* and multivariate Taylor series expansions. A consequence of using series is that we need to determine when they converge. In particular, we determine the series' *radius of convergence*, and we also need to work with truncated versions of these series. Finally, we wish to avoid rounding errors when summing the series. This is an added complication from a numerical point of view.

Let us consider the Black–Scholes PDE again:

$$\frac{\partial V}{\partial t} = \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV. \quad (17.31)$$

Defining $x = \log S$ and $U = U(x, t)$, we then get the well-known transformed PDE:

$$\frac{\partial U}{\partial t} = \frac{1}{2}\sigma^2 \frac{\partial^2 U}{\partial x^2} + \left(r - \frac{1}{2}\sigma^2\right) \frac{\partial U}{\partial x} - rU \quad (17.32)$$

with initial condition:

$$U(x, 0) = f(e^x) \equiv \phi(x) \quad (f \text{ is payoff function}). \quad (17.33)$$

We can differentiate (17.32) n times with respect to x to give us $\frac{\partial^n U}{\partial x^n}$ that satisfies the same PDE as $U(x, t)$ having the initial condition:

$$\phi^{(n)}(x) \equiv \frac{d^n \phi(x)}{dx^n} \text{ at } t = 0. \quad (17.34)$$

Thus, $\frac{\partial^n U}{\partial x^n}$ can be interpreted as the value of a *contingent claim with payoff* (17.34). For example, the gamma of a claim can be represented as (Carr (2001)):

$$\frac{\partial^2 V}{\partial S^2} = e^{(r+\sigma^2)t} E^{(2)} [f''(S_T) | S_t = S] \quad (17.35)$$

where the operator $E^{(2)}$ states that the expectation of the final gamma $f''(S_T)$ is calculated from the GBM:

$$dS = (r + 2\sigma^2)Sdt + \sigma SdW. \quad (17.36)$$

Series for arbitrary greeks are given in Carr (2001). Care must be taken by the impatient to ensure that the series is convergent for a given set of parameters.

An interesting project could be to carry out a PDE analysis (as in Section 17.3) but then apply it to PDE (17.32). For example, can we differentiate (17.32) with respect to volatility σ to produce a PDE for vega?

17.5 AN INTRODUCTION TO AUTOMATIC DIFFERENTIATION (AD) FOR THE IMPATIENT

We now give an introduction to a relatively new approach to evaluate the derivative of a function. To this end, AD is based on the observation that every computer program (no matter how complex) executes a sequence of elementary arithmetic (unary and binary) operations (such as addition, subtraction, multiplication and division) in combination with elementary functions (such as exp, log, sin and so on). According to Wikipedia, ‘By applying the chain rule repeatedly to these operations, derivatives of arbitrary order

can be computed automatically, accurately to working precision, and using at most a small constant factor more arithmetic operations than the original program'.

As an initial example, consider computing the composite function (taken from Nocedal and Wright (2006)):

$$f(x) = (x_1 x_2 \sin x_3 + e^{x_1 x_2}) / x_3, \quad x = (x_1, x_2, x_3) \in \mathbb{R}^3. \quad (17.37)$$

We can break this formula down into simpler parts by defining new variables as follows:

$$\begin{aligned} x_4 &= x_1 * x_2; \quad x_5 = \sin x_3 \\ x_6 &= e^{x_4}; \quad x_7 = x_4 * x_5 \\ x_8 &= x_6 + x_7; \quad x_9 = x_8 / x_3. \end{aligned} \quad (17.38)$$

The final node x_9 contains the function value $f(x)$ that we desire. The structure (17.38) can be cast in the language of *graph theory*. In particular, we define a *computational graph* for the function $f(x)$ as shown in Figure 17.1. In this graph, vertices hold variables, while directed arcs between vertices represent operations. We say that a vertex i is a *parent* of vertex j (and j is a *child* of i) if there is a directed arc from i to j . We can compute the value at a vertex when the values of all its parents are known. Thus, computation flows through the graph from left to right. For example, the child x_9 is computed in terms of its parent x_8 . The flow in this direction is known as a *forward sweep*.

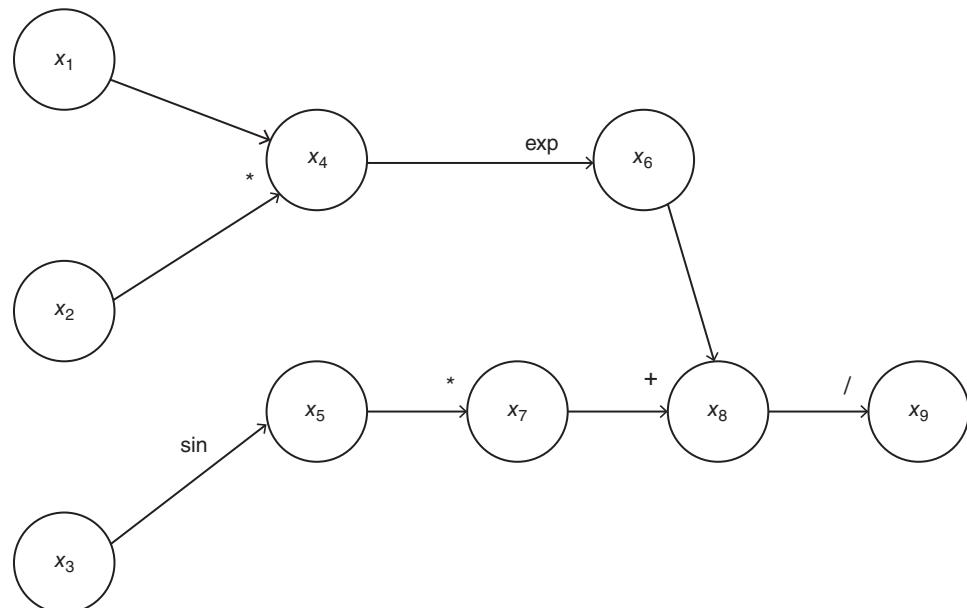


FIGURE 17.1 Example with three variables.

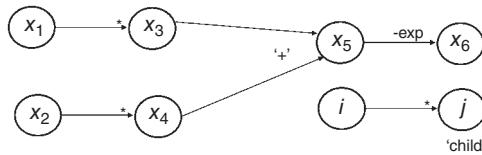


FIGURE 17.2 Example with two variables.

We take another example:

$$f(x) = e^{-(x_1^2 + x_2^2)}, \quad x = (x_1, x_2) \in \mathbb{R}^2. \quad (17.39)$$

The intermediate nodes are:

$$\begin{aligned} x_3 &= x_1 * x_1; \quad x_4 = x_2 * x_2 \\ x_5 &= x_3 + x_4; \quad x_6 = e^{-x_5}. \end{aligned} \quad (17.40)$$

and the computational graph is shown in Figure 17.2.

We note that the examples just given were for motivational purposes; when trying to understand a new concept or software library, it is always useful to take simpler examples and solve them without the aid of a computer, for example using pencil and paper. Once you understand how the process works, then it is time to use a software tool on bigger problems. Software tools for automatic differentiation do not require the user to break down the code for evaluating the function into its elements. These tools identify the intermediate quantities and they construct the computational graph.

17.5.1 What Is Automatic Differentiation: The Details

Automatic differentiation is a generic name for techniques that use the computational representation of a function to produce analytic values for the derivatives. We have already discussed the chain rule in Chapter 1, and we now apply it to Example (17.39) and Figure 17.2. In the *forward mode* we compute the *directional derivative* of each child vertex based on its parent vertices by using the chain rule:

$$\begin{aligned} \frac{\partial x_4}{\partial x_2} &= 2x_2 \\ \frac{\partial x_3}{\partial x_1} &= 2x_1 \\ \frac{\partial x_5}{\partial x_3} &= 1 \\ \frac{\partial x_6}{\partial x_5} &= -e^{-x_5}. \end{aligned} \quad (17.41)$$

In the *reverse mode* we recover the derivatives of the function with respect to each parent variable, including independent (input) and intermediate variables.

When the sweep has concluded, we will then have the derivative for the independent input variables.

Appealing to the chain rule in the current context:

$$\frac{\partial f}{\partial x_i} = \sum_{j \text{ child of } i} \frac{\partial f}{\partial x_j} \cdot \frac{\partial x_j}{\partial x_i} \quad (17.42)$$

leads us to the desired result:

$$\begin{aligned} \frac{\partial f}{\partial x_6} &= 1, \quad \frac{\partial f}{\partial x_5} = \frac{\partial f}{\partial x_6} \cdot \frac{\partial x_6}{\partial x_5} = -1e^{-x_5} \\ \frac{\partial f}{\partial x_3} &= \frac{\partial f}{\partial x_5} \cdot \frac{\partial x_5}{\partial x_3} = -e^{-x_5} \cdot 1 = -e^{x_5} \\ \frac{\partial f}{\partial x_1} &= \frac{\partial f}{\partial x_3} \cdot \frac{\partial x_3}{\partial x_1} = -e^{-x_5} \cdot 2x_1 = -2x_1 e^{-(x_1^2 + x_2^2)} \\ &\left(\text{Similar story for } \frac{\partial f}{\partial x_2} \right). \end{aligned} \quad (17.43)$$

We conclude with some remarks: AD is not the same as *symbolic differentiation* nor *numerical differentiation* using finite differences. Symbolic differentiation leads to inefficient code, while numerical differentiation can introduce roundoff errors and produce subtractive cancellation effects. Both methods become slow when computing derivatives of a function with respect to many inputs. Even CSM will produce roundoff errors when we wish to apply it to approximating second-order derivatives, for example.

The definitive reference on automatic differentiation is Griewank and Walther (2008). Summarising, forward automatic differentiation relates to the *forward propagation of derivative information*, while backward automatic differentiation concerns the *backwards propagation of derivatives*.

A good exercise is to apply ‘manual AD’ to Equation (17.37).

17.6 DUAL NUMBERS

To quote Wikipedia: ‘Forward mode automatic differentiation is accomplished by augmenting the algebra of real numbers and obtaining a new arithmetic. An additional component is added to every number to represent the derivative of a function at the number, and all arithmetic operators are extended for the augmented algebra. The augmented algebra is the algebra of dual numbers.’

A *dual number* is a kind of *hypercomplex number* (such as quaternions), and dual numbers form an algebraic structure that originated in the nineteenth century. A *dual number* has the form $a + b\varepsilon$ where ε is nilpotent of order 2, that is $\varepsilon^2 = 0$. In short, we replace each number a by number of the form $a + b\varepsilon$. Addition, subtraction and multiplication are defined by:

$$\begin{aligned} a \pm b\varepsilon + c \pm d\varepsilon &= (a \pm c) + (b \pm d)\varepsilon \\ (a + b) \cdot (c + d\varepsilon) &= ac + ad\varepsilon + bc\varepsilon + bd\varepsilon^2 = ac + (ad + bc)\varepsilon. \end{aligned} \quad (17.44)$$

Division of dual numbers is defined by:

$$\begin{aligned}\frac{a+b\varepsilon}{c+d\varepsilon} &= \frac{(a+b\varepsilon)(c-d\varepsilon)}{(c+d\varepsilon)(c-d\varepsilon)} = \frac{ac-ad\varepsilon+bc\varepsilon}{c^2} \\ &= \frac{a}{c} + \left(\frac{bc-ad}{c^2}\right)\varepsilon \text{ if } c \neq 0.\end{aligned}\tag{17.45}$$

Let us take an example of a n th-degree polynomial:

$$P(x) = \sum_{j=0}^n p_j x^j\tag{17.46}$$

Then:

$$P(a + b\varepsilon) = P(a) + bP'(a)\varepsilon \quad \left(\text{where } P' = \frac{dP}{dx}\right).$$

In this way we can compute derivatives of polynomials using dual numbers rather than reals. More generally, we can extend any (analytic) real function to the set of dual numbers by examining its Taylor series:

$$f(a + b\varepsilon) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)b^n\varepsilon^n}{n!} = f(a) + bf'(a)\varepsilon.\tag{17.47}$$

17.7 AUTOMATIC DIFFERENTIATION IN C++

There are many software tools for automatic differentiation in many programming languages. It is not possible to discuss them here. We do, however, give a brief overview of *Autodiff* that is part of Boost C++ libraries. It is a header-only C++ library that facilitates the automatic differentiation (*forward mode*) of mathematical functions of single and multiple variables. The implementation is based on the truncated Taylor series expansion of an analytic function at a given point:

$$f(x_0 + \varepsilon) = \sum_{n=0}^N \frac{f^{(n)}(x_0)}{n!} \varepsilon^n + O(\varepsilon^{N+1}).\tag{17.48}$$

Specifically, we discard powers greater than N :

$$\sum_{n=0}^N y_n \varepsilon^n = \sum_{n=0}^N \frac{f^{(n)}(x_0)}{n!} \varepsilon^n.\tag{17.49}$$

Finally, we can find the n th derivative of f by the formula:

$$\frac{d^n f}{dx^n} = n! y_n.\tag{17.50}$$

We take some examples of C++ code to show how to use the library.

```

// Test case from Squire&Trapp 1998
template <typename T> T SquireTrapp(const T& t)
{
    T n1 = exp(t);
    T d1 = sin(t);
    T d2 = cos(t);

    return n1 / (d1 * d1 * d1 + d2 * d2 * d2);
}

template <typename X1, typename X2>
promote<X1, X2> TestFunc(const X1& x1, const X2& x2)
{
    return exp(-(x1 * x1 + x2 * x2));
}

int main()
{
    constexpr unsigned Order = 3; // Highest order derivative calculated.
    // Find derivatives at x=1.5.
    auto const x = make_fvar<double, Order>(1.5);
    //auto const y = TestFunc(x);
    auto const y = SquireTrapp(x);

    for (unsigned i = 0; i <= Order; ++i)
        std::cout << "y.derivative(" << i << ") = "
        << y.derivative(i) << std::endl;

    double x1 = 1.0; double x2 = 1.0;
    std::cout << "func(x1,x2) " << TestFunc(x1, x2);
    return 0;
}

```

More examples can be found in the *Autodiff* user manual, including code to generate Black–Scholes greeks.

17.8 SUMMARY AND CONCLUSIONS

In this chapter we continued our discussion of sensitivity analysis that we introduced in Chapter 16. Summarising, we have discussed the following methods in Chapters 16 and 17 for computing sensitivities:

- Finite differences.
- Cubic splines.
- Complex Step Method (CSM).
- Continuous sensitivity (CSE).
- Automatic differentiation (AD).

These methods are used in many kinds of applications, for example, engineering design, computational finance and machine learning (ML).

Advanced Finite Difference Schemes for Two-Factor Problems

CHAPTER 18

Splitting Methods, Part I

Indeed, it is obvious that invention or discovery, be it in mathematics or anywhere else, takes place by combining ideas. Now, there is an extremely great number of such combinations, most of which are devoid of interest, while, on the contrary, very few of them can be fruitful. Which ones does our mind – I mean our conscious mind – perceive? Only the fruitful ones, or exceptionally, some which could be fruitful. However, to find these, it has been necessary to construct the very numerous possible combinations, among which the useful ones are to be found.

Jacques Hadamard

18.1 INTRODUCTION AND OBJECTIVES

In this chapter we introduce a number of numerical methods to solve complex, high-dimensional problems by a sequence of simpler (usually, lower-dimensional) problems. We approximate and merge the lower-dimensional problems using numerical methods to solve the original problem in an efficient way. In other words, we decompose a complex problem into simpler subproblems, and we then recombine the subsolutions to produce a solution to the original problem. The approach has universal appeal. Our main focus in this and later chapters is to apply what we call *splitting methods* to a wide range of algebraic, ordinary, stochastic and parabolic time-dependent partial differential equations in two space variables.

It is possible to directly extend the finite difference methods of previous chapters to multidimensional problems (Hageman and Young (1981), Tavella and Randall (2000), Varga (1962)). But there is a cost. First, explicit-in-time methods are inefficient because they place major restrictions on the time step in order to achieve stability, thus rendering them all but useless for serious computation. We can resolve this problem by using implicit-in-time methods, resulting in first-order or second-order accurate and stable difference schemes. However, we must solve block triangular or sparse linear systems at each time level, which forces us to use specialised matrix solvers (Davis (2006)). We do mention that it is possible to apply the *generalised minimal residual* (GMRES) iterative method with incomplete LU (ILU) factorisation preconditioning for multi-dimensional problems, thus avoiding the need to split them in the first place (see Saad and Schultz

(1986) and Saad (1996) for more details on GMRES). We do not discuss these methods here.

The main goal of this chapter is to introduce various kinds of splitting methods for initial boundary value problems for two-dimensional time-dependent diffusion equations with mixed derivatives. In this way we can concentrate on the most important issues in the short term and then add more functionality as we progress to more complicated examples and applications.

In particular, we discuss:

- Locally one-dimensional methods (LOD), also known as fractional step, Soviet splitting and splitting-up methods.
- Alternating Direction Implicit (ADI) method.
- Some popular LOD and ADI schemes.
- Error and stability analysis.
- Handling mixed derivatives.

In this chapter we show how to apply splitting methods to two-factor diffusion, convection (advection) and mixed derivative equations before we address convection-diffusion-reaction equations and advanced splitting methods in Chapter 22.

We should not confuse the Alternating Direction Explicit (ADE) method with ADI. We introduce ADE in Chapter 19.

18.2 BACKGROUND AND HISTORY

The two major categories of numerical methods for time-dependent partial differential equations are called *Alternating Direction Implicit* (ADI) method (Peaceman (1977), Roache (1998), Craig and Sneyd (1988)) and locally one-dimensional (LOD) or (Soviet) *splitting methods* (Yanenko (1971), Marchuk (1982), Marchuk, Rusakov, Zalesny and Diansky (2005)). These methods originated in the 1960s in the USA and the former USSR, and they are used to solve partial differential equations in reservoir engineering, fluid dynamics, heat transfer and nuclear engineering. The methods all have one thing in common: they decompose a multi-dimensional problem into a sequence of simpler one-dimensional problems. The differences are somewhat minor and non-essential to a certain degree, and it would seem that ADI is a special case of splitting.

Notwithstanding, ADI tends to be the term that is used in computational finance literature and applications. Historically, relatively few researchers in the West have had exposure to splitting methods. Most of the original research was written in Russian in journals that were not readily accessible in the West at the time.

To our knowledge, the application of splitting methods to computational finance was first introduced in Duffy (2006). The first production application is discussed in Davidson and Levin (2014), in which the authors use the Marchuk two-cycle modification of the Crank–Nicolson splitting method. (See Chapter 22 where we introduce Marchuk’s method) to value mortgage-backed securities (MBS) under a two-factor Gaussian rate model consisting of a short rate and a slope factor. These factors are uncorrelated.) Second-order monotone schemes are produced. Another notable

application is discussed in Sheppard (2007), in which splitting is applied to the Heston stochastic volatility model. The author used Duffy's exponential fitting for the convection-diffusion term and Yanenko's method for the mixed derivative term.

In general, we prefer splitting to ADI because it is more robust, is easy to program and has been successfully applied to various kinds of applications in computational finance (see, for example, Sheppard (2007) where splitting is used to price options in the Heston stochastic volatility model and for which ADI didn't perform as expected). We have more anecdotal examples, and for this reason we do not discuss ADI (at least, ADI *classic*) in this book.

18.3 NOTATION, PREREQUISITES AND MODEL PROBLEMS

In this chapter we focus primarily on the two-dimensional heat equation to build expertise:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \quad (18.1)$$

that is defined in some region of (x, y, t) space. In this section we extend the finite difference method that we defined in previous chapters to the case where the continuous (x, y) space is replaced by a two-dimensional mesh. To this end, we use some standard notation for difference operators in the x and y directions:

$$\begin{aligned}\Delta_x^2 u_{ij} &= h_1^{-2}(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) \\ \Delta_x^2 u_{ij} &= h_2^{-2}(u_{i,j+1} - 2u_{i,j} + u_{i,j-1}) \\ \Delta_x^+ u_{ij} &= h_1^{-1}(u_{i+1,j} - u_{i,j}) \\ \Delta_x^- u_{ij} &= h_1^{-1}(u_{i,j} - u_{i-1,j}) \\ \Delta_x^0 u_{ij} &= (2h_1)^{-1}(u_{i+1,j} - u_{i-1,j})\end{aligned} \quad (18.2)$$

(with similar definitions for Δ_y^- , Δ_y^+ and Δ_y^0). The parameters h_1 and h_2 are the step sizes in the x and y directions, respectively.

These operators are the two-dimensional extensions of the one-dimensional discrete operators of previous chapters. Thus, when approximating the heat equation (18.1), we can choose centred differencing in the x and y directions, while we can choose the following options for the time direction:

- Explicit Euler (EE)
- Implicit Euler (IE)
- Crank–Nicolson (CN).

The heat equation is probably one of the most famous equations in all of mathematical physics. In two dimensions it is given by:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}. \quad (18.3)$$

This equation is usually defined on a bounded, semi-infinite or infinite two-dimensional region. On the boundaries, we define *boundary conditions* as well as an associated *initial condition*. For example, on a bounded rectangle $[0, L] \times [0, M]$ we can define the Dirichlet boundary conditions on one part of the boundary and Neumann boundary conditions on the other part:

$$\begin{aligned} u(x, 0, t) &= 0, \quad 0 < x < L \\ u(x, M, t) &= 0, \quad 0 < x < L \\ \frac{\partial u}{\partial x}(0, y, t) &= 0, \quad 0 < y < M \\ \frac{\partial u}{\partial x}(L, y, t) &= 0, \quad 0 < y < M. \end{aligned} \tag{18.4}$$

Finally, we prescribe the initial condition:

$$u(x, y, 0) = f(x, y), \quad 0 \leq x \leq L, \quad 0 \leq y \leq M. \tag{18.5}$$

We call the combined set of Equations (18.3), (18.4) and (18.5) the *initial boundary value problem* (IBVP) for the heat equation. We are interested in finding stable and accurate finite difference schemes for this and other similar problems. In general, we employ centred difference schemes in the x and y directions, while for time discretisation we use the *Theta method* (its special cases are the explicit Euler, implicit Euler and Crank–Nicolson schemes).

We first discretise (18.3) by the *explicit Euler scheme*:

$$\frac{U_{i,j}^{n+1} - U_{i,j}^n}{k} = \Delta_x^2 U_{i,j}^n + \Delta_y^2 U_{i,j}^n. \tag{18.6}$$

This is a *time-marching scheme* from level n (where the value is known) to level $n + 1$ (where the value is unknown). Rearranging terms gives us an explicit formula as follows:

$$\begin{aligned} U_{i,j}^{n+1} &= \lambda U_{i-1,j}^n + r U_{i,j}^n + \lambda U_{i+1,j}^n + \lambda U_{i,j-1}^n + \lambda U_{i,j+1}^n \\ &= r U_{i,j}^n + \lambda \left\{ U_{i-1,j}^n + U_{i+1,j}^n + U_{i,j-1}^n + U_{i,j+1}^n \right\} \end{aligned} \tag{18.7}$$

where $\lambda = k/h^2$ and $r = 1 - 4\lambda$. (Compared to (18.2) we take $h_1 = h_2 = h$).

We now examine this discrete scheme from the following perspective: Given that the discrete solution is positive at time level n , can we find sufficient conditions that ensure that the solution is also positive at level $n + 1$? Examining Equation (18.7) allows us to conclude that this constraint is:

$$r \geq 0 \iff \frac{k}{h^2} \leq \frac{1}{4}.$$

Of course, we do not want to get negative solutions from positive input. Negative values are non-physical (or ‘non-financial’).

We can also apply the von Neumann stability (the index i is unfortunately used in two contexts) analysis technique to the scheme (18.6) to get the same constraint above. Let (see Peaceman (1977)):

$$\epsilon_{ij}^n = \gamma^n \exp(i\alpha ih) \exp(i\beta jh), \quad i = \sqrt{-1}$$

then constructing the terms:

$$\Delta_x^2 \epsilon_{ij}^n + \Delta_y^2 \epsilon_{ij}^n$$

and noting that:

$$\cos \alpha h - 1 = 2 \sin^2 \frac{\alpha h}{2}.$$

Doing a bit of arithmetic we see that the amplification factor is:

$$\gamma = 1 - 4\lambda \sin^2 \frac{\alpha h}{2} - 4\lambda \sin^2 \frac{\beta h}{2}.$$

For stability, we must have:

$$-1 \leq \gamma \leq 1$$

and this leads to the same constraint as before. This is a requirement for stability. Of course, the positivity argument is more intuitive than the von Neumann analysis.

The (*fully*) *implicit method* is given by:

$$\frac{U_{i,j}^{n+1} - U_{i,j}^n}{k} = \Delta_x^2 U_{i,j}^{n+1} + \Delta_y^2 U_{i,j}^{n+1} \quad (18.8)$$

and rearranging again gives us:

$$U_{i,j}^{n+1}(1 + 4\lambda) = U_{ij}^n + \lambda \left\{ U_{i-1,j}^{n+1} + U_{i+1,j}^{n+1} + U_{i,j-1}^{n+1} + U_{i,j+1}^{n+1} \right\}. \quad (18.9)$$

Again, we see that positive values at level n give us positive values at level $n + 1$ irrespective of the relative sizes of k and h . We say that this scheme is unconditionally stable. It is sometimes called a *monotonic scheme*. Some arithmetic shows that the amplification factor is:

$$\gamma = \frac{1}{1 + 4\lambda \sin^2 \frac{\alpha h}{2} + 4\lambda \sin^2 \frac{\beta h}{2}}. \quad (18.10)$$

This is always less than 1 in absolute value.

Finally, the *Crank–Nicolson* scheme is given by:

$$\frac{U_{i,j}^{n+1} - U_{i,j}^n}{k} = \Delta_x^2 U_{i,j}^{n+\frac{1}{2}} + \Delta_y^2 U_{i,j}^{n+\frac{1}{2}} \quad (18.11)$$

where:

$$U_{i,j}^{n+\frac{1}{2}} \equiv \frac{1}{2} (U_{i,j}^{n+1} + U_{i,j}^n).$$

This scheme is not positive in the above sense, but it is unconditionally stable. The von Neumann symbol is given by:

$$\gamma = \frac{1 - i\beta}{1 + i\beta} \quad (18.12)$$

where:

$$\beta = 2\lambda \sin^2 \frac{\alpha h}{2} + 2\lambda \sin^2 \frac{\beta h}{2}.$$

What is the absolute value of γ in this case?

18.4 MOTIVATION: TWO-DIMENSIONAL HEAT EQUATION

18.4.1 Alternating Direction Implicit (ADI) Method

In general, ADI is a method that approximates the solution of an initial boundary value problem by a sequence of simpler problems. In order to motivate what ADI is, we consider again the heat equation:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}. \quad (18.13)$$

In Section 18.3 (Equation (18.9)) we approximated this equation by a scheme (recall the notation for divided differences):

$$\frac{U_{ij}^{n+1} - U_{ij}^n}{k} = \Delta_x^2 U_{ij}^{n+1} + \Delta_y^2 U_{ij}^{n+1}. \quad (18.14)$$

The disadvantage of this scheme is that we must solve a large system of equations at each time level. In this chapter, however, we propose schemes that allow us to simplify the scheme (18.14) while still keeping the scheme stable and accurate. We now modify Equation (18.14) somewhat so that it becomes implicit in the x direction and explicit in the y direction:

$$\frac{U_{ij}^{n+1} - U_{ij}^n}{k} = \Delta_x^2 U_{ij}^{n+1} + \Delta_y^2 U_{ij}^n. \quad (18.15)$$

In this case we can solve problem (18.15) since it can be cast as a tridiagonal system that can subsequently be solved using LU decomposition, as discussed in Chapter 6 (Sections (6.3) and (6.4)). However, we must determine if it is stable (whether unconditionally (absolutely) or conditionally). To prove stability, we can employ the following techniques:

- Von Neumann stability analysis.
- Positivity and maximum principle analysis.

We examine the positivity argument first. We rewrite the system (18.15) as follows:

$$-\lambda U_{i-1,j}^{n+1} + (1 + 2\lambda)U_{ij}^{n+1} - \lambda U_{i+1,j}^{n+1} = \lambda U_{i,j-1}^n + (1 - 2\lambda)U_{ij}^n + \lambda U_{i,j+1}^n. \quad (18.16)$$

We wish to find sufficient conditions to ensure that the right-hand side of (18.16) is positive at time level $n + 1$, assuming that the discrete solution at time level n is positive. We then get the condition:

$$1 - 2\lambda \geq 0 \Leftrightarrow \lambda = \frac{k}{h^2} \leq 1/2. \quad (18.17)$$

We get the same condition if we apply von Neumann stability analysis. Continuing, we write (18.16) in the matrix form:

$$MU^{n+1} = BU^n \text{ or } U^{n+1} = M^{-1}BU^n \quad (18.18)$$

where M and B are matrices.

The solution at time level $n + 1$ is positive under constraint (18.17) because both the inverse of M and the matrix B are positive matrices, and since the product of positive matrices is positive, we get the result. The matrix B is positive because the constraint (18.17) must be satisfied, while the inverse of M is positive because M is an M -matrix, that is:

$$M = (m_{ij}), \quad i, j = 1, \dots, n, \quad m_{ii} > 0, \quad m_{ij} \leq 0, \quad i \neq j.$$

So we see unfortunately that the scheme (18.15) is only conditionally stable. This is unacceptable for real applications. Can we improve on this situation? To answer this question, let us consider consecutive applications of this scheme at two time legs; the first leg is implicit in x and explicit in y while the second leg is explicit in x and implicit in y . The new scheme moves from the time level n to somewhat ‘fictitious’ time level $n + 1/2$ and then to time level $n + 1$. The full scheme is:

$$(a) \frac{U_{ij}^{n+1/2} - U_{ij}^n}{k/2} = \Delta_x^2 U_{ij}^{n+1/2} + \Delta_y^2 U_{ij}^n$$

$$(b) \frac{U_{ij}^{n+1} - U_{ij}^{n+1/2}}{k/2} = \Delta_x^2 U_{ij}^{n+1/2} + \Delta_y^2 U_{ij}^{n+1}. \quad (18.19)$$

The hope is that even though the scheme at each leg is only conditionally stable (at best), there might be a chance that the full scheme that marches the solution from time level n to time level $n + 1$ will be stable. The scheme alternates between what are essentially one-dimensional implicit schemes, whence the name *Alternating Direction Implicit* (ADI). In general, the increase in the error due to the presence of the explicit term in a given leg is balanced by the error decrease in the implicit scheme in the next leg. To verify this statement, we use von Neumann stability analysis to prove unconditional stability of scheme (18.19). We assume a uniform length h in the x and y directions for convenience. Let:

$$\epsilon_{ij}^n = \gamma^n \exp(i\alpha ih) \exp(i\beta jh).$$

Then after using the results:

$$\begin{aligned}\Delta_x^2 \epsilon_{ij}^n &= -\frac{4}{h^2} \epsilon_{ij}^n \sin^2 \alpha h^2 / 2 \\ \Delta_y^2 \epsilon_{ij}^n &= -\frac{4}{h^2} \epsilon_{ij}^n \sin^2 \beta h^2 / 2\end{aligned}$$

we get the following expressions for the growth factors:

$$\begin{aligned}\frac{\gamma^{n+1/2}}{\gamma^n} &= \frac{1 - \alpha_1}{1 + \alpha_2} \\ \frac{\gamma^{n+1}}{\gamma^{n+1/2}} &= \frac{1 - \alpha_2}{1 + \alpha_1}\end{aligned}$$

where:

$$\begin{aligned}\alpha_1 &= 4\lambda \sin^2 \beta h_2 / 2 \\ \alpha_2 &= 4\lambda \sin^2 \alpha h / 2\end{aligned}$$

and:

$$\lambda = k/h^2.$$

Hence:

$$\frac{\gamma^{n+1}}{\gamma^n} = \frac{1 - \alpha_2}{1 + \alpha_1} \cdot \frac{1 - \alpha_1}{1 + \alpha_2}.$$

We thus see that the growth factor γ from n to $n + 1$ is less than 1 in absolute value. Hence the scheme (18.19) is unconditionally stable. It is called the *Peaceman–Rachford* scheme. It is second-order accurate in time and space (see, for example, Thomas (1995)).

Please note that we have not yet discussed boundary conditions, but we shall need to incorporate them into these ADI schemes.

18.4.2 Soviet (Operator) Splitting

We examine the two-dimensional heat equation:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}. \quad (18.20)$$

The idea behind operator splitting is to reduce Equation (18.20) to two one-dimensional problems. We then approximate each subproblem by implicit or explicit schemes. Thus, we are thinking intuitively of two one-dimensional partial differential equations:

$$\frac{\partial v}{\partial t} = \frac{\partial^2 v}{\partial x^2} \text{ and } \frac{\partial w}{\partial t} = \frac{\partial^2 w}{\partial y^2} \quad (18.21)$$

where the functions v and w are unspecified. In general we take centred differencing in space and explicit or implicit time marching in time. For example, using explicit Euler we get the two-leg scheme:

$$\begin{aligned}\frac{\tilde{U}_{ij} - U_{ij}^n}{\Delta t} &= \Delta_x^2 U_{ij}^n \\ \frac{U_{ij}^{n+1} - \tilde{U}_{ij}}{\Delta t} &= \Delta_y^2 \tilde{U}_{ij}\end{aligned}\quad (18.22)$$

where we have used the notation \tilde{U} for the *intermediate value*. (It is sometimes denoted as $U^{n+1/2}$.)

Let us assume for convenience that the mesh size in the x and y directions is the same, namely h . We wish to examine the stability of scheme (18.22). We expect it to be conditionally stable only, and we can prove this using either von Neumann stability analysis or the maximum principle. Using the former method, we see that the amplification factor is given by:

$$\frac{\gamma^{n+1}}{\gamma^n} = (1 - 4\lambda \sin^2 \alpha h/2)(1 - 4\lambda \sin^2 \beta h/2), \quad \lambda = \frac{k}{h^2}. \quad (18.23)$$

This leads to the constraint:

$$\frac{k}{h^2} \leq \frac{1}{2}. \quad (18.24)$$

Now, the *implicit splitting scheme* is defined by:

$$\begin{aligned}\frac{\tilde{U}_{ij} - U_{ij}^n}{\Delta t} &= \Delta_x^2 \tilde{U}_{ij} \\ \frac{U_{ij}^{n+1} - \tilde{U}_{ij}}{\Delta t} &= \Delta_y^2 U_{ij}^{n+1}.\end{aligned}\quad (18.25)$$

This scheme is unconditionally stable. In fact, each leg is stable, a property not shared by the ADI schemes. Finally, it is possible to define a splitting method in conjunction with Crank–Nicolson time marching:

$$\begin{aligned}\frac{\tilde{U}_{ij} - U_{ij}^n}{k} &= \frac{1}{2} (\Delta_x^2 \tilde{U}_{ij} + \Delta_x^2 U_{ij}^n) \\ \frac{U_{ij}^{n+1} - \tilde{U}_{ij}}{k} &= \frac{1}{2} (\Delta_y^2 U_{ij}^{n+1} + \Delta_y^2 \tilde{U}_{ij}).\end{aligned}\quad (18.26)$$

Having motivated splitting, we now discuss a number of important issues that will be useful when we model multi-factor Black–Scholes problems.

18.4.3 Mixed Derivative and Yanenko Scheme

The ADI method is not good at approximating *mixed derivatives*, and a number of workarounds have been suggested by researchers and practitioners. The splitting method is better, and to this end we examine the constant-coefficient problem:

$$\begin{aligned}\frac{\partial u}{\partial t} &= Lu \\ Lu &\cong \sum_{i,j=1}^2 a_{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} \\ a_{11} a_{22} - a_{12}^2 > 0, \quad a_{11} > 0, \quad a_{22} > 0.\end{aligned}\tag{18.27}$$

In Yanenko (1971) the following scheme is proposed (using notation in (18.2)):

$$\begin{aligned}\frac{\tilde{U}_{ij} - U_{ij}^n}{\Delta t} &= a_{11} \Delta_x^2 \tilde{U}_{ij} + a_{12} \Delta_x \Delta_y U_{ij}^n \\ \frac{U_{ij}^{n+1} - \tilde{U}_{ij}}{\Delta t} &= a_{21} \Delta_x \Delta_y \tilde{U}_{ij} + a_{22} \Delta_y^2 U_{ij}^{n+1}.\end{aligned}\tag{18.28}$$

This scheme is stable and convergent (see Yanenko (1971)), and it resolves the problems that ADI methods show for this equation.

We shall see later how to use important scheme (18.28) in multi-factor Black–Scholes problems. It is a very robust method.

Yanenko has also produced a scheme for the three-dimensional anisotropic heat conduction equation:

$$\frac{\partial u}{\partial t} = \sum_{i,j=1}^3 a_{ij} \frac{\partial^2 u}{\partial x_i \partial x_j}.\tag{18.29}$$

The proposed scheme is:

$$\begin{aligned}\frac{U^{n+1/6} - U^n}{k} &= \frac{1}{2} \Lambda_{11} U^{n+1/6} + \Lambda_{12} U^n \\ \frac{U^{n+2/6} - U^{n+1/6}}{k} &= \Lambda_{21} U^{n+1/6} + \frac{1}{2} \Lambda_{22} U^{n+2/6} \\ \frac{U^{n+3/6} - U^{n+2/6}}{k} &= \frac{1}{2} \Lambda_{11} U^{n+3/6} + \Lambda_{13} U^{n+2/6} \\ \frac{U^{n+4/6} - U^{n+3/6}}{k} &= \Lambda_{31} U^{n+3/6} + \frac{1}{2} \Lambda_{33} U^{n+4/6} \\ \frac{U^{n+5/6} - U^{n+4/6}}{k} &= \frac{1}{2} \Lambda_{22} U^{n+5/6} + \Lambda_{23} U^{n+4/6} \\ \frac{U^{n+1} - U^{n+5/6}}{k} &= \Lambda_{32} U^{n+5/6} + \frac{1}{2} \Lambda_{33} U^{n+1}\end{aligned}\tag{18.30}$$

where:

$$\Lambda_{jj}u \sim a_{jj} \frac{\partial^2 u}{\partial x_j^2}, \quad j = 1, 2, 3$$

$$\Lambda_{i,j}u \sim a_{ij} \frac{\partial^2 u}{\partial x_i \partial x_j}, \quad i \neq j, \quad i, j = 1, 2, 3.$$

This scheme is consistent with PDE (18.29) and is stable, provided the matrix $B = (b_{ij})$ is positive definite where $b_{ij} = a_{ij}$, $i \neq j$ and $b_{ii} = \frac{a_{ii}}{2}$.

This scheme can be generalised to other differential operators that appear in the financial engineering literature, as we shall see in Chapters 22 and 23.

We conclude our discussion of mixed derivatives by proving a result concerning the approximation of the mixed derivative by divided differences:

$$\frac{\partial^2 u}{\partial x \partial y}(x_i, y_i) \sim \frac{1}{4h_x h_y} (u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}). \quad (18.31)$$

The steps in the proof are given as follows:

$$\begin{aligned} \Delta_x \Delta_y u_{ij} &= \frac{1}{2h_y} \Delta_x (u_{i,j+1} - u_{i,j-1}) \\ &= \frac{1}{4h_x h_y} ([u_{i+1,j+1} - u_{i-1,j+1}] - [u_{i+1,j-1} - u_{i-1,j-1}]) \\ &= \frac{1}{4h_x h_y} (u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}) \end{aligned} \quad (18.32)$$

as was to be shown. There are other ways to approximate the mixed derivatives, but we do not discuss them here.

18.5 OTHER RELATED SCHEMES FOR THE HEAT EQUATION

Section 18.5 discusses a number of specialised topics and may be skipped on a first reading.

18.5.1 D'Yakonov Method

We introduce some variations on the basic ADI scheme.

In this section we discuss some modifications of the original ADI scheme (18.19) in order to improve computational efficiency. First, we eliminate the solution at time level $n + 1/2$ by using Equation (18.19)(a) to get the scheme:

$$\left(1 - \frac{k}{2}\Delta_x^2\right) \left(1 - \frac{k}{2}\Delta_y^2\right) U_{ij}^{n+1} = \left(1 + \frac{k}{2}\Delta_x^2\right) \left(1 + \frac{k}{2}\Delta_y^2\right) U_{ij}^n. \quad (18.33)$$

This equation suggests another splitting by the so-called *D'Yakonov scheme* that we define as follows:

$$\begin{aligned} \left(1 - \frac{k}{2}\Delta_x^2\right)U_{ij}^* &= \left(1 + \frac{k}{2}\Delta_x^2\right)\left(1 + \frac{k}{2}\Delta_y^2\right)U_{ij}^n \\ \left(1 - \frac{k}{2}\Delta_y^2\right)U_{ij}^{n+1} &= U_{ij}^*. \end{aligned} \quad (18.34)$$

This set of equations is easy to solve: we apply LU decomposition at each leg. We note that the matrix in the matrix system is tridiagonal.

18.5.2 Approximate Factorisation of Operators

We now discuss a technique that allows us to factor a given difference operator in two dimensions into the product of two one-dimensional operators. Let us take the example of the Crank–Nicolson scheme for the two-dimensional heat equation (18.1) again:

$$\frac{U_{ij}^{n+1} - U_{ij}^n}{k} = \frac{1}{2} \left(\Delta_x^2 U_{ij}^{n+1} + \Delta_y^2 U_{ij}^{n+1} + \Delta_x^2 U_{ij}^n + \Delta_y^2 U_{ij}^n \right). \quad (18.35)$$

We write this equation in the equivalent form:

$$(1 - L_x - L_y)U_{ij}^{n+1} = (1 + L_x + L_y)U_{ij}^n \quad (18.36)$$

where:

$$L_x \equiv \frac{k}{2}\Delta_x^2, \quad L_y \equiv \frac{k}{2}\Delta_y^2.$$

We now factor the terms on both sides of Equation (18.36) by using the formula:

$$\begin{aligned} (1 - L_x)(1 - L_y) &= 1 - L_x - L_y + L_x L_y \\ (1 + L_x)(1 + L_y) &= 1 + L_x + L_y + L_x L_y. \end{aligned}$$

We then get the so-called *approximate factorisation scheme* by neglecting the mixed terms:

$$(1 - L_x)(1 - L_y)U_{ij}^{n+1} = (1 + L_x)(1 + L_y)U_{ij}^n. \quad (18.37)$$

This scheme is second-order in k . The idea can be generalised to more complex PDEs.

As a more general example, let us now examine the heat equation in m dimensions:

$$\frac{\partial u}{\partial t} = \sum_{j=1}^m \frac{\partial^2 u}{\partial x_j^2} \quad (18.38)$$

and its approximation by the difference scheme:

$$\begin{aligned} \frac{U^{n+1} - U^n}{k} &= LU^{n+1} \\ L &= \sum_{j=1}^m L_j \text{ (discrete operator)} \\ L_j &= \frac{\Delta_+ \Delta_-}{h_j^2} \end{aligned} \quad (18.39)$$

where Δ_+ and Δ_- are the forward and backward approximations to the first derivative of a function in the direction j .

Please note that we have suppressed the subscripts that show dependence on the spatial mesh points (this is for convenience). We then write equation (18.39) in the form:

$$(I - kL)U^{n+1} = IU^n \equiv U^n. \quad (18.40)$$

We now factor the operator $I - kL$ by producing a second-order accurate approximation (Yanenko (1971)):

$$(I - kL_1)(I - kL_2) \dots (I - kL_m) = 1 - kL + k^2\Phi. \quad (18.41)$$

where:

$$\Phi = \sum_{i < l} L_i L_j - k \sum_{i < j < k} L_i L_j L_k + \dots + (-1)^m k^{m-2} L_1 \dots L_m.$$

Based on this expression, we now propose a modified form of scheme (18.41):

$$\prod_{j=1}^m (1 - kL_j)U^{n+1} = U^n. \quad (18.42)$$

The splitting scheme based on the so-called *upper operator* in (18.42) is now defined as:

$$\begin{aligned} (1 - kL_1)U^{n+1/m} &= U^n \\ (1 - kL_2)U^{n+2/m} &= U^{n+1/m} \\ &\dots \\ (1 - kL_m)U^{n+1} &= U^{n+(m-1)/m}. \end{aligned} \quad (18.43)$$

As an application of this scheme, we now examine the convection-diffusion equation:

$$\frac{\partial u}{\partial t} + A \frac{\partial u}{\partial x} + B \frac{\partial u}{\partial y} = v \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right). \quad (18.44)$$

We assume that this problem is to be solved in a rectangular region $D = \{(x, y) : 0 < x < 1, 0 < y < 1\}$. However, we do not worry about boundary conditions just yet. Furthermore, we assume that all the coefficients appearing in (18.44) are constant. We define the divided differences:

$$\begin{aligned}\Delta_x U_{ij} &= \frac{1}{2h}(U_{i+1,j} - U_{i-1,j}) \\ \Delta_y U_{ij} &= \frac{1}{2h}(U_{i,j+1} - U_{i,j-1}).\end{aligned}$$

Let us consider the *two-level difference scheme* for (18.44) depending on a single parameter β :

$$\begin{aligned}\frac{U_{ij}^{n+1} - U_{ij}^n}{k} + A\beta\Delta_x U_{ij}^{n+1} + A(1-\beta)\Delta_x U_{ij}^n + B\beta\Delta_y U_{ij}^{n+1} + B(1-\beta)\Delta_y U_{ij}^n \\ = \beta(v\Delta_x^2 U_{ij}^{n+1} + v\Delta_y^2 U_{ij}^{n+1}) + (1-\beta)(v\Delta_x^2 U_{ij}^n + v\Delta_y^2 U_{ij}^n) \quad (0 \leq \beta \leq 1).\end{aligned}\quad (18.45)$$

We write this long-winded expression in the more compact form:

$$(1 + k\beta L_x + k\beta L_y)U_{ij}^{n+1} = (1 - k(1 - \beta)L_x - k(1 - \beta)L_y)U_{ij}^n \quad (18.46)$$

where $L_x \equiv A\Delta_x - v\Delta_x^2$ and $L_y \equiv B\Delta_y - v\Delta_y^2$.

As before, we factor out as follows:

$$(1 + k\beta L_x)(1 + k\beta L_y) = (1 + k\beta L_x + k\beta L_y) + k^2\beta^2 L_x L_y$$

that leads us to the approximate factorisation scheme:

$$(1 + k\beta L_x)(1 + k\beta L_y)U_{ij}^{n+1} = (1 - k(1 - \beta)L_x - k(1 - \beta)L_y)U_{ij}^n \equiv L_3 U_{ij}^n. \quad (18.47)$$

As before, we can implement this scheme as a two-stage algorithm:

$$\begin{aligned}(1 + k\beta L_x)U_{ij}^* &= L_3 U_{ij}^n \\ (1 + k\beta L_y)U_{ij}^{n+1} &= U_{ij}^*.\end{aligned}\quad (18.48)$$

Some remarks:

- The scheme can be applied to more general convection-diffusion problems than those proposed in Equation (18.44), for example coefficients that depend on both space and time and equations having inhomogeneous terms.
- The scheme can be generalised to higher dimensions, as we saw with the m -dimensional heat equation in this section.
- The technique can be applied to systems of equations.
- Convection-dominated problems will impact the stability of the schemes. In this case we could use the exponentially-fitted schemes or upwinding in each leg of the approximate factorisation scheme, for example.

18.5.3 Predictor-Corrector Methods

These are methods that are based on *predictor-corrector* methods for initial value problems for ordinary differential equations (Conte and de Boer (1980)). Again, let us examine the three-dimensional heat equation:

$$\frac{\partial u}{\partial t} = \sum_{j=1}^3 \frac{\partial^2 u}{\partial x_j^2}. \quad (18.49)$$

The following scheme is then unconditionally stable and second-order accurate (for a proof, see Yanenko (1971), p. 29):

$$\begin{aligned} \frac{U^{n+1/6} - U^n}{k/2} &= \Delta_x^2 U^{n+1/6} + \Delta_y^2 U^n + \Delta_z^2 U^n \\ \frac{U^{n+2/6} - U^{n+1/6}}{k/2} &= \Delta_y^2 (U^{n+2/6} - U^n) \\ \frac{U^{n+3/6} - U^{n+2/6}}{k/2} &= \Delta_z^2 (U^{n+3/6} - U^n) \\ \frac{U^{n+1} - U^n}{k} &= \Delta_x^2 U^{n+1/6} + \Delta_y^2 U^{n+2/6} + \Delta_z^2 U^{n+3/6}. \end{aligned} \quad (18.50)$$

In this case we have defined *three predictors* and the '*final*' *corrector* that represents the desired approximate solution at time level $n + 1$. This is called a *stabilising corrections scheme*. The scheme is unconditionally stable and of second order accuracy in both time and space. We use a similar scheme in Chapters 22 and 23.

One final example of a predictor-corrector method is given by:

$$\begin{aligned} \frac{U^{n+1/6} - U^n}{k/2} &= \Delta_x^2 U^{n+1/6} \\ \frac{U^{n+2/6} - U^{n+1/6}}{k/2} &= \Delta_y^2 U^{n+2/6} \\ \frac{U^{n+3/6} - U^{n+2/6}}{k/2} &= \Delta_z^2 U^{n+3/6} \\ \frac{U^{n+1} - U^n}{k} &= (\Delta_x^2 + \Delta_y^2 + \Delta_z^2) U^{n+3/6}. \end{aligned} \quad (18.51)$$

Again, we have three predictors and one corrector. Again, this scheme is unconditionally stable and second-order accurate. This scheme can be generalised to more general partial differential equations, for example convection-diffusion equations and equations with mixed derivatives. Furthermore, the scheme is easy to implement and has good stability and convergence properties.

18.5.4 Partial Integro Differential Equations (PIDEs)

The splitting technique has been applied to the solution of PIDEs by Yanenko, Marchuk and others. For example, consider the PIDE for the kinetic theory equation:

$$\frac{\partial u}{\partial t} + \sum_{k=1}^{m-1} u_k \frac{\partial u}{\partial x_k} + \sigma u = \frac{\sigma_s}{4\pi} \int u(x, y, t) dy + f(x, y, t). \quad (18.52)$$

Now let:

$$\Lambda_1 = \text{approximation to } \sigma I + \frac{\sigma_s}{4\pi} \int u dy,$$

$$\Lambda_2 = \text{approximation to } \sum_{k=1}^{m-1} u_k \frac{\partial u}{\partial x_k},$$

$$\bar{f} = \text{approximation to } f,$$

where the integral term is taken on some interval (it may be bounded, infinite or semi-infinite).

Then the splitting scheme is defined by:

$$\begin{aligned} \frac{U^{n+1/2} - U^n}{k} &= \Lambda_1(\alpha U^{n+1/2} + \beta U^n) + \bar{f} \\ \frac{U^{n+1} - U^{n+1/2}}{k} &= \Lambda_2(\alpha U^{n+1} + \beta U^{n+1/2}) \end{aligned} \quad (18.53)$$

here:

$$\alpha \geq 0, \beta \geq 0, \alpha + \beta = 1$$

$$\Lambda_2 = \Lambda_{21} + \dots + \Lambda_{2,m-1}$$

$$\Lambda_{2j} = \text{approximation to the differential operator } u_j \frac{\partial}{\partial x_j}, \quad j = 1, \dots, m-1.$$

A so-called complete splitting is defined in Yanenko (1971), in which the first-order terms in equation (18.52) are split.

Then the complete splitting scheme is given by:

$$\begin{aligned} \frac{U^{n+1/m} - U^n}{k} &= \Lambda_1(\alpha U^{n+1/m} + \beta U^n) + \bar{f} \\ \frac{U^{n+(j+1)/m} - U^{n+j/m}}{k} &= \Lambda_{2j}(\alpha U^{n+(j+1)/m} + \beta U^{n+j/m}) \\ j &= 1, \dots, m-1. \end{aligned} \quad (18.54)$$

We can choose between different marching schemes in each leg of this scheme, for example explicit in the first leg and fully implicit in the second leg when $m = 2$:

$$\begin{aligned} \frac{U^{n+1/2} - U^n}{k} &= \Lambda_1 U^n & \begin{cases} \alpha = 0 \\ \beta = 1 \end{cases} \\ \frac{U^{n+1} - U^n}{k} &= \Lambda_{21} U^{n+1} & \begin{cases} \alpha = 1 \\ \beta = 0 \end{cases}. \end{aligned} \quad (18.55)$$

We can thus solve the problem as a sequence of one-dimensional problems.

18.6 BOUNDARY CONDITIONS

Of course, when solving initial boundary value problems for the heat equation, we must model the bounded or unbounded region in which the equation is defined. In particular, we describe the conditions on the solution at the boundary of the region. There are five main issues that we address:

- The shape or geometry of the region.
- The kinds of boundary conditions (Dirichlet, Neumann, Robins, linearity).
- How to approximate the boundary conditions.
- How to incorporate the boundary conditions into the ADI or splitting equations.
- Ensuring that boundary approximation does not adversely affect the stability and accuracy of the difference approximation.

We now give a brief discussion of each of these topics, and we focus on creating the algorithm for the two-dimensional heat equation in a rectangular region with Dirichlet boundary conditions. In general, it would seem that ADI and splitting methods are better suited to rectangular regions than to non-rectangular regions, because it is more difficult to approximate function values and their derivatives on curved boundaries than on horizontal or vertical boundaries.

In this chapter we concentrate on Dirichlet boundary conditions for the two-dimensional heat equation, how to approximate boundary conditions and to incorporate them into ADI and splitting schemes. A good discussion of these and other issues can be found in Thomas (1995). In particular, Thomas discusses how to define first-order and second-order approximations to the exact solution on the boundary of the region of interest.

We now discuss the case of Dirichlet boundary conditions. To this end, we consider the model problem on a unit square:

$$(1) \quad \frac{\partial u}{\partial t} = a \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right), (x, y) \in R, \quad t > 0, \quad R = (0, 1) \times (0, 1)$$

$$(2) \quad u(x, y, t) = g(x, y, t), (x, y) \in \partial R, \quad t > 0$$

$$(3) \quad u(x, y, 0) = f(x, y), (x, y) \in \bar{R} \cup \partial R.$$

We rewrite the ADI equations (18.19a) and (18.19b) for the two-dimensional heat equation by grouping known terms on the right-hand side of the equations and unknown terms on the left-hand side (\tilde{U} is the intermediate value $U^{n+1/2}$):

$$(a) \left(1 - \frac{k}{2} \Delta_x^2\right) \tilde{U}_{ij} = \left(1 - \frac{k}{2} \Delta_y^2\right) U_{ij}^n \\ (b) \left(1 - \frac{k}{2} \Delta_y^2\right) U_{ij}^{n+1} = \left(1 + \frac{k}{2} \Delta_x^2\right) \tilde{U}_{ij}. \quad (18.56)$$

In general, there is not much difficulty involved if we wish to calculate the boundary values of the approximate solution at times n and $n + 1$. The real challenge is to determine suitable boundary conditions for the intermediate value $n + 1/2$ in Equation (18.56). To this end, we add the left-hand side of Equation (18.56a) to the right-hand side of Equation (18.56b) and vice versa. This gives us a formula for the intermediate solution by adding the terms of the solution at time levels n and $n + 1$:

$$\tilde{U}_{ij} = \frac{1}{2} \left(1 - \frac{k}{2} \Delta_y^2\right) U_{ij}^{n+1} + \frac{1}{2} \left(1 + \frac{k}{2} \Delta_y^2\right) U_{ij}^n. \quad (18.57)$$

This formula allows us to find the appropriate boundary values. For example, in the x direction these will be:

$$i = 0$$

$$\tilde{U}_{0j} = \frac{1}{2} \left(1 - \frac{k}{2} \Delta_y^2\right) g(0, jh_2, (n + 1)k) + \frac{1}{2} \left(1 + \frac{k}{2} \Delta_y^2\right) g(0, jh_2, nk) \quad (18.58)$$

$$i = M_x$$

$$\tilde{U}_{M_{x,ij}} = \frac{1}{2} \left(1 - \frac{k}{2} \Delta_y^2\right) g(1, jh_2, (n + 1)k) + \frac{1}{2} \left(1 + \frac{k}{2} \Delta_y^2\right) g(1, jh_2, nk).$$

Similarly, we can find the corresponding boundary conditions in the y direction by plugging in special index values of j in Equation (18.57). Equation (18.58) is a second-order (in time) accurate approximation to the boundary condition. An alternative solution is to use the (again) second-order approximation:

$$\tilde{U}_{0j} = g\left(0, jh_2, \left(n + \frac{1}{2}\right)k\right) \\ \tilde{U}_{M_{x,ij}} = g\left(1, jh_2, \left(n + \frac{1}{2}\right)k\right). \quad (18.59)$$

Thus, you may choose between (18.58) and (18.59), as each gives second-order accuracy. See Thomas (1995) and Yanenko (1971) for a justification.

18.7 TWO-DIMENSIONAL CONVECTION PDEs

First-order hyperbolic equations have been extensively studied in the literature. We motivate the theory by providing some appropriate examples. Let us examine the scalar first-order hyperbolic equation (initial value problem):

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} + b \frac{\partial u}{\partial y} = 0 \quad (18.60)$$

$-\infty < x < \infty, \quad -\infty < y < \infty$

with the associated initial condition:

$$u(x, y, 0) = f(x, y). \quad (18.61)$$

We assume in this case that:

$$a > 0, \quad b > 0.$$

The solution of the initial value problem (18.60), (18.61) is then given by:

$$u(x, y, t) = f(x - at, y - bt). \quad (18.62)$$

Thus, as in the one-dimensional case, the solution consists of translating the initial condition in the appropriate direction. The constant coefficients a and b are called the *speed of propagation* in the x and y directions, respectively. The curve through the point (x, y, t) defined by the equations:

$$\begin{aligned} x - at &= x_0 \\ y - bt &= y_0 \end{aligned} \quad (18.63)$$

is called a *characteristic curve*. Here x_0 and y_0 are arbitrary points.

First-order hyperbolic equations are a bit more tricky than the heat equation and other second-order parabolic equations. Some of the reasons are:

- Since the equation is first-order in x and y , we need just a single boundary condition at one of the boundaries in the *domain of dependence*. But the question is: Where do we place the boundary condition?
- Centred-difference schemes do not necessarily produce stable results.
- Unlike parabolic equations (where discontinuities in the initial condition are smoothed after a short time), discontinuities propagate through the domain of dependence when we model hyperbolic equations. Furthermore, for some kinds of nonlinear hyperbolic equations, the solution may become discontinuous after a finite time, even if the initial conditions are continuous.
- The imposition of boundary conditions can be tricky, especially for systems (Friedrichs (1958)).

Let us start with an example and suppose that we discretise Equation (18.60) using explicit Euler in time and centred differencing in the x and y directions:

$$\frac{U_{i,j}^{n+1} - U_{i,j}^n}{k} + \frac{a}{2h_1} (U_{i+1,j}^n - U_{i-1,j}^n) + \frac{b}{2h_2} (U_{i,j+1}^n - U_{i,j-1}^n) = 0 \quad (18.64)$$

where h_1 and h_2 are the step lengths in the x and y directions, respectively. The symbol for this operator and its absolute value are given by (Thomas (1995)):

$$\begin{aligned} \gamma &= 1 - i(R_x \sin \xi + R_y \sin \eta) \\ \left(R_x = \frac{ak}{h_1}, R_y = \frac{bk}{h_2} \right) \\ |\gamma|^2 &= 1 + R_x^2 \sin^2 \xi + R_y^2 \sin^2 \eta \geq 1. \end{aligned} \quad (18.65)$$

We thus see that this innocuous scheme is unconditionally unstable! The problem is that some centred difference schemes are not suitable for this kind of problem. Instead, the first-order upwinding schemes produce better results, as we shall now see. The scheme is:

$$\frac{U_{i,j}^{n+1} - U_{i,j}^n}{k} + \frac{a}{h_1} (U_{i,j}^n - U_{i-1,j}^n) + \frac{b}{h_2} (U_{i,j}^n - U_{i,j-1}^n) = 0. \quad (18.66)$$

Calculation shows that the symbol is:

$$\gamma = \gamma(\xi, \eta) = 1 - R_x(1 - e^{-i\xi}) - R_y(1 - e^{-i\eta}) \quad (18.67)$$

and that it is less than one in absolute value if:

$$\begin{aligned} 0 &\leq R_x + R_y \leq 1 \\ R_x &\geq 0, R_y \geq 0. \end{aligned} \quad (18.68)$$

We now try to derive the same result based on positivity arguments. From Equation (18.66), the value at level $n + 1$ can be written in terms of the solution at level n as follows:

$$U_{i,j}^{n+1} = \left(1 - \frac{ak}{h_1} - \frac{bk}{h_2}\right) U_{i,j}^n + \frac{a}{h_1} U_{i-1,j}^n + \frac{b}{h_2} U_{i,j-1}^n. \quad (18.69)$$

We would like to define sufficient conditions for the right-hand side of Equation (18.69) to be positive. This criterion thus leads to the inequality:

$$1 - \frac{ak}{h_1} - \frac{bk}{h_2} \geq 0, \text{ or } R_x + R_y \leq 1 \quad (18.70)$$

and this reproduces the inequality in Equations (18.68). First-order hyperbolic problems are important in the Black–Scholes environment because we need to model them in Asian option pricing problems and basket option models, for example.

Initial Boundary Value Problems

A new challenge arises when we wish to approximate the solution of first-order hyperbolic initial boundary value problems. The theory is well developed (see, for example, Friedrichs (1958)), and knowing where to place the boundary conditions is important when we model Asian options and the convective terms in the Black–Scholes PDE, for example. Let us consider Equation (18.60) in the rectangle:

$$\begin{aligned} 0 \leq x \leq L \\ 0 \leq y \leq M \end{aligned} \quad (18.71)$$

where a and b are positive and the boundary conditions are specified at the ‘incoming’ boundaries, thus:

$$\begin{aligned} u(0, y, t) &= g(y, t), \quad 0 \leq y \leq M, \quad t > 0 \\ u(x, 0, t) &= h(x, t), \quad 0 \leq x \leq L, \quad t > 0. \end{aligned} \quad (18.72)$$

If we use one-sided upwinding schemes to solve this problem, then everything works fine. If we use centred difference schemes (for example, the scheme in Equation (18.64) and Crank–Nicolson in time), we have to provide a numerical boundary condition on the boundaries that do not have analytic boundary conditions. This has been a source of errors when modelling Asian options in the past. A solution to this problem is to use upwinding schemes. A thorough treatment of numerical boundary conditions is given in Thomas (1995).

18.8 THREE-DIMENSIONAL PROBLEMS

We have already seen that ADI produces a conditionally stable scheme on each leg, but this instability gets balanced out at the next leg. Of course, if there is an uneven number of legs, we will not get stable schemes! Take for example, the scheme for approximating the three-dimensional heat equation:

$$\begin{aligned} \frac{U^{n+1/3} - U^n}{k/3} &= \Delta_x^2 U^{n+1/3} + \Delta_y^2 U^n + \Delta_z^2 U^n \\ \frac{U^{n+2/3} - U^{n+1/3}}{k/3} &= \Delta_x^2 U^{n+1/3} + \Delta_y^2 U^{n+2/3} + \Delta_z^2 U^{n+1/3} \\ \frac{U^{n+1} - U^{n+2/3}}{k/3} &= \Delta_x^2 U^{n+2/3} + \Delta_y^2 U^{n+2/3} + \Delta_z^2 U^{n+1}. \end{aligned} \quad (18.73)$$

In this equation we have suppressed dependence on the space variable for readability reasons. It has been proved that this scheme is not unconditionally stable

(Yanenko (1971)). There are a number of solutions to this problem. First, the *Douglas–Rachford scheme* is:

$$\begin{aligned}\frac{U^{n+1/3} - U^n}{k} &= \Delta_x^2 U^{n+1/3} + \Delta_y^2 U^n + \Delta_z^2 U^n \\ \frac{U^{n+2/3} - U^{n+1/3}}{k} &= \Delta_y^2 (U^{n+2/3} - U^n) \\ \frac{U^{n+1} - U^{n+2/3}}{k} &= \Delta_z^2 (U^{n+1} - U^n).\end{aligned}\tag{18.74}$$

Furthermore, the simplest splitting scheme for this problem is:

$$\begin{aligned}\frac{U^{n+1/3} - U^n}{k} &= \Delta_x^2 U^{n+1/3} \\ \frac{U^{n+2/3} - U^{n+1/3}}{k} &= \Delta_y^2 U^{n+2/3} \\ \frac{U^{n+1} - U^{n+2/3}}{k} &= \Delta_z^2 U^{n+1}.\end{aligned}\tag{18.75}$$

Another problem with the standard ADI method is that it is not applicable to problems with mixed derivatives:

$$\frac{\partial u}{\partial t} = \sum_{i,j=1}^m a_{ij} \frac{\partial^2 u}{\partial x_i \partial x_j}.\tag{18.76}$$

even in the case $m = 2$, because an explicit operator reaches the stability of the scheme. This is bad news for two-factor Black–Scholes problems, where we have correlation between the underlying assets.

18.9 THE HOPSCOTCH METHOD

For completeness, we give an introduction to the *Hopscotch method* (Gourlay (1970)).

We focus on the two-dimensional heat equation (18.1) for convenience. The basic idea is to divide the mesh points in the two-dimensional $x - y$ mesh into those points (ih, jh) as follows:

$$i + j \text{ odd}$$

$$i + j \text{ even.}$$

The Hopscotch consists of two ‘sweeps’. In the first sweep (and subsequent odd-numbered sweeps), the mesh points that are marked by a diamond (see Figure 18.1), that is for which $i + j$ is odd, are calculated based on current values (time level n) at the neighbouring points. We use a Forward in Time, Centred in Space (FTCS) scheme defined as follows:

$$\frac{U_{ij}^{n+1} - U_{ij}^n}{k} = \Delta_x^2 U_{ij}^n + \Delta_y^2 U_{ij}^n \text{ for } (i + j) \text{ odd.}\tag{18.77}$$

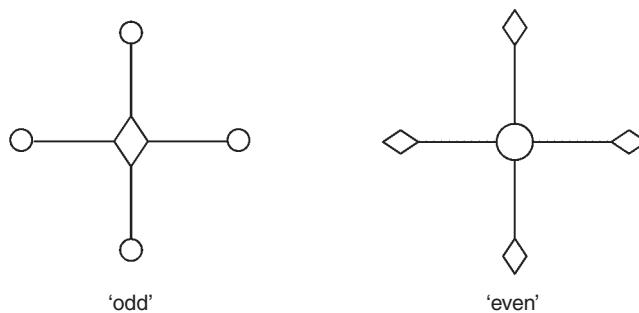


FIGURE 18.1 Hopscotch mesh point.

For the second sweep at the same time level $n + 1$, the same calculation is used at nodes marked with an ‘o’ as shown in Figure 18.1. This second sweep is fully implicit. The scheme is:

$$\frac{U_{ij}^{n+1} - U_{ij}^n}{k} = \Delta_x^2 U_{ij}^{n+1} + \Delta_y^2 U_{ij}^{n+1} \quad (i+j) \text{ even.} \quad (18.78)$$

From this equation we can find the value at time level $n + 1$ as follows:

$$U_{ij}^{n+1} = \frac{\left[U_{ij}^n + k \left(U_{i+1,j}^{n+1} + U_{i-1,j}^{n+1} \right) + k \left(U_{i,j+1}^{n+1} + U_{i,j-1}^n \right) \right]}{\left[1 + \frac{2k}{h_x^2} + \frac{2k}{h_y^2} \right]}. \quad (18.79)$$

In the second and subsequent even-numbered time steps, the roles of the diamonds and ‘o’ are interchanged.

Some remarks on the Hopscotch method are in order.

- It can be applied to convection-diffusion equations, and the scheme is unconditionally stable if upwind (one-sided) differencing is used for approximating the first-order derivative terms (see Gourlay (1970)).
- The method is three to four times faster than the Peaceman–Rachford method due to the absence of tridiagonal inversions.
- The method has been applied to problems with mixed derivatives, but this fact does not seem to be well documented in the literature.
- How would you apply Hopscotch to problems in three space dimensions? (The neighbouring points live in a cube.)

The devil is in the details. It seems that the Hopscotch method is not widely used in practice.

18.10 SOFTWARE DESIGN AND IMPLEMENTATION GUIDELINES

We now discuss how to solve ADI and splitting systems. For example, we set up the systems of equations, then we describe the solution using some kind of pseudo-code, and finally we map this pseudo-code to C++. Since ADI is essentially a method for solving an n -dimensional problem as a series of (simpler) one-dimensional problems, we can use results from previous chapters.

We develop the algorithm that describes the ADI and splitting schemes. We discuss the algorithm that gets us from the solution at time level n to the solution at level $n + 1$. We first describe the algorithm in general terms. Once we have done that, we then describe the algorithm in more detail so that the *cognitive distance* between this level of detail and C++ is not too great.

The first-cut algorithm for solving the heat equation is described by a series of activities:

- A1: Calculate the right-hand side (RHS) of Equation (18.56 (a)).
- A2: Create the stencil (system of equations) for (18.56 (a)).
- A3: Solve system of equations (by *LU* decomposition, for example).
- A4: Calculate the right-hand side (RHS) of equation (18.56 (b)).
- A5: Create the stencil (system of equations) for (18.56 (b)).
- A6: Solve the system of equations (by *LU* decomposition, for example).

This approach is based on the algorithms in Thomas (1995). Each of the above activities has input and output. There is a lot of commonality between the eventual code that implements legs 1 and 2. In fact they share the following common steps:

- Calculate a RHS vector (A1 and A4).
- Create a tridiagonal matrix (A2 and A5).
- Solve the tridiagonal system (A3 and A6).

18.11 THE FUTURE: CONVECTION-DIFFUSION EQUATIONS

A convection-diffusion equation in n dimension contains both diffusion and convection terms, and these equations have received much attention in the engineering literature in the last 50 years because they model many kinds of physical problems. In financial engineering, we view the Black–Scholes equation as an instance of a convection-diffusion-reaction equation:

$$-\frac{\partial C}{\partial t} + \frac{1}{2} \sum_{i,j=1}^n \frac{\partial^2 C}{\partial S_i \partial S_j} \rho_{ij} \sigma_i \sigma_j S_i S_j + \sum_{i=1}^n r S_i \frac{\partial C}{\partial S_i} - r C = 0. \quad (18.80)$$

In this case we have n underlying assets, and C is the contingent claim. We note the presence of mixed derivatives if the assets are correlated, and our resulting finite difference schemes must also produce accurate approximations to these terms.

An example of Equation (18.80) is with $n = 2$. In this case we model an option with two underlying assets. In particular, we can mode the following kinds of options:

- The difference of two assets (spread option).
- Options on the maximum or minimum of two assets.

In this case the partial differential equation (a specialisation of Equation (18.80)) is given by:

$$\begin{aligned} \frac{\partial C}{\partial t} &= (r - D_1)S_1 \frac{\partial C}{\partial S_1} + (r - D_2)S_2 \frac{\partial C}{\partial S_2} \\ &+ \frac{1}{2}\sigma_1^2 S_1^2 \frac{\partial^2 C}{\partial S_1^2} + \frac{1}{2}\sigma_2^2 S_2^2 \frac{\partial^2 C}{\partial S_2^2} + \rho\sigma_1 S_1 \sigma_2 S_2 \frac{\partial^2 C}{\partial S_1 \partial S_2} - rC. \end{aligned} \quad (18.81)$$

We discuss multi-asset options in more detail in Chapter 23.

We can transform this equation to a simpler form:

$$\frac{\partial C}{\partial t} = v_1 \frac{\partial C}{\partial x_1} + v_2 \frac{\partial C}{\partial x_2} + \frac{1}{2}\sigma_1^2 \frac{\partial^2 C}{\partial x_1^2} + \frac{1}{2}\sigma_2^2 \frac{\partial^2 C}{\partial x_2^2} + \rho\sigma_1 \sigma_2 \frac{\partial^2 C}{\partial x_1 \partial x_2} - rC \quad (18.82)$$

where $v_1 = r - D_1 - \frac{1}{2}\sigma_1^2$ and $v_2 = r - D_2 - \frac{1}{2}\sigma_2^2$ namely, by the change of variables:

$$x_1 = \log(S_1) \quad x_2 = \log(S_2)$$

where $v_1 = r - D_1 - \frac{1}{2}\sigma_1^2$ and $v_2 = r - D_2 - \frac{1}{2}\sigma_2^2$.

18.12 SUMMARY AND CONCLUSIONS

In this chapter we gave an introduction to several methods to approximate the solution of multi-dimensional time-dependent PDEs by breaking them into a sequence of simpler one-dimensional PDEs. In a sense, it is an overview as well as a short history of some well-known *splitting methods* for the two-dimensional heat equation and generic convection-diffusion-reaction equations that are documented in textbooks (Thomas (1995), Marchuk (1982), Peaceman (1977)) and Yanenko (1971) and finance journals, for example. They are not directly applicable to computational finance in their current form. Extra analysis and methods are needed (and that is the reason for writing this book); in particular, we extend these methods in Chapters 22 and 23 to approximating the solution of two-factor Black–Scholes PDEs. In particular, we shall extend current documented results with new findings and insights:

- The ability to approximate linear and non-linear convection-diffusion-reaction equations that are needed in finance.
- A new approach to mapping PDEs to a form that allows us to discover relevant boundary conditions in a more mathematical and predictable way (less hand waving and less ad hoc heuristics).

- How to accurately approximate or remove mixed-derivative (correlation) terms in PDEs.
- Using mathematical and numerical methods that are not well-known in mainstream finance applications (avoid becoming a ‘one-trick pony’).
- Alternatives to splitting methods such as Alternating Direction Explicit (ADE) method (Chapter 19) and the Method of Lines (MOL) (Chapter 20).
- Splitting convection-diffusion-reaction equations methods should ‘work both on paper and in the computer’. An important and often insufficiently documented aspect is to show how to design the algorithms and data structures that implement splitting methods. We discuss this topic in Chapter 23 for the case of spread options, and the findings can be transferred to other two-factor problems.

These methods have also been applied by my MSc students (approximately 40 students since 2014) for a range of problems in finance.

The Alternating Direction Explicit (ADE) Method

Richardson describes how 'A myriad computers are at work upon the weather of the part of the map where each sits The work of each region is coordinated by an official of higher rank.' The image divides the globe into about twenty zones.

The upper floor, with the desks of four senior clerks, is shown below the central column. A banner on each desk identifies a major figure in the history of computing. The Director of Operations stands on a dais atop the central tower, 'like the conductor of an orchestra in which the instruments are slide-rules and calculating machines'. He coordinates the computations, signalling with a red spotlight to those who are racing ahead, and with a blue light to those who are behindhand.

L. F. Richardson, *Weather Prediction by Numerical Process*

19.1 INTRODUCTION AND OBJECTIVES

In this chapter we introduce the Alternating Direction Explicit (ADE) method and its applications to solving time-dependent PDEs, in particular Black–Scholes-style equations. The method was invented by V.K. Saul'yev (Saul'yev (1964)) who applied it to solving time-dependent diffusion problems. It is worth noting that ADE probably predicated the invention of ADI and Soviet Splitting methods. It differs in that it does not break a multi-dimensional problem into a sequence of one-dimensional problems but is a stable explicit method that solves multi-dimensional problems 'directly' without splitting. The method has been used in many engineering applications down the years (Tannehill, Anderson and Pletcher (1997), Roache (1998)) and it has relatively recently made its way into finance (Duffy (2009a), Pealat and Duffy (2011), Buchova, Ehrhardt and Guenther (2015)). In fact, it was the author of this book who first introduced the ADE, Soviet splitting and exponential fitting methods to finance.

ADE is not a single method but a family of methods that modify or extend the basic Saul'yev method for the diffusion equation. The method names are attributed to their originators, and we shall introduce them later in this chapter. They came into being

in order to solve convection-diffusion equations and to improve the accuracy of the Saul'yev method. We discuss these methods, and we extend them to one-factor and two-factor PDEs in finance. (We mention in passing that a number of the author's MSc students in the UK and USA have done theses in recent years on ADE applied to a wide range of finance problems.) Having decided to use ADE, we can still analyse it using the techniques already discussed in this book, for example domain transformation, domain truncation, boundary conditions, the Fichera theory and von Neumann stability analysis.

To conclude this section, we summarise some of the perceived advantages of the ADE method in the pantheon of numerical methods to approximate initial boundary value problems:

- Several ADE variants (notably Barakat and Clark) are explicit, second-order accurate in space and time, and unconditionally stable. Thus, they compete with the explicit Euler method with regards to ease of implementation on the one hand and with the splitting ADI and Crank–Nicolson methods with regards to robustness and accuracy on the other hand.
- In contrast to ADI, ADE does not have the burden of having to split a two-factor problem into two one-factor problems, which can lead to unwanted splitting errors. Further issues to address are whether component or operator splitting is more appropriate and how to accommodate mixed (cross) derivative terms. In other words, ADE does not incur a splitting error because no splitting takes place!
- Some anecdotal evidence suggests that ADE is 40% faster than ADI and Crank–Nicolson (Duffy (2018), Pealat and Duffy (2011), Larkin (1964)). The original Saul'yev method is the fastest at the cost of lower accuracy that may or may not be acceptable in a particular application.
- ADE is easier to use with non-linear problems, for example *uncertain volatility* (Pealat and Duffy (2011)) and PDEs that have semi-linear terms when we introduce penalty terms for early exercise (Duffy (2018)).
- It is yet another method to add to our arsenal of finite difference methods. It is easy for novices to learn ADE, and it can be used to solve initial boundary value problems without too much hassle.
- The separate sweeps of the Barakat–Clark ADE method can be executed independently of each other and are thus amenable to parallelisation. ADI and splitting methods, on the other hand, are not easy to parallelise.
- It is important to mention that it is advisable to take the number of time steps to be a (small) multiple of the number of space steps, typically $NT = 4NX$. In this regard, spatial amplification of errors can occur when NX is much greater than NT (see Campbell and Yin (2007) for a detailed discussion). Numerical experimentation in addition to a theoretical analysis is needed in order to discover the optimal parameters for a given problem.

In general, an effective way to learn ADE is to apply it to one-dimensional diffusion equations and then one-dimensional convection-diffusion equations. Ideally, domain transformation and Dirichlet-style boundary conditions should be used. Having done that, the transition to two-dimensional problems is relatively easy.

19.2 BACKGROUND AND PROBLEM STATEMENT

The ADE method has a rich history, and it has been used in many applications in fluid dynamics, heat transfer and engineering. It predates both ADI and splitting methods. It is less well known in finance. In this section we give a short overview of the most influential articles that we base our results on. A good place to start is Campbell and Yin (2007), which analyses the stability and accuracy of ADE variants for one-dimensional convection-diffusion PDEs. It deals with many practical issues related to the method. The fundamental variants for the *diffusion equation* are:

- *Saul'yev*: a two-sweep scheme using a single array to hold the data at consecutive time levels n and $n + 1$ (Saul'yev (1964)).
- *Barakat and Clark* (B&C): a two-sweep scheme in which the *left-to-right* (LR) and *right-to-left* (RL) sweeps are executed/marched in parallel. The two sweeps are averaged to provide the final value. The method is unconditionally stable and second-order accurate in space and time (Barakat and Clark (1966)).
- *Larkin*: a variant of B&C but less accurate than B&C (Larkin (1964)).

For *convection equations* we propose two methods:

- *Towler–Yang*: a variant of the standard second-order three-point scheme for both LR and RL sweeps (Towler and Yang (1978)).
- *Roberts–Weiss*: another ADE scheme to approximate the convection term (Roberts and Weiss (1966)).

In general, the Barakat and Clark scheme in combination with Roberts–Weiss (or Towler–Yang) is good practice. However, the Saul'yev method has good run-time performance. These methods are easy to generalise to two-factor models. As far as we know, the first discussion of ADE for problems in computational finance is to be found in Duffy (2009a).

19.3 GLOBAL OVERVIEW AND APPLICABILITY OF ADE

Before we jump into the details of the ADE method, we pause to reflect on the types of PDEs that interest us and the various finite difference methods that compute approximations to their solutions. In particular, we realise that there are many kinds of PDEs and many finite difference schemes (and variants) that can be applied to them! It is not possible to discuss them all in equal detail; there are too many options. Conceptually, we can write the relationships as follows:

- A generic PDE can have several forms (non-conservative, conservative, ...).
- A generic PDE can be domain transformed or domain truncated.
- A generic PDE has multiple instance PDEs (multi-asset, interest rate, convertible bonds, ...).

- There are several FD schemes to approximate generic PDEs (ADE, ADI, MOL, splitting).
- A given FD scheme has several variants (usually named after their inventors).

Having too much choice can make people nervous, but in general we usually focus on a single problem and a single finite difference scheme at a given time, and this allows us to reduce the scope. Finally, these remarks have relevance to both one-factor and two-factor problems, as we saw in previous chapters. We now give a quick checklist:

- S1: Employ domain truncation and domain transformation.
- S2: PDE in conservative and non-conservative forms.
- S3: Simplify the PDE by transforming its coefficients.
- S4: Barakat–Clark, Larkin and Saul’ev methods for diffusion.
- S5: Towler–Yang, Roberts–Weiss and upwinding for convection.
- S6: Constant versus non-constant meshes.
- S7: Yanenko strategy for mixed derivatives.
- S8: Use of extrapolation methods.
- S9: Exponential fitting for convection-dominated problems.

Having chosen for a particular PDE and associated ADE variant, we can consider testing the following use cases:

- A1: Large expiration (similar to an elliptic problem, infinite time).
- A2: Convection dominance.
- A3: Analogous, simpler cases.
- A4: Choice of boundary conditions.
- A5: Correlation with values in the range $(-1, 1)$.

Having an exact solution is ideal but not always possible, and in these cases we can run two (or more) variants of ADE in parallel and compare the output.

19.4 MOTIVATING EXAMPLES: ONE-DIMENSIONAL AND TWO-DIMENSIONAL DIFFUSION EQUATIONS

The ADE family of methods differs in a number of subtle ways from the well-known finite difference schemes that we have already discussed in this book. First, they are explicit and stable, and the resulting discrete system of equations can be solved without the need to use matrix solvers. Second, some ADE variants are only *conditionally consistent* with a given PDE, which means that the step sizes in space and time must be related to each other if the ADE scheme is to converge to the PDE. Finally, ADE uses updated information (in particular, known values on the boundaries of the spatial domain) as soon as it becomes available. In the one-dimensional case, we solve the

system of equations by ‘sweeping’ from the left boundary to the right boundary and then from the right boundary to the left boundary. This principle generalises to higher dimensions.

Our goal in this chapter is to apply ADE to convection-diffusion equations and to applications in computational finance. To this end, we proceed by examining diffusion and convection equations in turn. We then combine them, as it were, to see how ADE works in the case of convection-diffusion equations. In the following subsections we take the one-dimensional heat equation as test case.

19.4.1 Barakat and Clark (B&C) Method

We discuss a one-factor, time-independent diffusion equation, and in particular we are interested in approximating the diffusion term by finite differences. To this end, we consider a non-constant partition $\{x_0, x_1, \dots, x_J\}$ with $x_0 = 0, x_J = 1$ of the unit interval as shown in Figure 19.1. We then consider approximating the diffusion operator as follows:

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial}{\partial x} \left(\frac{\partial u}{\partial x} \right) \approx \frac{\frac{u_{j+1} - u_j}{h_{j+1}} - \frac{u_j - u_{j-1}}{h_j}}{(h_j + h_{j+1})/2}. \quad (19.1)$$

We can see this approximation as a divided difference of a divided difference, and in the case of a constant mesh size h , it reduces to the well-known centred-difference formula:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{j+1} - u_j - u_j + u_{j-1}}{h^2} = \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2}. \quad (19.2)$$

This concept will form the basis for the ADE method; in general, some terms in Equations (19.1) and (19.2) will be evaluated at time level n (known values), while other terms will be taken at time level $n + 1$ (both known and unknown values). To this end, we first consider the heat equation on the interval $(0, 1)$ with Dirichlet boundary conditions:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < 1, t > 0 \\ u(x, 0) &= f(x), \quad 0 < x < 1 \\ u(0, t) &= g_0(t), \quad u(1, t) = g_1(1, t). \end{aligned} \quad (19.3)$$

We now apply ADE to (19.3) by first ‘sweeping’ from the left boundary $x = 0$ to produce a kind of *predictor* solution, and then we sweep from the right boundary

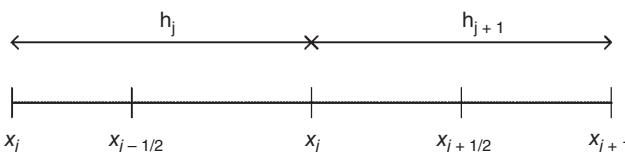


FIGURE 19.1 Non-uniform mesh.

$x = 1$ to produce a *corrector* solution. Having done that, we average these subsolutions to produce the desired approximation to the solution of system (19.3) at the time level $n + 1$:

$$\begin{aligned}\frac{U_j^{n+1} - U_j^n}{k} &= \frac{1}{h^2}(U_{j+1}^n - U_j^n - U_j^{n+1} + U_{j-1}^{n+1}), \quad 1 \leq j \leq J-1, \quad n \geq 0 \text{ ('up sweep')} \\ \frac{V_j^{n+1} - V_j^n}{k} &= \frac{1}{h^2}(V_{j+1}^{n+1} - V_j^{n+1} - V_j^n + V_{j-1}^n), \quad J-1 \geq j \geq 1, \quad n \geq 0 \text{ ('down sweep')}.\end{aligned}\tag{19.4}$$

These two subsolutions are aggregated to produce the final solution:

$$u_j^n = \frac{1}{2}(U_j^n + V_j^n), \quad 0 \leq j \leq J, \quad n \geq 0.$$

The B&C scheme (Barakat and Clark (1966)) can be rewritten in such a way that all terms on the right-hand side are known quantities:

$$U_j^{n+1}(1 + \lambda) = U_j^n(1 - \lambda) + \lambda(U_{j+1}^n + U_{j-1}^{n+1}), \quad \left(\lambda = \frac{k}{h^2}\right)$$

and:

$$V_j^{n+1}(1 + \lambda) = V_j^n(1 - \lambda) + \lambda(V_{j+1}^{n+1} + V_{j-1}^n).$$

As already mentioned, the final solution involves averaging the two sweep solutions.

This method is unconditionally stable, and the truncation error is $O(h^2, dt^2)$ because the two simultaneous sweeps cancel the $(dt/h)^2$ terms which are present in the Saul'yev method (Barakat and Clark (1966)). For the two-dimensional heat equation, the B&C method is 18/16 (1.125) times faster than the ADI method (Tannehill, Anderson and Pletcher (1997)).

19.4.2 Saul'yev Method

This is the original ADE method (Saul'yev (1964)). It can be seen as a special case of ADE B&C method in the sense that it uses two sweeps (left-to-right and right-to-left) at consecutive time levels, but no averaging is performed, and there is only one array in memory that gets updated at each time level:

$$\begin{aligned}\frac{u_j^{n+1} - u_j^n}{k} &= \frac{u_{j-1}^{n+1} - u_j^{n+1} - u_j^n + u_{j+1}^n}{h^2} \\ \frac{u_j^{n+2} - u_j^{n+1}}{k} &= \frac{u_{j-1}^{n+1} - u_j^{n+1} - u_j^{n+2} + u_{j+1}^{n+2}}{h^2}.\end{aligned}$$

This method is unconditionally stable, and the truncation error is $O(h^2, dt^2, (dt/h)^2)$. It is formally first-order accurate due to the presence of the *inconsistent term*

$(dt/h)^2$. We have seen it to be less accurate but much faster than the B&C method. We have also found that it produces reasonably accurate results for Asian-style options (Wilmott, Lewis and Duffy (2014)). In general, it is a trade-off between speed and accuracy as to whether to use B&C or the Saul'yev method. It is worth investigating the trade-offs.

19.4.3 Larkin Method

This is a variant of the B&C method that replaces the up and down sweep arrays by the final solution as soon as it becomes available (Larkin (1964)):

$$\begin{aligned}\frac{U_j^{n+1} - u_j^n}{k} &= \frac{U_{j-1}^{n+1} - U_j^{n+1} - u_j^n + u_{j+1}^n}{h^2} \\ \frac{V_j^{n+1} - u_j^n}{k} &= \frac{u_{j-1}^n - u_j^n - V_j^{n+1} + V_{j+1}^{n+1}}{h^2} \\ u_j^{n+1} &= \frac{1}{2} (U_j^{n+1} + V_j^{n+1}).\end{aligned}$$

We have seen that this scheme tends to be less accurate than the B&C method. It is not clear to the author when it would be advantageous to use this method.

19.4.4 Two-Dimensional Diffusion Problems

The application of ADE to two-dimensional diffusion problems is relatively straightforward, and it essentially entails the application of the previous methods to each coordinate direction. In this case the sweeps take place in a rectangle using the B&C method. We sweep from the *left lower apex* of the rectangle, followed by a sweep from the *upper right apex*. Two sweeps are sufficient, even in n dimensions.

In the interest of completeness, we discuss the applicability of ADE to two-factor PDEs. To this end, we consider the two-dimensional heat equation with inhomogeneous forcing term $f(x, y, t)$ and Dirichlet boundary conditions on the unit square:

$$\begin{aligned}\frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + f(x, y, t) \text{ in } (0, 1)^2 \times (0, T) \\ u(x, y) &= g(x, y) \text{ on boundary of } (0, 1)^2 \\ u(x, y, 0) &= h(x, y) \text{ (initial condition).}\end{aligned}\tag{19.5}$$

In the case in the quarter plane $(0, \infty) \times (0, \infty)$, we have used a domain transformation:

$$z = \frac{x}{x + \alpha_1}, w = \frac{y}{y + \alpha_2} \text{ where } \alpha_1 \text{ and } \alpha_2 \text{ are user-defined parameters.}$$

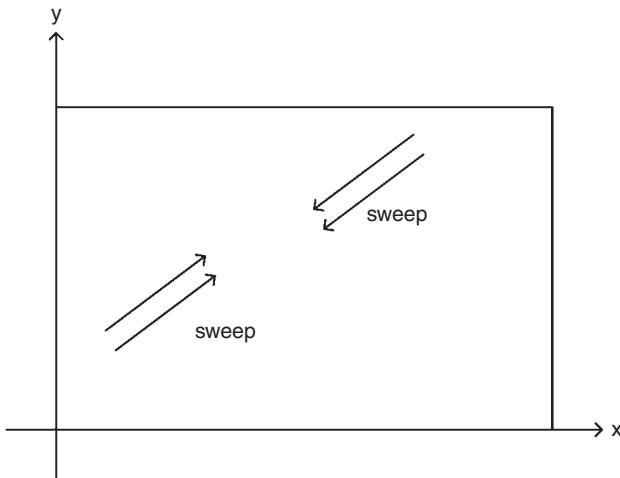


FIGURE 19.2 Visualisation of ADE process.

We now consider a two-dimensional mesh and corresponding discrete mesh functions indexed by indices i and j in the directions x and y , respectively. A straightforward generalisation of the Barakat–Clark method to two dimensions is:

$$\begin{aligned}\frac{U_{i,j}^{n+1} - U_{i,j}^n}{k} &= \frac{1}{h_1^2} (U_{i-1,j}^{n+1} - U_{i,j}^{n+1} - U_{i,j}^n + U_{i+1,j}^n) + \frac{1}{h_2^2} (U_{i,j-1}^{n+1} - U_{i,j}^{n+1} - U_{i,j}^n + U_{i,j+1}^n) \\ \frac{V_{i,j}^{n+1} - V_{i,j}^n}{k} &= \frac{1}{h_1^2} (V_{i-1,j}^n - V_{i,j}^n - V_{i,j}^{n+1} + V_{i+1,j}^{n+1}) + \frac{1}{h_2^2} (V_{i,j-1}^n - V_{i,j}^n - V_{i,j}^{n+1} + V_{i,j+1}^{n+1}) \\ u_{i,j}^n &= \frac{1}{2} (U_{i,j}^n + V_{i,j}^n)\end{aligned}\tag{19.6}$$

where h_1 and h_2 are the mesh sizes in the x and y directions, respectively.

This scheme consists of two sweeps in the unit square; the first sweep uses boundary conditions starting at apex $(0,0)$, while the second sweep uses boundary conditions bases starting at apex $(1,1)$. See Figure 19.2.

Only two sweeps are needed. Ideally, Dirichlet boundary conditions are advised. See Section 19.6.2 for a discussion of other kinds of boundary conditions. An interesting initial exercise would be to apply ADE to the three-dimensional heat equation.

19.5 ADE FOR CONVECTION (ADVECTION) EQUATION

There are several ways to approximate the convection term in the convection-diffusion equation. As before, we employ two sweeps. Let us consider the convection equation $\frac{\partial u}{\partial t} = b \frac{\partial u}{\partial x}$. First, we discuss the Towle–Yang approximation (Towler and Yang (1978)).

The LR (left-to-right) and RL (right-to-left) sweeps are:

$$\begin{aligned}\frac{U_j^{n+1} - U_j^n}{k} &= \frac{b}{2h} (U_{j+1}^n - U_{j-1}^{n+1}), \quad 1 \leq j \leq J-1 \\ \frac{V_j^{n+1} - V_j^n}{k} &= \frac{b}{2h} (V_{j+1}^{n+1} - V_{j-1}^n), \quad J-1 \geq j \geq 1 \\ u_j^n &= \frac{1}{2} (U_j^n + V_j^n).\end{aligned}\tag{19.7}$$

Second, the *Roberts–Weiss method* (Roberts and Weiss (1966), Piacsek and Williams (1970)) is an averaging method:

$$\begin{aligned}\frac{\partial U^{n+1}}{\partial x} &\approx \left(\frac{U_{j+1}^n - U_j^n}{h} + \frac{U_j^{n+1} - U_{j-1}^{n+1}}{h} \right) / 2 \\ \frac{\partial V^{n+1}}{\partial x} &\approx \left(\frac{V_{j+1}^{n+1} - V_j^{n+1}}{h} + \frac{V_j^n - V_{j-1}^n}{h} \right) / 2.\end{aligned}\tag{19.8}$$

The Roberts–Weiss is a better approximation, and we have used it in Pealat and Duffy (2011). Each sweep is first-order accurate in time. See Campbell and Yin (2007) for more details.

We discuss these schemes in more detail (Campbell and Yin (2007)). The Towler–Yang scheme is unconditionally stable when applied to first-order hyperbolic wave PDEs because the amplification factors for the two sweeps are:

$$\begin{aligned}\gamma_1 &= \frac{1 - \frac{\lambda}{2} e^{i\alpha h}}{1 - \frac{\lambda}{2} e^{-i\alpha h}}, \quad |\gamma_1| = 1, \quad \lambda = \frac{b\Delta t}{h} \\ \gamma_2 &= \frac{1 + \frac{\lambda}{2} e^{-i\alpha h}}{1 + \frac{\lambda}{2} e^{i\alpha h}}, \quad |\gamma_2| = 1.\end{aligned}$$

The Roberts–Weiss is unconditionally stable when applied to the first-order wave equation because of its amplification factors:

$$\begin{aligned}\gamma_1 &= \frac{1 + \frac{\lambda}{2} - \frac{\lambda}{2} e^{i\alpha h}}{1 + \frac{\lambda}{2} - \frac{\lambda}{2} e^{-i\alpha h}}, \quad |\gamma_1| = 1 \\ \gamma_2 &= \frac{1 - \frac{\lambda}{2} + \frac{\lambda}{2} e^{-i\alpha h}}{1 - \frac{\lambda}{2} + \frac{\lambda}{2} e^{i\alpha h}}, \quad |\gamma_2| = 1.\end{aligned}$$

The Roberts–Weiss method can be used as both a one-step method and a two-step method. The truncation errors are $O(dt)$. We have used it in Pealat and Duffy (2011) and have found that it gives better results than the Towler–Yang scheme. We have also used it in combination with ADE B&C in Wilmott, Lewis and Duffy (2014).

19.6 CONVECTION-DIFFUSION PDEs

In this section we focus on generic one-factor PDEs of the form:

$$\frac{\partial u}{\partial t} = a(x, t) \frac{\partial^2 u}{\partial x^2} + b(x, t) \frac{\partial u}{\partial x} + c(x, t)u + f(x, t), \quad 0 < x < \infty, \quad 0 < t < T. \quad (19.9)$$

In this case the variable t represents (increasing) time, while the variable x represents a spatial dimension. We shall discuss specialisations of (19.9) in later chapters, where the variable x has a special meaning (for example, it could represent an interest rate or a stock price) and the coefficients $a(x, t)$, $b(x, t)$ and $c(x, t)$ have specific forms. In order to specify (19.9), we first need to augment it with an initial condition:

$$u(x, 0) = f(x), \quad 0 < x < \infty \quad (19.10)$$

and we must specify some kind of boundary or asymptotic behaviour when $x = 0$ and $x = \infty$.

Equation (19.9) is called a *convection-diffusion-reaction equation* with an initial condition given by Equation (19.10). We are particularly interested in applications in which the independent variable x is defined on the positive semi-infinite real axis. Since we are interested in solving PDE (19.9) using numerical methods, we must decide how to replace this semi-infinite interval by a bounded one. The two main choices are *domain truncation* (which is popular in computational finance, see Kangro and Nicolaides (2000)) and *domain transformation* that we now discuss. In particular, we transform the positive semi-infinite real axis to the unit interval by some kind of mapping. There are many choices, for example:

$$y = \frac{x}{x + \alpha} \quad (19.11)$$

where α is a scale factor that can be specified by the user. Some properties that we need later are:

$$\begin{aligned} x &= \frac{\alpha y}{1 - y} \\ \frac{\partial u}{\partial x} &= \alpha^{-1}(1 - y)^2 \frac{\partial u}{\partial y} \\ \frac{\partial^2 u}{\partial x^2} &= \alpha^{-2}(1 - y)^2 \frac{\partial}{\partial y} \left\{ (1 - y)^2 \frac{\partial u}{\partial y} \right\}. \end{aligned} \quad (19.12)$$

Using these features, we see that the PDE in the variables (x, t) can be transformed to a PDE in the variables (y, t) where the variable y is now defined in the unit interval

(0, 1). The general form of the transformed PDE is:

$$\frac{\partial u}{\partial t} = A(y, t) \frac{\partial}{\partial y} \left((1-y)^2 \frac{\partial u}{\partial y} \right) + B(y, t) \frac{\partial u}{\partial y} + c(y, t)u + f(y, t), \quad 0 < y < 1, \quad 0 < t < T \quad (19.13)$$

where:

$$A(y, t) = a(y, t)\alpha^{-2}(1-y)^2$$

$$B(y, t) = b(y, t)\alpha^{-1}(1-y)^2.$$

It is now possible to differentiate the diffusion term in Equation (19.13) with respect to y in order to give a convection-diffusion-reaction equation in standard form as in Equation (19.9).

Another way to modify Equation (19.9) is to first transform it to *conservative form* using the *integrating factor method*. In this case we are able to merge the diffusion and convection terms into a modified diffusion term. The convection term is thus effectively eliminated, and this can be advantageous when approximating the PDE by the finite difference method. In other words, the PDE (19.9) becomes:

$$\frac{\partial u}{\partial t} = \alpha(x, t) \frac{\partial}{\partial x} \left(\beta(x, t) \frac{\partial u}{\partial x} \right) + c(x, t)u + f(x, t) \quad (19.14)$$

where the functions $\alpha(x, t)$ and $\beta(x, t)$ can easily be calculated in many cases. Subsequently, we can use domain transformation to transform PDE (19.14) on a semi-infinite interval to one on the unit interval. This transformation does not alter the basic form of the PDE; that is, it is still a conservative PDE without a convection term. It would be an interesting project to solve this problem using the finite difference method.

Irrespective of which one-factor or two-factor PDE is used, ADE can then be used to approximate it. In particular, the Barakat–Clarke method (19.4) for diffusion and either Towler–Yang (19.7) or Roberts–Weiss (19.8) for convection are recommended, and they function as ‘building blocks’ for PDEs such as (19.9) and general semi-linear PDEs of the form:

$$\frac{\partial u}{\partial t} = a \frac{\partial}{\partial x} \left(b \frac{\partial u}{\partial x} \right) + \left(c \frac{\partial u}{\partial x} \right) + bu = f(u)$$

where $a = a(x, t)$, $b \equiv b(x, t)$, $c = c(x, t)$.

The generalisation to two-factor problems is essentially a composition of one-dimensional solutions using what we could call a *building block approach*. We have tested this approach on many two-factor problems.

19.6.1 Example: Black–Scholes PDE

In the previous section we discussed several techniques to modify a PDE to make it more amenable to approximation by the finite difference method. We summarise these techniques as follows:

1. Domain truncation resulting in a PDE on a finite domain $[0, S_{\max}]$.
2. Domain transformation resulting in a PDE on the unit interval $[0, 1]$.

3. Transforming the PDE to conservative form and using domain truncation.
4. Transforming the PDE to conservative form and using domain transformation.

As an example, we examine the Black–Scholes PDE:

$$\frac{\partial V}{\partial t} = \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV, \quad 0 < S < \infty, \quad 0 < t < T. \quad (19.15)$$

Option 2 implies applying the transformation (19.11), and we then get a PDE:

$$\frac{\partial V}{\partial t} = \frac{1}{2}\sigma^2 y^2 \frac{\partial}{\partial y} \left\{ (1-y)^2 \frac{\partial V}{\partial y} \right\} + ry(1-y) \frac{\partial V}{\partial y} - rV. \quad (19.16)$$

or equivalently:

$$\frac{\partial V}{\partial t} = \frac{1}{2}\sigma^2 y^2 (1-y)^2 \frac{\partial^2 V}{\partial y^2} + \{ry(1-y) - \sigma^2 y^2 (1-y)\} \frac{\partial V}{\partial y} - rV. \quad (19.17)$$

We now discuss option 4. First, we transform PDE (19.15) to conservative form:

$$\frac{\partial V}{\partial t} = \frac{1}{2}S^2 S^{-A} \frac{\partial}{\partial S} \left\{ S^A \frac{\partial V}{\partial S} \right\} - rV \quad \text{where } A = 2r/\sigma^2. \quad (19.18)$$

This PDE is still defined on the positive real axis, and we now apply the transformation (19.11) to arrive at a conservative PDE on the unit interval, namely:

$$\frac{\partial V}{\partial t} = a(y, t) \frac{\partial}{\partial y} \left\{ b(y, t) \frac{\partial V}{\partial y} \right\} - rV \quad (19.19)$$

where:

$$a(y, t) = \frac{1}{2}\sigma^2(\alpha y)^{2-A}\alpha^{-1}(1-y)^A$$

$$b(y, t) = (\alpha y)^A \alpha^{-1}(1-y)^{2-A}$$

$$A = 2r/\sigma^2.$$

We have now completed our initial discussion of the Black–Scholes PDE. The steps taken to transform it to different forms constitute a repeatable process, and they can be applied to a range of one-factor and multi-factor PDEs. It is our hope that the enthusiastic reader will experiment with these schemes.

19.6.2 Boundary Conditions

In an ideal world, we prefer to reduce the PDE problem to a case in which the solution on the boundary is easily computable in some way; in other words, the numerical boundary conditions are essentially of Dirichlet type. This serendipitous situation is not always possible, and in many cases we may need to define Neumann or linearity boundary

conditions, for example. In general, we resolve these problems by solving a mini ADE system at the boundary. Let us consider the case of the one-factor heat equation with a Robin or Neumann boundary condition at $x = 0$, for example:

$$\frac{\partial u}{\partial x} + au = b, x = 0. \quad (19.20)$$

In this case we solve for the approximate solution at the first two mesh points (the near field) by discretising the heat equation using (19.4) coupled with a discretisation of (19.20):

$$\begin{aligned} \frac{U_1^{n+1} - U_1^n}{\Delta t} &= \frac{1}{h^2} (U_2^n - U_1^n - U_1^{n+1} + U_0^{n+1}) \\ \frac{U_1^{n+1} - U_0^{n+1}}{h} + \frac{a}{2} (U_1^{n+1} + U_0^{n+1}) &= b. \end{aligned} \quad (19.21)$$

Note that we have used the second-order box scheme to approximate the Robin boundary condition at $x = 0$. Regarding Equation (19.21), we have a system of two equations in two unknowns U_0^{n+1} and U_1^{n+1} . Having solved for these unknowns, we are now in a position to execute the left-to-right sweep starting at the first interior mesh point x_1 . For the right-to-left sweep, we compute the value down to mesh point x_1 starting from the last interior mesh point. Then we apply the Box scheme at the boundary, again resulting in a system of two equations in two unknowns similar to the reasoning with Equation (19.21), except that indexing starts at $j = J$ (end point of domain).

The same approach can be applied to other kinds of boundary conditions.

19.6.3 Spatial Amplification Errors

The B&C ADE method is unconditionally stable and second-order accurate. However, the von Neumann method does not pick up *spatial amplification errors* that can be caused by machine round-off, for example (see Roache (1998), Campbell and Yin (2007)). Any error in a boundary condition at the beginning of a left or especially right computational sweep may be amplified as the computation proceeds forward in space (not as the time proceeds).

We take an example of pricing a put option (we will explain the full context in later sections):

```
Option myOption;
myOption.sig = 0.3; myOption.K = 65.0; myOption.T = 0.25;
myOption.r = 0.08; myOption.b = 0.08; myOption.beta = 1.0;
myOption.SMax = 325.0; myOption.type = 'P';
myOption.earlyExercise = false;
```

for fixed $NT = 325$, $NX = \{325, 650, 1300, 2600, 5200, 10400\}$. The put option values get progressively worse as we increase NX while keeping NT fixed; namely the values become $\{5.841754, 5.844434, 5.84294, 5.834118, 5.801135, 5.72656\}$. We should be aware

of this somewhat pathological behavior. We (nor others) have yet resolved this issue. A rule of thumb is to use an upper limit that is $NT \sim 4NX$. On the other hand, keeping NX fixed and increasing NT does not lead to amplification errors. An interesting exercise is to determine whether the original Saul'yev scheme causes spatial amplification errors.

19.7 ATTENTION POINTS WITH ADE

We now give some practical tips and guidelines when testing and debugging ADE schemes. In particular, we need to take a number of issues into account and to be aware of some fundamental facts relating to the method.

The Consequences of Conditional Consistency

In contrast to traditional finite difference methods, we note that the local truncation error associated with the ADE method is not always a polynomial function of the mesh size parameter. Instead, the error can include terms such as $O(\Delta t/h)^2$. This means that accuracy depends on the relative sizes of these parameters. In particular, we get some degradation in accuracy if the step size h is much less than Δt , typically $h \leq 4\Delta t$. But this is hopefully a case that is not too common. In general, taking the mesh sizes to have the same order of magnitude gives good results. And finally, we have the option to define *non-uniform grids* that allow us to achieve better performance near hotspots, for example at the strike price.

Call Pay-Off Behaviour at the Far Field

The boundary conditions for put options are always bounded for both $y = 0$ and $y = 1$. And the same conclusion holds for the pay-off function (initial condition); it is always bounded. For call options, on the other hand, the transformed pay-off function becomes:

$$\max(S - K, 0) = \max\left(\frac{\alpha y}{1-y} - K, 0\right).$$

In this case we see that the pay-off is unbounded at $y = 1$ (and the corresponding boundary condition). In such a case we adopt the heuristic approach by offsetting the boundary by a small amount eps (typically 0.001) and then using $1 - \text{eps}$ as the new boundary point. An alternative is to code our schemes in such a way that they do not need to evaluate the pay-off function on the boundaries.

An open problem is to establish a firm mathematical/PDE foundation for problems with unbounded behaviour on boundaries.

19.7.1 General Formulation of the ADE Method

In this section we give an analysis of the ADE method. To this end, let us discrete Equation (19.3) or Equation (19.5) in the x direction only. This is called the *semi-discretisation process* or the *Method of Lines* (MOL). This process leads to a system

of ordinary differential equation for the dependent vector T (use variable name T for this):

$$\frac{dT}{dt} = AT \equiv (L + D + U)T, \quad t \geq 0 \quad (19.22)$$

where A is the matrix that originates from semi-discretisation that we have decomposed into lower triangular L , diagonal D and upper triangular U submatrices. In the analysis we assume that the boundary values of T are given, and in fact Equation (19.22) is essentially a discretisation of a PDE with Dirichlet boundary conditions.

The next step is to discretise Equation (19.22) in time. This is a well-known process for traditional finite difference methods, but the main difference in the present case is that we construct two schemes by sweeping from the corners of the space domain. In our case we have two sweeps:

$$\frac{T_1^{n+1} - T_1^n}{k} = LT_1^{n+1} + \frac{D}{2}T_1^{n+1} + \frac{D}{2}T_1^n + UT_1^n \equiv BT_1^{n+1} + CT_1^n, \quad n \geq 0 \quad (19.23)$$

and:

$$\frac{T_2^{n+1} - T_2^n}{k} = LT_2^{n+1} + \frac{D}{2}T_2^{n+1} + \frac{D}{2}T_2^n + UT_2^n \equiv BT_2^{n+1} + CT_2^n \quad n \geq 0. \quad (19.24)$$

Finally, the approximate solution is the average of the above two subsolutions:

$$T^n = \frac{1}{2}(T_1^n + T_2^n), \quad n \geq 0. \quad (19.25)$$

These systems are easier to solve than traditional implicit schemes because the value of T at time level $n + 1$ can be computed if we rewrite the systems as:

$$\begin{aligned} (I - kB)T_1^{n+1} &= (I + kC)T_1^n \\ (I - kC)T_2^{n+1} &= (I + kB)T_2^n \end{aligned} \quad (19.26)$$

where $B = L + D/2$ and $C = U + D/2$.

This system is easy to solve because we are working with lower triangular and upper triangular matrices whose inverses can be found by using simple forward and backward substitution. Here we see immediately how generally applicable ADE is. We can rewrite the solution (19.25) in the form by taking the average of the separate subsolutions:

$$T^{n+1} = \frac{1}{2}[(I - kB)^{-1}(I + kC) + (I - kC)^{-1}(I + kB)]T^n, \quad n \geq 0. \quad (19.27)$$

Other approaches can be found in Saul'yev (1964), Barakat and Clark (1966) and Larkin (1964).

Theorem 19.1 The ADE scheme (19.27) is second-order accurate in time if all the diagonal elements of the matrix A in Equation (19.22) are non-positive.

19.8 SUMMARY AND CONCLUSIONS

In this section we enumerated the steps to be executed when setting up the ADE scheme for a general derivatives pricing problem. The end result of this process is an unambiguous description of the finite difference scheme that approximates the PDE in question. We focused mainly on linear one-factor plain options. There are three main activities in this process:

- A1 Preprocessing: prepare the PDE (*continuous problem*) before we apply ADE to it.
 - Use domain truncation?
 - Use domain transformation?
 - Determine boundary behaviour.
 - Other preprocessing: for example, calibrated PDE coefficients, reduction to conservative form.
- A2 Core ADE scheme: define the *discrete problem* by discretising each artifact in activity A1.
 - Approximating the diffusion term.
 - Approximating the convection term.
 - Uniform or non-uniform meshes?
- A3 Assemble the artifacts from activity A2.
 - Explicit representation of the left-to-right and right-to-left sweeps.
 - Explicit representation of the boundary conditions.
 - A final, unambiguous algorithm for the ADE method.

Having executed these activities, we are then in a position to implement the ADE scheme in C++.

Finally, the Appendix in Chapter 7 discussed numerical results for option pricing problems with ADE (code was written in C# in this case). You can use these results as test data for an ADE implementation.

CHAPTER 20

The Method of Lines (MOL), Splitting and the Matrix Exponential

The management question, therefore, is not whether to build a pilot system and throw it away. You will do that. The only question is whether to plan in advance to build a throwaway, or to promise to deliver the throwaway to customers.

Frederick P Brooks Jr, *The Mythical Man-Month: Essays on Software Engineering*

20.1 INTRODUCTION AND OBJECTIVES

In this chapter we introduce a number of techniques to reduce the complexity of time-dependent PDEs and finite difference schemes by transforming them in some way. The transformation may lead to an equation that we already know how to solve, or it may lead to a simpler equation with known properties. We give a number of associated use cases by focusing on the one-dimensional convection-diffusion equation, but the results also extend to multi-dimensional problems. In most cases here, a given transformation converts a PDE to an ODE. The following options are open to us:

- *Vertical Method of Lines (VMOL)*: discretise the PDE in space and keep time continuous. This results in a system of first-order ODEs in time. This is a common and popular method.
- *Horizontal Method of Lines (HMOL)* (also known as *Rothe's method* (Rothe (1930), Ladyženskaja, Solonnikov and Ural'ceva (1988))): discretise the PDE in time and keep space continuous. This results in a second-order two-point boundary problem in space. We do not discuss Rothe's method in this book. A good discussion for finance is given in Meyer (2015).

In both cases we arrive at a system of *difference-differential equations* whose qualitative properties (such as existence, uniqueness and stability) can be analysed using a range of techniques. Finally, we can numerically approximate these difference-differential equations using numerical methods. We also need to consider associated boundary and initial conditions. Some specific and common use cases are:

- U1: discretise in space using standard finite difference schemes (Duffy (2018)).
- U1: semi-discretisation in space using the finite element method (FEM) (Strang and Fix (1973), Achdou and Pironneau (2005)).
- U2: discretise in time using the backward Euler method or the three-level backward difference formula (Meyer (2015)).

There are numerous other techniques to simplify PDEs or reduce their dimensionality in some way, but we do not discuss them here. We discuss the C++ implementation of MOL in Duffy (2018). The MOL approach is very robust, especially if used in combination with commercial and open-source ODE solvers. It can be used as a ‘second opinion’ and support for other finite difference schemes.

20.2 NOTATION AND PREREQUISITES: THE EXPONENTIAL FUNCTION

The *exponential function* is undoubtedly the most important function in mathematics (Rudin (1964), Rudin (1970)). It is defined by the *power series*:

$$\exp(z) = e^z = \sum_{n=0}^{\infty} \frac{z^n}{n!}, \quad z \in \mathbb{C}. \quad (20.1)$$

This series is a special case of a general *power series*:

$$f(z) = \sum_{n=0}^{\infty} c_n z^n \equiv \sum_{n=0}^{\infty} a_n, \quad z \in \mathbb{C}. \quad (20.2)$$

The main question at this stage is to determine for which values of $z \in \mathbb{C}$ the power series is convergent. To this end, we use the *root test* to compute the *radius of convergence R*:

$$\alpha = \limsup_{n \rightarrow \infty} \sqrt[n]{c_n}, \quad R = \frac{1}{\alpha}. \quad (20.3)$$

In the current case, we have $R = \infty$ and hence (20.1) converges for all values in the complex plane. A special case is when $z = 1$, and we get a representation for *Euler’s number*:

$$e = \sum_{n=1}^{\infty} \frac{1}{n!} \equiv \sum_{n=1}^{\infty} d_n. \quad (20.4)$$

We can show that this series is convergent by using the *ratio test* (Rudin (1964)):

$$\lim_{n \rightarrow \infty} \sup \left| \frac{d_{n+1}}{d_n} \right| = \lim_{n \rightarrow \infty} \sup \left| \frac{1}{n+1} \right| < 1.$$

The discovery of the Euler number is credited to Jacob Bernoulli in 1683, who attempted to find the value of the following expression when working on problems related to *compound interest*:

$$e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n} \right)^n. \quad (20.5)$$

Having given a mathematically precise account of the exponential function, we now proceed to discussing some of its interesting properties.

20.2.1 Initial Results

We have the famous and far-reaching *addition formula*:

$$\exp(a) \exp(b) = \exp(a+b) = e^{a+b}, \quad a, b \in \mathbb{C} \quad (20.6)$$

which can be proved as follows:

$$e^a \cdot e^b = \sum_{k=0}^{\infty} \frac{a^k}{k!} \sum_{m=0}^{\infty} \frac{b^m}{m!} = \sum_{n=0}^{\infty} \frac{1}{n!} \sum_{k=0}^n \frac{n!}{k!(n-k)!} a^k b^{n-k} = \sum_{n=0}^{\infty} \frac{(a+b)^n}{n!} = e^{a+b}.$$

The exponential function is its own derivative:

$$\frac{d}{dz} e^z = e^z.$$

Euler's identity is $e^{it} = \cos t + i \sin t$, $i = \sqrt{-1}$.

A generalisation of formula (20.6) to n arguments is:

$$\exp \left(\sum_{j=1}^n z_j \right) = \prod_{j=1}^n \exp(z_j), \quad z_j \in \mathbb{C}, \quad j = 1, \dots, n. \quad (20.7)$$

20.2.2 The Exponential of a Matrix

The formal generalisation of Equation (20.1) to a matrix argument A is given formally by the power series:

$$\exp(A) = e^A = \sum_{n=0}^{\infty} \frac{1}{n!} A^n \quad (20.8)$$

where A is an $n \times n$ real or complex matrix.

This series always converges, and hence the exponential of A is well-defined. In other words, this series is defined in terms of the powers of its matrix argument.

20.3 THE EXPONENTIAL OF A MATRIX: ADVANCED TOPICS

In general, the formula (20.6) does not hold for matrices A and B unless they *commute* (that is, $AB = BA$). If they commute, then we can say that:

$$e^A e^B = e^{A+B}, \quad A, B \in \mathbb{R}^{n,n}. \quad (20.9)$$

However, if A and B do not commute, then (20.9) does not hold, and this is important to know in applications. A number of splitting methods are second-order accurate if the matrices commute, but they are first-order accurate if they do not commute. There are other methods (notably Marchuk's two-cycle method that we discuss in Chapters 22 and 23) that are second-order accurate without needing this commutativity requirement. Nonetheless, we would like to have a way to test whether two matrices commute. In order to determine if two matrices commute, we use the *Golden–Thompson inequality*:

$$\text{tr}(e^{A+B}) \leq \text{tr}(e^A e^B) \quad (20.10)$$

where tr is the *trace functional* defined by:

$$\text{tr}(A) = \sum_{j=1}^n a_{jj}, \quad A = (a_{ij})_{1 \leq i,j \leq n}.$$

If this inequality becomes an equality for matrices A and B , then they commute.

20.3.1 Fundamental Theorem for Linear Systems

Time-dependent PDEs can be converted to a system of ODEs by a process call *semi-discretisation*. For example, we can discretise the Black–Scholes using finite differences, finite volumes or finite elements to produce a system of ODEs that can be used as input to an ODE solver that performs the subsequent integration in time (see Duffy (2018)). To this end, the process leads us to a system of the form:

$$\begin{aligned} \frac{dx}{dt} &= Ax, \quad x(0) = x_0 \\ x_0, x &\in \mathbb{R}^n, \quad A \in \mathbb{R}^{n,n}. \end{aligned} \quad (20.11)$$

Theorem 20.1 (Fundamental Theorem for Linear Systems) Let A be an $n \times n$ matrix. Then for a given $x_0 \in \mathbb{R}^n$ the initial value problem (20.11) has a unique solution $x(t)$ given by:

$$x(t) = e^{At} x_0. \quad (20.12)$$

Lemma 20.1 Let A be a square matrix. Then

$$\frac{d}{dt} e^{At} = A e^{At}. \quad (20.13)$$

Proof.

$$\begin{aligned}\frac{d}{dt}e^{At} &= \lim_{h \rightarrow 0} \frac{e^{A(t+h)} - e^{At}}{h} = \lim_{h \rightarrow 0} e^{At} \frac{(e^{Ah} - I)}{h} \quad (I = \text{identity matrix}) \\ &= e^{At} \lim_{h \rightarrow 0} \lim_{k \rightarrow \infty} \left(A + \frac{A^2 h}{2} + \cdots + \frac{A^k h^{k-1}}{k!} \right) = Ae^{At}.\end{aligned}$$

PROOF OF THEOREM 20.1.

We prove the theorem using the well-known *integrating factor method* for scalar ODEs. We transform the ODE (20.11) in a series of steps as follows (Perko (2001)):

1. $\frac{dx}{dt} = Ax \Rightarrow \frac{dx}{dt} - Ax = 0$
2. $e^{-At} \left(\frac{dx}{dt} - Ax \right) = \frac{d}{dt} (e^{-At}x) = 0$
3. Define $y = e^{-At}x$; Step 2 implies $y = \text{constant} = x_0$
4. Hence $x_0 = e^{-At}x \Rightarrow x = e^{At}x_0$.

20.3.2 An Example

We take a two-dimensional problem for motivation. First, consider the matrix $A = \begin{pmatrix} a & -b \\ b & a \end{pmatrix}$ whose exponential is given by:

$$e^A = \begin{pmatrix} \cos b & -\sin b \\ \sin b & \cos b \end{pmatrix}.$$

You should check this solution based on Definition (20.8).

Now consider the initial value problem:

$$\frac{dx}{dt} = Ax, \quad x(0) = (1, 0)$$

where:

$$A = \begin{pmatrix} -2 & -1 \\ 1 & -2 \end{pmatrix}.$$

It can be checked that the solution is given by:

$$x(t) = e^{At}x_0 = e^{-2t} \begin{pmatrix} \cos t & -\sin t \\ \sin t & \cos t \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = e^{-2t} \begin{pmatrix} \cos t \\ \sin t \end{pmatrix}.$$

You can also experiment with different values of a and b (for example $a = 1$, $b = 0$) or perform some numerical work using explicit and implicit Euler schemes, for example.

We now devote a number of sections to showing how to apply the Method of Lines (MOL). We set up the ODE system, and in practice we can solve the system using the Boost C++ library *odeint* (see Duffy (2018)). The cases that we take in this book are:

- One-dimensional heat equation.
- Plain European options, barrier options (this chapter).
- Spread options (Chapter 23).
- Option sensitivities using CSE (continuous sensitivity equation) method (Chapter 17).

The process to be executed is the same for all initial boundary value problems: ideally, we solve these kinds of problems with Dirichlet numerical boundary conditions on a bounded domain by discretising in space only while ensuring that the boundary conditions are incorporated into the ODE system. The method is easy to apply to multi-dimensional as well as to non-linear problems.

20.4 MOTIVATION: ONE-DIMENSIONAL HEAT EQUATION

We discuss the application of MOL to the initial boundary value problem for the one-dimensional heat equation on the unit interval with zero Dirichlet boundary conditions.

The initial boundary value problem for the one-dimensional heat equation (Tolstov (1962)) describes heat flow in a rod of length L :

$$\begin{cases} \frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2}, & 0 < x < L, \quad t > 0 \\ u(x, 0) = f(x), & 0 \leq x \leq L \\ u(0, t) = A, \quad u(L, t) = B, & t > 0. \end{cases} \quad (20.14)$$

In this case, we can assume without loss of generality that $L = 1$. Here a , A and B are constants.

We can find a solution to the system (20.14) in the case when $A = B = 0$ and $a = 1$ by the method of *separation of variables* (Kreider, Kuller, Ostberg and Perkins (1966)). In this case the *analytical solution* is given by:

$$u(x, t) = \frac{8}{\pi^2} \sum_{n=1}^{\infty} \frac{1}{n^2} \left(\sin \frac{n\pi}{2} \right) (\sin n\pi x) \exp(-n^2\pi^2 t) \quad (20.15)$$

and we use this solution as the benchmark against which numerical schemes can be compared.

As test case we discretise a parabolic PDE in the space direction only (using centred difference schemes, for instance) while keeping the time variable t continuous. We examine the following initial boundary value problem for the one-dimensional heat

equation on the unit interval with zero Dirichlet boundary conditions. It is a special case of (20.14) in fact. It is easy to extend the idea to more general cases. The problem is:

$$\begin{cases} \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, & 0 < x < 1, \quad t > 0 \\ u(0, t) = u(1, t) = 0, & t > 0 \\ u(x, 0) = f(x), & 0 \leq x \leq 1. \end{cases} \quad (20.16)$$

We now partition the space interval $(0, 1)$ into J subintervals and we approximate (20.16) by the *semi-discrete scheme*:

$$\begin{cases} \frac{dU_j}{dt} = h^{-2}(U_{j+1} - 2U_j + U_{j-1}), & 1 \leq j \leq J-1 \\ U_0 = U_J = 0, & t > 0 \\ U_j(0) = f(x_j), & j = 1, \dots, J-1 \end{cases} \quad (20.17)$$

where $h = 1/J$ is the constant mesh size.

We define the following vectors by:

$$\begin{aligned} U(t) &= (U_1(t), \dots, U_{J-1}(t))^T \\ U_0 &= (f(x_1), \dots, f(x_{J-1}))^T. \end{aligned}$$

Then we can rewrite system (20.17) as a system of ordinary differential equations (ODEs):

$$\begin{cases} \frac{dU}{dt} = AU, & t > 0 \\ U(0) = U_0 \end{cases} \quad (20.18)$$

where the matrix A is given by:

$$A = h^{-2} \begin{pmatrix} -2 & 1 & & & 0 \\ 1 & \ddots & \ddots & & \\ 0 & \ddots & \ddots & 1 & \\ & & 1 & -2 & \end{pmatrix}.$$

Where do we go from here? There are a number of questions, for example:

Q1: Does system (20.18) have a unique solution, and what are its *qualitative* properties?

Q2: How accurate is scheme (20.18) as an approximation to the solution of system (20.16)?

Q3: How do we discretise (20.18) in time, and how accurate is the resulting discretisation?

We have discussed questions Q1, Q2 and Q3 in Chapter 15. There are many specific ODE solvers for (20.18), ranging from *one-step methods* to *multi-step methods*

(see Dahlquist and Björck (1974), for example) and from *explicit* to *implicit methods*. We can use other approximate methods such as Runge–Kutta (Stoer and Bulirsch (1980)). In this section we concentrate on one-step explicit and one-step implicit methods to discretise system (20.18) defined by the *Theta method*:

$$\begin{aligned}\frac{U^{n+1} - U^n}{\Delta t} &= \theta A U^{n+1} + (1 - \theta) A U^n, \quad 0 \leq n \leq N - 1, \quad 0 \leq \theta \leq 1 \\ U^0 &= U_0.\end{aligned}\tag{20.19}$$

In this case Δt is the constant mesh size in time.

We rewrite Equation (20.19) in the equivalent form:

$$[I - \Delta t \theta A] U^{n+1} = (I + \Delta t(1 - \theta)) A U^n\tag{20.20}$$

or formally as:

$$U^{n+1} = [I - \Delta t \theta A]^{-1} (I + \Delta t(1 - \theta)) A U^n,\tag{20.21}$$

where I is the identity matrix of size $J - 1$.

Some special cases of θ are:

$$\begin{aligned}\theta &= 1, \text{ implicit Euler scheme} \\ \theta &= 0, \text{ explicit Euler scheme} \\ \theta &= \frac{1}{2}, \text{ Crank Nicolson scheme.}\end{aligned}\tag{20.22}$$

When the schemes are implicit, we can solve the system of equations (20.19) at each time level $n + 1$ using the Double Sweep method or the Thomas algorithm. No matrix inversion is needed in the case of explicit schemes.

The formulation (20.18) is called the *Method of Lines* (MOL), and it corresponds to a semi-discretisation of system (20.16) in the space direction while keeping the time variable continuous. We can write schemes (20.19)–(20.21) in *component form*:

$$\begin{aligned}\frac{U_j^{n+1} - U_j^n}{\Delta t} &= \theta(U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1})/h^2 \\ &\quad + (1 - \theta)(U_{j+1}^n - 2U_j^n + U_{j-1}^n)/h^2, \quad 1 \leq j \leq J - 1\end{aligned}\tag{20.23}$$

or:

$$\begin{aligned}U_j^{n+1} - U_j^n &= \lambda \theta (U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1}) + \lambda (1 - \theta) (U_{j+1}^n - 2U_j^n + U_{j-1}^n) \\ (\lambda &= \Delta t/h^2).\end{aligned}\tag{20.24}$$

Finally:

$$\begin{aligned}-\lambda \theta U_{j+1}^{n+1} (1 + 2\lambda\theta) U_j^{n+1} - \lambda \theta U_{j-1}^{n+1} \\ = \lambda (1 - \theta) U_{j+1}^n + (1 - 2\lambda(1 - \theta)) U_j^n + \lambda (1 - \theta) U_{j-1}^n \quad 1 \leq j \leq J - 1.\end{aligned}\tag{20.25}$$

We see that system (20.25) is tridiagonal, and we can apply the Double Sweep or Thomas algorithms to solve it. In the case of the explicit Euler scheme ($\theta = 0$), these matrix algorithms are not needed, because the solution at time level $n + 1$ can be explicitly computed:

$$U_j^{n+1} = \lambda U_{j+1}^n + (1 - 2\lambda)U_j^n + \lambda U_{j-1}^n, \quad 1 \leq j \leq J - 1. \quad (20.26)$$

We note that the one-factor Black–Scholes PDE can be converted to the heat equation by a change of variables. This is a common trick, but we prefer not to use it. A possible advantage is that it will have good run-time performance. We shall see how to assemble and solve more general problems in later chapters based on the methods that we have developed here.

20.5 SEMI-LINEAR PROBLEMS

We now discuss the abstract *semi-linear* problem:

$$\begin{aligned} \frac{dU}{dt} + A(t, U) &= B(t, U), \quad 0 < t \leq T \\ U(0) &= U_0 \end{aligned} \quad (20.27)$$

where:

$$A(t, \cdot) : D_t \subset H \rightarrow H, \quad t > 0$$

is a strongly dissipative and maximal operator, and H is a real or complex Hilbert space. The operator $B(t, \cdot) : D \rightarrow H$ is a uniformly Lipschitz continuous operator with Lipschitz constant K .

By ‘semi-linear’ in (20.27) we mean that the operator A is linear in u , and the operator B is non-linear in u .

This is a short discussion, and a full treatment is outside the scope of this book (for more information see Hille and Philips (1975)). A special case is the n -factor Black–Scholes equation where the operator A is seen as a mapping from a Hilbert space of functions to itself:

$$A(t, \cdot) : \sum_{i,j=1}^m a_{ij}(x, t) \frac{\partial^2}{\partial x_i \partial x_j} + \cdots + \sum_{i=1}^m b_i(x, t) \frac{\partial u}{\partial x_i} + c(x, t). \quad (20.28)$$

Our aim in this section is to propose some discrete schemes for (20.27) and examine their properties in a Hilbert-space setting. Equation (20.27) can be seen as an ODE in Hilbert or Banach space. Some special cases are:

- Ordinary differential equations
- Partial differential equations
- Integro-differential equations
- Systems of equations.

We shall give some examples of these equations, but first let us examine some schemes for approximating Equation (20.27). The explicit scheme is given by:

$$\frac{U^{n+1} - U^n}{k} + A(t_n, U^n) = B(t_n, U^n) \quad n \geq 0 \quad (20.29)$$

and the fully implicit method is given by:

$$\frac{U^{n+1} - U^n}{k} + A(t_n, U^{n+1}) = B(t_n, U^{n+1}) \quad n \geq 0. \quad (20.30)$$

Finally, the Crank–Nicolson scheme is given by:

$$\begin{aligned} \frac{U^{n+1} - U^n}{k} + A(t_{n+1/2}, U^{n+1/2}) &= B(t_{n+1/2}, U^{n+1/2}) \quad n \geq 0 \\ U^{n+1/2} &\equiv \frac{1}{2}(U^n + U^{n+1}). \end{aligned} \quad (20.31)$$

The advantage of the explicit scheme is that it is easy to program, but it is only conditionally stable. The other two schemes are unconditionally stable, but we must solve a non-linear system at every time level. Can we find a compromise? The answer is yes. When the system (20.27) is *semi-linear* (by which we mean that A is linear in u and B is non-linear in u), a ploy is to apply some kind of implicit scheme with respect to the A part and an explicit scheme with respect to the B part. The result is called the *semi-implicit method*, and one particular case is given by:

$$\frac{U^{n+1} - U^n}{k} + A(t_{n+1}, U^{n+1}) = B(t_n, U^n) \quad n \geq 0. \quad (20.32)$$

We can solve this system using standard matrix solvers at each time level since there are no non-linear terms.

Finally, we can use the predictor-corrector method.

Of course, we wish to know how good the scheme (20.32) is. In general, we should perform a full error analysis, including *consistency*, *stability* and *convergence*. We summarise the main results here. To this end, we write (20.27) in the more general form:

$$\begin{aligned} \frac{dU}{dt} &= f(t, U, U), \quad 0 < t \leq T \\ U(0) &= U_0 \end{aligned} \quad (20.33)$$

where the second parameter corresponds to the derivative terms (the A operator), for example, and the third term might correspond to the zero-order terms (the B operator), for example. In this case we assume that the function $f(t, \cdot, v)$ satisfies a Lipschitz condition with respect to the inner product $\langle \cdot, \cdot \rangle$ in H , that is:

$$\begin{aligned} \forall t \in [0, T], \quad v \in D \\ \langle f(t, u_1, v) - f(t, u_2, v), u_1 - u_2 \rangle \leq K_1 \|u_1 - u_2\|^2 \quad \forall u_1, u_2 \in H \end{aligned} \quad (20.34)$$

where $K_1 \in \mathbb{R}$, $\|\cdot\|$ is the norm in a Hilbert space H and $\langle \cdot, \cdot \rangle$ is the inner product in H .

Condition (20.34) is called the *one-sided Lipschitz condition*. Furthermore:

$f(t, u, \cdot)$ is uniformly Lipschitz continuous in the classical sense with constant:

$$K_2 > 0 \text{ with } \|f(t; u, v_1) - f(t; u, v_2)\| \leq K_2 \|v_1 - v_2\|.$$

A particular case is when:

$$f(t, u, v) = A(t, u) + B(t, v) \quad (20.35)$$

where A is *dissipative* ($K_1 \leq 0$) or *strongly dissipative* ($K_1 < 0$), and $B(t, \cdot)$ is Lipschitz continuous.

The approximate scheme is defined by:

$$\frac{U^{n+1} - U^n}{k} = f(t_{n+1}, U^{n+1}, U^n). \quad (20.36)$$

In general, system (20.27) is easy to implement using MOL software. Finally, we mention that ODE (20.27) can be discretised using the ADE method, as we discussed in Chapter 19.

20.6 TEST CASE: DOUBLE-BARRIER OPTIONS

Barrier options are extensions of standard stock options. The pay-off depends on two market levels, namely a strike price and a *barrier*. The premiums from barrier options are lower than those of standard options in general with the same strike and expiration.

There are several kinds of barrier options (Derman and Kani (1996), Derman and Kani (1997), Haug (2007), Ikeda and Kunimoto (1992)), and for our purposes we scope the problem by considering down-and-out/up-and-out call options. This means that we define a lower barrier and an upper barrier. If the stock value hits either barrier, it becomes worthless. Otherwise, the price is the same as that of a standard option. It is possible to associate *cash rebates* if the asset price crosses the lower and upper barriers. In other words, we are discussing barrier options.

Mathematically, we compute barrier option prices by solving an initial boundary value problem on a bounded domain (thus avoiding domain truncation or domain transformation) with Dirichlet boundary conditions (the rebates will be the boundary values). We discretise the problem in space, using centred differencing to give a system of ODEs, and we note that the boundary conditions have already been incorporated into the system. An optional feature is to use exponential fitting to ensure monotonicity, especially for convection-dominated problems.

We now discuss the steps on how to apply MOL to this problem.

20.6.1 PDE Formulation

Barrier options are options where the payoff depends on whether the underlying asset's price reaches a given level during a certain period of time before expiration. There are two kinds of barriers:

- *In barrier*: This is reached when the asset price S hits the barrier value H before maturity. In other words, if S never hits H before maturity, then the payout is zero.
- *Out barrier*: This is similar to a plain option except that the option is knocked out or becomes worthless if the asset price S hits the barrier H before expiration. This is an option that is knocked out if the underlying asset touches a lower boundary L or upper boundary U prior to expiration.

The above examples were based on constant values for U and L . In other words, we assume that the values of U and L are time-independent. This is a simplification; in general U and L are functions of time, namely $U = U(t)$ and $L = L(t)$. In fact, these functions may even be discontinuous at certain points. For more information, see Haug (2007) and Tavella and Randall (2000).

We concentrate on one-factor barrier options described by the following partial differential equation:

$$-\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + r \frac{\partial V}{\partial S} - rV = 0.$$

In contrast to plain options, we may need to specify two boundary conditions at finite values of S in this case. For a *double-barrier option* two boundaries are specified:

$$\begin{aligned} V(A, t) &= g_0(t), \quad 0 < t < T \\ V(B, t) &= g_1(t), \quad 0 < t < T \end{aligned} \tag{20.37}$$

where g_0 and g_1 are given functions of t .

Here A and B are specified values of the underlying S , and we assume that these barriers are constant. More generally, *time-dependent barriers* $L(t)$ and $U(t)$ are defined in boundary conditions such as:

$$V(L(t), t) = g_0(t), \quad 0 < t < T$$

$$V(U(t), t) = g_1(t), \quad 0 < t < T$$

where $g_0(t)$ and $g_1(t)$ are given functions.

For single barriers (we are only given one barrier), we have to decide on how to define the other barrier. Given a positive single barrier, we then can choose between $S = 0$ or some large value for S , depending on the kind of barrier.

To this end, there are a number of scenarios when working with single-barrier options. For example, we view a single *up-and-out* barrier as a double-barrier option with rebate of value 0 at the down-and-out barrier (that is, when $S = 0$). In this case the company whose stock is being modelled is probably bankrupt and is therefore unable to recover. Another example is a *down-and-out* call option, in which case we need to

truncate the semi-infinite domain. In this case we take the boundary conditions as follows:

$$V(S_{\max}, t) = S_{\max} - Ke^{-r(T-t)} \quad (20.38)$$

where S_{\max} is ‘large enough’.

The payoff function is the initial condition and is given by:

$$V(S, 0) = \max(S - K, 0). \quad (20.39)$$

We shall now examine how to approximate barrier option problems by finite difference methods.

20.6.2 Using Exponential Fitting of Barrier Options

The exponentially fitted schemes were developed by the author specifically for boundary layer problems and time-dependent convection-diffusion equations whose solutions have large gradients in certain regions of the domain of interest. We use *exponential fitting* in the space S direction and implicit Euler time marching in the t direction. If needed, we can employ Richardson extrapolation techniques in the time direction to improve accuracy. For convenience, we write the Black-Scholes equation in the more general and convenient form:

$$LV \equiv -\frac{\partial V}{\partial t} + \sigma(S, t) \frac{\partial^2 V}{\partial S^2} + \mu(S, t) \frac{\partial V}{\partial S} + b(S, t)V \quad (20.40)$$

where:

$$\sigma(S, t) = \frac{1}{2}\sigma^2 S^2, \quad \mu(S, t) = rS, \quad b(S, t) = -r.$$

The corresponding fitted scheme is defined as:

$$L_k^h V_j^n \equiv -\frac{V_j^{n+1} - V_j^n}{k} + \rho_j^{n+1} D_+ D_- V_j^{n+1} + \mu_j^{n+1} D_0 V_j^{n+1} + b_j^{n+1} V_j^{n+1}, \quad 1 \leq j \leq J-1 \quad (20.41)$$

where:

$$\rho_j^n \equiv \frac{\mu_j^n h}{2} \coth \frac{\mu_j^n h}{2\sigma_j^n}.$$

We define the discrete variants of the initial condition and the boundary conditions as follows:

$$V_j^0 = \max(S_j - K, 0), \quad 1 \leq j \leq J-1$$

and:

$$\left. \begin{array}{l} V_0^n = g_0(t_n) \\ V_J^n = g_1(t_n) \end{array} \right\} \quad 0 \leq n \leq N.$$

The system can be cast as a linear matrix system:

$$A^n U^{n+1} = F^n, \quad n \geq 0 \quad (20.42)$$

with U^0 given, and we solve this system using LU decomposition, for example.

20.6.3 Performing MOL with Boost C++ *odeint*

We now discuss how to apply the *odeint* library to price barrier options (see Duffy (2018) for more details on *odeint*). To this end, we define a class *OdeBlackScholes* that mirrors the ODE system of equations and contains the discretised form (in space) of the Black–Scholes PDE. It has a constructor to initialise a mesh array, and in this version the mesh size is constant. (In later versions we can consider non-constant meshes, but doing this at this stage would be confusing.) The C++ class uses *operator overloading* to define the system of equations as well as to define what to do at each time point: for example, checking for early exercise or some other adjustment to the solution:

```
class OdeBlackScholes
{ // Basic ODE class for plain Black Scholes

public:
    state_type mesh;
    double h; // version 1: constant mesh.
public:
OdeBlackScholes(std::size_t NX, double meshSize)

    :mesh(state_type(NX+1)), h(meshSize)
{
    mesh[0] = L;
    for (std::size_t j = 1; j < mesh.size(); ++j)
    {
        mesh[j] = mesh[j - 1] + h;
    }
}

void operator() (const state_type& U, state_type& dUdt,
const value_type t)
{
    double xval, diff,con;

    value_type h2 = 1.0 / (h*h);
    value_type hm1 = 1.0 / (2.0*h);

    // Boundaries (left, interior, right)

    // Left boundary (j = 1)
    std::size_t index = 1;
    xval = mesh[index];
    dUdt[index] = diffusion(xval, t)*h2*(U[index + 1]
        - 2.0*U[index] + bcl(t))
```

```

+ 0.5*convection(xval, t)*(U[index + 1] - bcl(t)) / h
+ reaction(xval, t)*U[index];
//+ penalty(mesh[index], U[index]);

// Right boundary (j = J-1)
index = U.size() - 2;
xval = mesh[index];
dUdt[index] = diffusion(xval, t)*h2*(bcr(t) - 2.0*U[index]
+ U[index - 1])
+ 0.5*convection(xval, t)*(bcr(t) - U[index - 1]) / h
+ reaction(xval, t)*U[index];
//+penalty(mesh[index], U[index]);

// Interior of domain (1 < j < J-1)
for (std::size_t index = 2; index < U.size() - 2; index += 1)
{
    xval = mesh[index];
    diff = diffusion(xval, t)*h2;
    con = convection(xval, t)*hm1;
    dUdt[index] = (diff + con) *U[index + 1]
        + (-2.0*diff + reaction(xval, t))*U[index]
        + (diff - con) *U[index - 1];
    //+ penalty(mesh[index], U[index]);
}

void operator () /*const*/ state_type& U, const value_type t)
{
    // Can modify U at time t (events)
    // constraints, jumps, early exercise, print, save to disk etc.
    // coupon adjustment, signals to other objects etc. etc.
}
};
```

Incidentally, this structure functions as a template for other kinds of problems. We note the presence of free (global) functions for diffusion, convection and reaction terms as well as functions for boundary conditions in the above code, as we shall see. We have also implemented American options by including a penalty term. These are in fact the terms in the Black–Scholes PDE encapsulated as lambda functions for convenience and readability reasons (we do not go overboard with lambda functions):

```

// Functions
auto diffusion = [] (value_type x, value_type t) {      return 0.5*sig*sig*x*x;};
auto convection = [] (value_type x, value_type t) { return (r - d)*x; };
auto reaction = [] (value_type x, value_type t) { return -r; };

// BC
auto bcl = [] (value_type t) // BCL
```

```

{
//  return K*std::exp(-r*t);
//  return RebateL;// K*std::exp(-(r - d)*(t));
};

auto bcr = [] (value_type t) // BCR
{
    return RebateH;
};

auto payoff = [] (value_type x)
{
//  return std::max<value_type>(K - x, 0.0);
    return std::max<value_type>(x-K, 0.0);
};

```

We now discuss how to use Boost C++ *odeint* to discretise the ODE system in time. It supports several well-known solvers (Lambert (1991)) such as *Euler*, *modified midpoint*, *Cash–Karp*, *Runge–Kutta4*, *Bulirsch–Stoer*, *Fehlberg and Rosenbrock*. A detailed discussion of these methods is outside the scope of this book. See Duffy (2018) for a discussion. We first need to define the variables as input to the solver:

```

namespace Bode = boost::numeric::odeint;
/*
using value_type = double;
// The type of container used to hold the state vector
using state_type = std::vector<value_type>;*/

// Input data
const value_type T_0 = 0.0;           // Time 0
const value_type dt = 1.0e-5;         // Step-size in time

const int NX = 64 * (H - L);
double h = (H - L) / static_cast<value_type>(NX);
std::cout << "NX, h: " << NX << ", " << h << '\n';

//
state_type U(NX + 1);    // [0,J]
U[0] = bcl(0.0); U[U.size() - 1] = bcr(0.0); // Compatibility
// Initial condition; discretise IC
value_type x = L + h;
for (std::size_t j = 1; j < U.size() - 1; ++j)
{
    U[j] = payoff(x); x += h;
}

// Integration_class
OdeBlackScholes ode(NX, h);

```

We now choose the Cash-Karp solver for time-integration:

```
Bode::bulirsch_stoer<state_type, value_type> myStepper;
std::size_t steps = Bode::integrate_adaptive
    (myStepper, ode, U, T_0, T, dt, ode);
std::cout << "Steps: " << steps << '\n';

ExcelDriver xl; xl.MakeVisible(true);
xl.CreateChart(ode.mesh, U, "Barrier Option price");
```

As a first *alpha ‘get it working’* test, we take the data that these functions need as follows (taken from Topper (2005)):

```
// Topper page 133, output page 134
// Call up-and-out-down-and-out

value_type K = 100.0;
value_type T = 1.0;
value_type r = 0.1;
value_type d = 0.0;
value_type sig = 0.2;

// Barrier parameters
value_type L = 75.0;
value_type H = 130.0;

// Rebates
value_type RebateL = 0.0;
value_type RebateH = 0.0;
```

We give a list of 3-tuples in which each tuple has elements for stock price, analytical price and MOL price: {76,0.27306,0.27304}, {80,1.22027,1.22012}, {90,2.90287,2.90259}, {100,3.52511,3.52488}, {110,2.89967,2.899614}, {120,1.47489,1.47488}, {129,0.13192,0.131917}.

We see that the analytical and numerical results are in good agreement with each other.

20.6.4 Computing Sensitivities

We now wish to compute the price, delta and gamma of a down-and-out/up-and-out European call option with the following data:

```
value_type K = 100.0;
value_type T = 0.25;
value_type r = 0.1;
value_type d = 0.0;
value_type sig = 0.25;

// Barrier parameters
value_type L = 50.0;
```

```

value_type H = 150.0;

// Rebates
value_type RebateL = 0.0;
value_type RebateH = 0.0;

```

The option price is 6.14448. The down-and-out call is close in value to the standard call because it gets knocked out as the stock moves down to levels where the standard call has little value. The up-and-out call is worth only a fraction of the standard call because it is knocked out for those upward stock moves that contribute most of the value to the standard call. Furthermore, the delta of a barrier option can differ from the delta of the corresponding standard option. For example, the delta of a barrier call can have values greater than one or less than zero; up-and-out deep in-the-money calls whose values vanish at the barrier have a negative delta near the boundary because of the rapid decline in its value there. Barrier option values can also decrease with increasing volatility; for example, an up-and-out call is more likely to get knocked out near the barrier as volatility increases.

We test the robustness and accuracy of our MOL code by computing the option price as well as its delta and gamma. We first compute the barrier option price as a vector at expiration using the ODE solver. We then compute both delta and gamma *using cubic spline methods* (Ahlberg, Nilson and Walsh (1967), Duffy (2018). The graphs produced are similar to Exhibits 8 and 10 of Derman and Kani (1996). See Figures 20.1, 20.2 and 20.3.

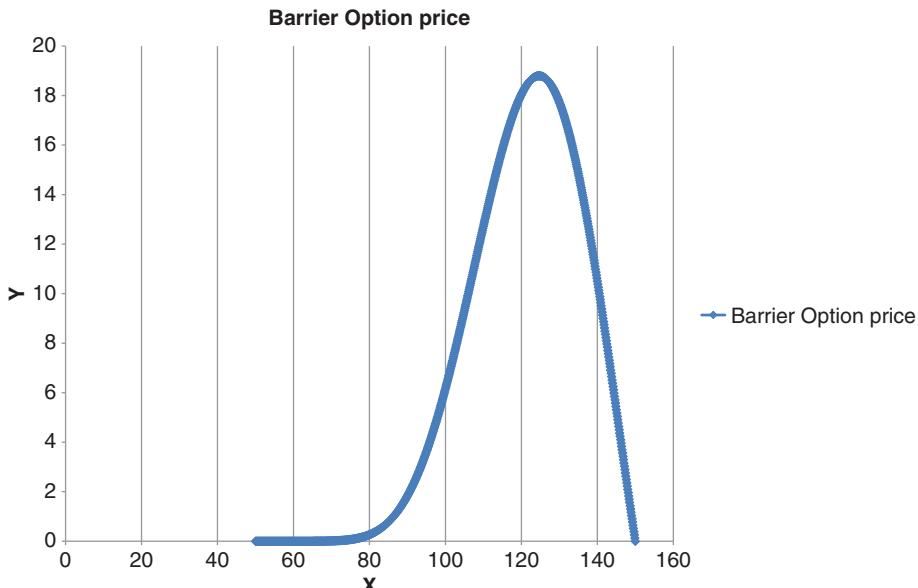


FIGURE 20.1 Price profile.

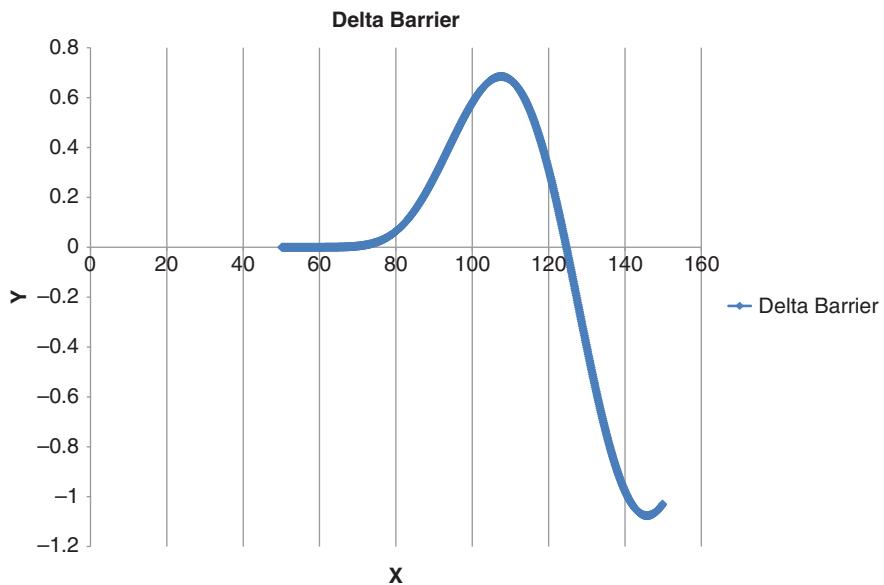


FIGURE 20.2 Delta profile.

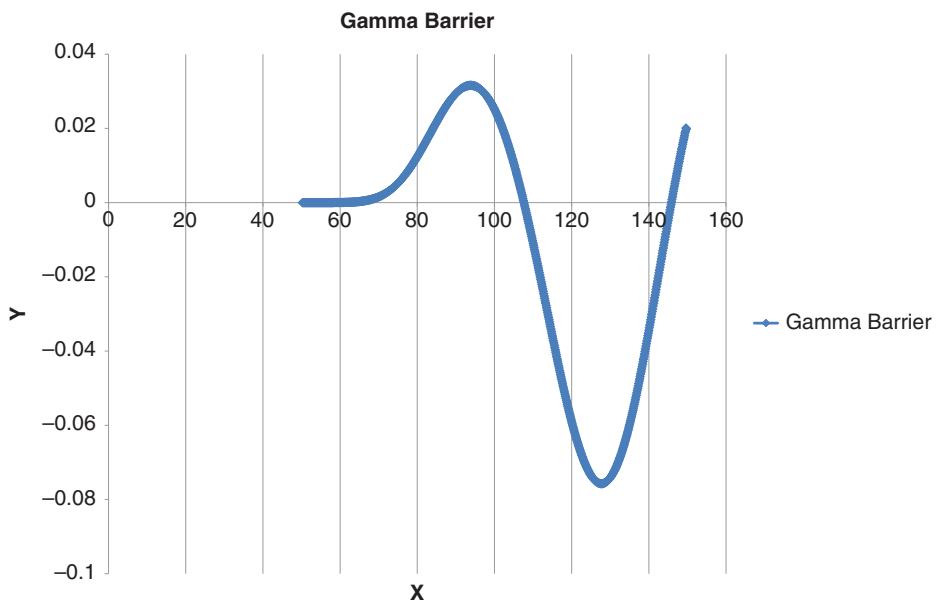


FIGURE 20.3 Gamma profile.

We have already discussed computing option sensitivities using divided differences and by the CSE method in Chapter 16 and in Chapter 17. One final remark on the cubic spline solution that improves the accuracy of the computation of gamma is the following: we use cubic spline interpolation to compute a second-order approximation to delta (Theorem 2.3.4, p. 29 of Ahlberg, Nilson and Walsh (1967)). Having done that, we can get an approximate gamma by taking divided differences of the approximate delta, but this process is unstable and is only first-order accurate at best; furthermore, the resulting curve will not be smooth. An alternative then is to use cubic spline interpolation to compute a second-order approximation to gamma by using the delta values as the underlying input. This produces a smooth second-order accurate approximation to gamma. This is called the ‘*spline on spline*’ variant. See also Hilderbrand (1974), p. 492.

20.6.5 American Options

We have also tested the MOL code to price American options using the data from Section 20.6.4. The answer we got for the double-barrier knock-out call was 6.25388 computed using Courant’s quadratic barrier function and the early exercise constraint trick, by the binomial method. We note that the system of ODEs becomes stiff for large values of the penalty term, and in this case the MOL software performance starts to degrade.

20.7 SUMMARY AND CONCLUSIONS

Some of the advantages of using MOL are:

1. It can be applied to a wide range of scalar (and vector) time-dependent PDEs, containing Dirichlet, Neumann and other kinds of boundary conditions.
2. The resulting system of ODEs is amenable to analysis, including proving existence and uniqueness results and qualitative properties of the ODE system.
3. Highly accurate, professional and robust open-source and commercial ODE solvers are available. There is no need to create your own home-grown solvers. Most developers are not full-time numerical analysts working on the numerical solution of ODEs, and it is better to outsource this part of the project rather than creating your own solvers.
4. MOL can be applied to non-linear PDEs and PDEs with discontinuous initial conditions (pay-off). In particular, special ODE solvers can be used to solve *stiff* ODEs.
5. MOL is a competitor to the *Alternating Direction Implicit* (ADI) and *Fractional Steps* (‘Soviet splitting’) methods in the sense that splitting an n -dimensional PDE into a sequence of one-dimensional PDEs may be redundant. ADI and splitting methods are more difficult to apply than MOL to non-linear and stiff problems.
6. Time-to-market and time-to-develop: it takes less time to write and debug code that uses MOL and ODE solvers than hand-crafted code. The results from MOL code can be used as a ‘second opinion’. In other words, MOL is an effective tool for getting quick results.
7. For PDE novices, it is easier to apply the finite difference method by delegating to ODE solvers rather than trying to hand-craft, test and debug your own (yet another) Crank–Nicolson solver, at least in the short term.

8. We can analyse the mathematical properties of the semi-discretised ODE system as already discussed in this chapter.
9. We do not need to employ a matrix solver.
10. This approach facilitates experimentation with different kinds of boundary conditions and their overall impact on convergence. Examples of choices are: pay-off boundary conditions, PDE boundary conditions, Margrabe boundary conditions, boundary conditions based on the Fichera theory and so on. Once we have determined the best one, we can then use it with confidence in other finite difference schemes. In other words, MOL is a prototyping tool.
11. All finite difference schemes encountered in finance are essential ‘full’ discretisations in time of semi-discrete schemes. And this is good to know.
12. MOL solvers tend to be less (run-time) efficient than hand-crafted solvers, especially for two-factor problems in which the underlying data structures are matrices. In particular, Boost *uBlas* matrices are not very efficient. This feature may make it unsuitable for high-performance PDE solvers.

We mention that MOL in the current context is VMOL or *Vertical MOL* because discretisation takes place in a vertical direction. This is in contrast to HMOL (*Horizontal MOL*, also known as *Rothe's method*), where discretisation takes place in a horizontal direction. A discussion of this method can be found in Ladyženskaja, Solonnikov and Ural'ceva (1988) and Meyer (2015). A discussion of HMOL is outside the scope of this book.

Free and Moving Boundary Value Problems

The mathematician does not blindly confide in the results of the rules which he uses. He knows that faults of calculations are possible and even not infrequent; if the purpose of the calculation is to verify a result which unconscious or subconscious inspiration has foreseen and if this verification fails it is by no means impossible that the calculation be at first false and the inspiration be right.

Jacques Hadamard *The Psychology of Invention in the Mathematical Field* (1945).

21.1 INTRODUCTION AND OBJECTIVES

The boundary value problems that we have discussed in this book until now assumed the space domain was known, either because the original problem was defined on a bounded domain or was defined on an infinite or semi-infinite domain to which we apply domain transformation to map it into a domain that is bounded. In all cases we prescribe boundary conditions of the Dirichlet, Neumann or Robin types. In short, the boundaries are ‘flat’ or constant (independent of space and time). Thus, we exclude non-linear radiation boundary conditions based on the Stefan–Boltzmann law (Chapman (1984)) and differential equations with integral boundary conditions (Jankowski (2002), Friedman (1982)), for example.

We characterise boundaries for the purposes of this book as follows:

- B1: *Known time-dependent boundaries* (with flat boundaries as a special case).
- B2: *Free boundaries* (unknown time-independent boundary).
- B3: *Moving boundaries* (unknown time-dependent boundary).

In this book and chapter we are mainly interested in case B3, with special emphasis on the pricing of options with early exercise feature (American options).

Boundary value problems involving free and moving boundaries are non-linear by definition, because we must solve them by taking into account that the solution and the free (or moving) boundary are both unknown quantities.

This chapter contains new material and advanced mathematics that we have not already discussed in previous chapters. For this reason, we introduce this material by a combination of concrete examples and abstract mathematical formulations. It is a good idea to review Chapters 1 and 2, because there we discuss mathematical analysis skills. In particular, practice, practice, practice (especially calculus and the chain rule for differentiation) until you get the answer!

21.2 BACKGROUND, PROBLEM STATEMENT AND FORMULATIONS

There are many numerical methods for solving free and moving boundary value problems. A definitive work on the subject is Crank (1984) (the same John Crank of Crank–Nicolson fame).

21.3 NOTATION AND PREREQUISITES

Free and moving boundary value problems have their origins in the physical sciences. Problems in which the solution of differential equations must satisfy certain conditions on the boundary of a prescribed domain are called *boundary value problems*. In many cases the boundary of the domain is not known *a priori*, but it must be determined as part of the problem. We partition such problems into two groups; first, the term *free boundary* problem is used when the boundary is stationary and a steady-state solution exists (for example, the solution of an elliptic problem or a perpetual American option). We then have the class of *moving boundary* value problems that are associated with time-dependent problems (for example, defined by a parabolic partial differential equation). The unknown boundary in the latter case is a function of both space and time. In all cases we must specify two conditions on the free or moving boundary. Of course, the usual boundary conditions are specified on the fixed boundary as well as some appropriate initial condition, as already discussed in this book.

In general, we can classify free and moving boundary value problems into different categories depending on the types of problem that they model. For example, a *one-phase* or *single-phase problem* is one where we model a PDE in a single domain with an unknown boundary. The solution on the other side of the unknown boundary is known. With *two-phase* problems, we model different PDEs that are defined in two domains that are separated by a free or moving boundary problems. Most problems in financial engineering at the moment of writing are described as one-phase problems. In this case, the solution is zero on one side of the moving boundary (for example), and it satisfies the Black–Scholes equation on the other side of the boundary, for example.

Moving boundary value problems are sometimes called *Stefan problems* in honour of the Austrian mathematician J. Stefan, who studied the melting of the polar ice cap in 1890.

21.4 SOME INITIAL EXAMPLES OF FREE AND MOVING BOUNDARY VALUE PROBLEMS

We discuss a number of problems to motivate the theory. An excellent and definitive source of these problems is Crank (1984). These problems originate in many application areas, such as:

- Soil mechanics
- Engineering
- Physical and biological sciences
- Metallurgy
- Decision and control theory.

We shall see that the techniques for these problems can be applied to pricing applications with an early exercise feature.

21.4.1 Single-Phase Melting Ice

Consider a semi-infinite sheet of ice. The initial point is at $x = 0$, and we assume that the sheet is initially at the melting temperature, that is zero degrees Celsius. We now raise the temperature of the sheet surface at time $t = 0$, and we maintain the temperature. What we get is the following phenomenon: a boundary surface or interface is born at which melting occurs. This boundary moves from the surface into the sheet and separates a region of water from one of ice at zero degrees. Let us denote the moving boundary by the function $B(t)$.

Let $u(x, t)$ be the temperature at time t and at some point x in the water phase. (The temperature on the other side of the moving boundary is zero.) Then the heat equation is valid in the *liquid region* and is defined by:

$$c\rho \frac{\partial u}{\partial t} = K \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < B(t), \quad t > 0 \quad (21.1)$$

where:

c = specific heat

ρ = density

K = heat conductivity.

We augment this equation, first by a fixed boundary condition:

$$u(0, t) = A, \quad t > 0 \quad (21.2)$$

where the value A is the constant surface temperature, and second by an initial condition:

$$\begin{aligned} u(x, 0) &= 0, \quad 0 < x < \infty, \quad t > 0 \\ B(0) &= 0. \end{aligned} \quad (21.3)$$

Continuing, we need two further conditions on the moving boundary $x = B(t)$, namely:

$$\left. \begin{array}{l} u = 0 \\ -K \frac{\partial u}{\partial x} = L\rho \frac{dB}{dt} \end{array} \right\} t > 0 \quad (21.4)$$

where L = latent heat required to melt ice, and K and ρ are defined in (21.1).

Equation (21.4) is called the *Stefan condition*, and it expresses the heat balance on the moving boundary. It is similar to the *smooth pasting condition* for the pricing of American options. We state that both $u(x, t)$ and $B(t)$ are unknowns in (21.1).

The name ‘one-phase’ should be clear at this stage: we are modelling the temperature in the liquid region by the heat equation, while in the solid region the temperature is identically zero. Thus, we do not need to model the solid region by a PDE; the temperature is always zero.

21.4.2 Oxygen Diffusion

Another example concerns oxygen diffusing into a medium that absorbs and immobilises the oxygen at a constant rate (Crank (1984)). The concentration of the oxygen at the surface of the medium is kept constant. Then a moving boundary marks the innermost limit of oxygen penetration. The surface is then sealed so that no more oxygen penetration takes place. This problem is special because there is a discontinuity in the derivative boundary condition due to the abrupt sealing of the outer surface. If $u(x, t)$ denotes the concentration of oxygen free to diffuse at a distance x from the outer surface at time t , then the partial differential equation (in non-dimensional form) is given by:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad (x, t) \in \Omega_T^+ \quad (21.5)$$

where:

$$\Omega_T^+ = \{(x, t) : 0 < x < B(t), \quad 0 < t < T\}$$

and $B(t)$ is the *moving boundary*; the fixed boundary condition is given by:

$$u = g_0(t) \text{ or } \frac{\partial u}{\partial x} + b(t)u = g_1(t), \quad x = 0 \quad (21.6)$$

while the free boundary condition is given by:

$$\left. \begin{array}{l} u = 0 \\ \frac{dB}{dt} = -\frac{\partial u}{\partial x} \end{array} \right\} x = B(t), \quad 0 < t < T. \quad (21.7)$$

When $t = 0$, the initial condition is given by:

$$\begin{aligned} u(x, 0) &= u_0(x) \\ B(0) &= 0. \end{aligned} \quad (21.8)$$

In general, the problem shown in (21.5)–(21.8) does not have an analytical solution, and numerical methods are needed.

21.4.3 American Option Pricing

We already know that a European option can be exercised only at expiration. American options, on the other hand can be exercised at any time before or up to expiration. In this section we concentrate on a put option with an early exercise feature. Let $P = P(S, t)$ be the put option price. Then P satisfies the PDE:

$$\frac{\partial P}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 P}{\partial S^2} + rS \frac{\partial P}{\partial S} - rP = 0, \quad S > B(t), \quad 0 \leq t \leq T. \quad (21.9)$$

Here $B(t)$ is the moving boundary. We are assuming that no dividends are paid throughout the life of the option. The *terminal condition* is given by:

$$P(S, T) = \max(K - S, 0), \quad S \geq 0, \quad 0 \leq t \leq T \quad (21.10)$$

where K is the strike price.

We now need to prescribe boundary conditions. Since the problem is defined on a region containing both fixed and free boundaries, we define the first fixed boundary condition as:

$$\lim_{S \rightarrow \infty} P(S, t) = 0 \quad (21.11)$$

and the so-called *pasting conditions* at the free boundary as:

$$\begin{aligned} \frac{\partial P}{\partial S}(B(t), t) &= -1 \\ P(B(t), t) &= K - B(t). \end{aligned} \quad (21.12)$$

Furthermore, we define the terminal value for the free boundary as follows:

$$B(T) = K. \quad (21.13)$$

Finally, in front of the free boundary the option price is given by:

$$P(S, t) = \max(K - S, 0), \quad 0 \leq S < B(t). \quad (21.14)$$

The problem shown in (21.9)–(21.14) is similar to the Stefan problem that we studied in Section 21.4.1. It is an example of a single-phase problem.

Since early exercise is permitted, the option price P must satisfy the *constraint*:

$$P(S, t) \geq \max(K - S, 0), \quad S \geq 0, \quad 0 \leq t \leq T. \quad (21.15)$$

As in the previous section, we see that there are two unknowns, namely the option price $P(S, t)$ and the moving boundary $B(t)$. The curve $B(t)$ is called the *optimal exercise boundary*. When $S > B(t)$, we see from Equation (21.9) that P satisfies the Black–Scholes equation, while if $S \leq B(t)$, it is optimal to exercise the put.

21.4.4 Two-Phase Melting Ice

This section can be skipped on a first reading without loss of continuity.

We now revisit the melting-ice problem of Section 21.4.1. In particular, we assume that the ice is initially at a temperature below the melting point, and we assume that heat flows in both the water and ice phases. Then we model a PDE in each phase, (that is, ice and water). The problem is to find a triple:

$$u_1(x, t), \quad u_2(x, t), \quad B(t)$$

where:

(21.16)

u_1 = temperature in the water phase

u_2 = temperature in the ice phase

and where $B(t)$ is the free boundary between the two phases. The heat equation in the two phases in a bounded interval $(0, L)$ is given by:

$$c_j \rho_j \frac{\partial u_j}{\partial t} = K_j \frac{\partial^2 u_j}{\partial x^2}, \quad 0 < x < L$$

where:

(21.17)

c_j = specific heat in phase j

ρ_j = density in phase j

K_j = thermal conductivity in phase j , $j = 1, 2$.

In the interior of the interval $(0, L)$ there is an unknown moving boundary $B(t)$ where the following so-called *Stefan condition* is satisfied:

$$\left. \begin{array}{l} u_1 = u_2 = 0 \\ K_2 \frac{\partial u_2}{\partial x} - K_1 \frac{\partial u_1}{\partial x} = L \rho \frac{dB}{dt} \end{array} \right\} x = B(t). \quad (21.18)$$

We must thus solve two PDEs, one in each domain. The domains are separated by a common free boundary.

21.5 AN INTRODUCTION TO PARABOLIC VARIATIONAL INEQUALITIES

The origins of variational inequalities can be traced back to the 1960s in France and Italy. The work that was done in those early years is making its way into financial engineering applications.

21.5.1 Formulation of Problem: Test Case

In order to motivate variational inequalities, we take a one-dimensional heat equation, and we discretise it using finite difference schemes. This model is useful because we can apply the results to American option pricing problems.

We consider the oxygen diffusion problem again. Recall that this is the problem of oxygen that diffuses into some medium; the medium absorbs and immobilises the oxygen at a constant rate. The concentration of the oxygen at the moving surface remains constant, and we thus conclude that this boundary represents the limit of oxygen penetration. Let us denote this *sealed surface* by $s(t)$. Then the initial boundary value problem in non-dimensional form is given by:

$$\begin{aligned} \frac{\partial c}{\partial t} &= \frac{\partial^2 c}{\partial x^2} - 1, \quad 0 \leq x \leq s(t) \\ \frac{\partial c}{\partial x} &= 0, \quad x = 0, \quad t \geq 0 \quad (\text{fixed boundary condition}) \\ c &= \frac{\partial c}{\partial x} = 0, \quad x = s(t), \quad t \geq 0 \quad (\text{free boundary condition}) \\ c &= \frac{1}{2}(1-x)^2, \quad 0 \leq x \leq 1 \text{ when } t = 0 \quad (\text{initial condition}). \end{aligned} \tag{21.19}$$

This problem is amenable to a variational approach. In this case we get the *differential inequality*:

$$\frac{\partial c}{\partial t} - \frac{\partial^2 c}{\partial x^2} + 1 \geq 0, \quad c \geq 0 \tag{21.20}$$

in conjunction with the equality:

$$\left(\frac{\partial c}{\partial t} - \frac{\partial^2 c}{\partial x^2} + 1 \right) c = 0, \quad 0 \leq x \leq 1. \tag{21.21}$$

This is always zero because the first inequality in (21.20) is zero in $0 < x < s(t)$ and $c \equiv 0$ in the interval $s(t) \leq x \leq 1$.

We now discretise this problem in space and time. In particular, we use centred differencing in space and implicit Euler in time. For the inequality (21.20) we have:

$$\frac{c_j^{n+1} - c_j^n}{k} - \frac{c_{j+1}^{n+1} - 2c_j^{n+1} + c_{j-1}^{n+1}}{h^2} + 1 \geq 0 \quad j = 1, \dots, J-1. \tag{21.22}$$

The Neumann boundary condition at $x = 0$ can be approximated by centred differences with ghost points:

$$\frac{c_{-1}^n - c_1^n}{2h} = 0. \tag{21.23}$$

We can put these discrete equations in the form:

$$\begin{aligned} \text{Find } c \text{ where } c &= (c_1, \dots, c_{J-1})^\top \\ Ac + b &\geq 0 \quad c \geq 0 \\ (Ac + b)c^\top &= 0 \end{aligned} \tag{21.24}$$

where A is a tridiagonal matrix and b is a known vector. This is now a problem in *quadratic programming*.

The Black–Scholes equation can be transformed to the heat equation and then posed in a general *linear complementarity (LCP) form*, as follows:

$$\begin{aligned} \frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} &\geq 0, \quad u - g \geq 0 \\ \left(\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} \right) (u - g) &= 0 \end{aligned} \tag{21.25}$$

where $g = g(x, t)$ is the transformed pay-off. As in the oxygen diffusion problem, we can reduce this problem to the discrete form as in (21.22):

$$\begin{aligned} AU^{n+1} - b^n &\geq 0, \quad U^{n+1} - g^{n+1} \geq 0 \\ (AU^{n+1} - b^n)(U^{n+1} - g^{n+1})^\top &= 0. \end{aligned} \tag{21.26}$$

Here the index n refers to discrete time levels, as in the usual sense in this book.

The next question is to determine how to solve the system (21.26). There are several techniques; one of the original and famous ones is the *Cryer Projected SOR (PSOR) method* (Cryer (1979)). We define new notation as follows:

$$\begin{aligned} z &= Ac + b \\ \text{then} \\ Ac = z - b, c^\top z &= 0 \\ c \geq 0, z \geq 0 \end{aligned} \tag{21.27}$$

and then this problem is equivalent to the minimisation problem:

$$\text{minimise } b^\top c + \frac{1}{2} c^\top A c \text{ for } c \geq 0. \tag{21.28}$$

The Cryer algorithm produces sequences of vectors as follows:

$$\begin{aligned} z_j^{(k+1)} &= b_j + \sum_{i=1}^{j-1} A_{ji} c_i^{(k+1)} - \sum_{i=j}^J A_{ji} c_i^{(k)} \\ c_j^{(k+1)} &= \max \left\{ 0, c_j^{(k)} + \omega z_j^{(k+1)} / A_{jj} \right\} \end{aligned} \tag{21.29}$$

where J is the size of the square matrix A , and ω is the so-called *relaxation parameter*.

Theorem 21.1 (Cryer (1979)) *Let A be positive definite. Then PSOR scheme (21.29) converges for all initial guessed $c^{(0)}$ if and only if $0 < \omega < 2$.*

Caveat: the positive-definiteness of the matrix A is crucial.

The PSOR scheme can be used for schemes that result from a finite element/variational formulation of moving boundary value problems. There are many other schemes, for example the *conjugate gradient method* and *Lagrange method with penalty terms*.

21.5.2 Examples of Initial Boundary Value Problems

Let $\Omega \subset \mathbb{R}^n$ be an open bounded set with smooth boundary Γ ; let the final time $T < \infty$ be given.

Consider the problem of finding $u(x, t)$ such that:

$$\frac{\partial u}{\partial t} - \Delta u = f(x, t) \in \Omega \times (0, T) \quad (21.30)$$

$$u(x, 0) = u_0(x), \quad x \in \Omega \quad (21.31)$$

$$u \geq 0, \quad \frac{\partial u}{\partial \eta} \geq 0, \quad u = \frac{\partial u}{\partial \eta} = 0 \text{ for } (x, t) \in \Gamma \times (0, T)$$

where:

$$\Delta u \equiv \sum_{j=1}^n \frac{\partial^2 u}{\partial x_j^2}. \quad (21.32)$$

Defining $V = H^1(\Omega)$, we seek a solution $u \in V = L^2(0, T; V)$. Assume further that $f(t) \in V^*$ and $u_0 \in H = L^2(\Omega)$.

Let:

$$K \subset V, \quad K = \{v \in V : v(x) \geq 0, \quad x \in \Gamma\}$$

for any $v \in V$, with $v(t) \in K$.

We multiply Equation (21.30) by $v(t) - u(t)$, and integrating over Ω gives the equality:

$$\begin{aligned} & \int_{\Omega} \left\{ \frac{\partial u(t)}{\partial t} - f(t) \right\} (v(t) - u(t)) dx \\ &= \int_{\Omega} \Delta u(t) (v(t) - u(t)) dx, \quad v \in V, \quad v(t) \in K. \end{aligned} \quad (21.33)$$

We now use the divergence theorem (n -dimensional integration by parts), and using the boundary conditions (21.32), we formulate (21.33) as the equivalent form by the application of Green's formula:

$$\begin{aligned} \int_{\Omega} \Delta u(t) (v(t) - u(t)) dt &= \int_{\Gamma} \frac{\partial u(t)}{\partial \eta} (v(t) - u(t)) dt \\ &\quad - \int_{\Omega} \nabla u(t) \cdot \nabla (v(t) - u(t)) dt \end{aligned} \quad (21.34)$$

or:

$$\begin{aligned} & \int_{\Omega} \left\{ \Delta u(t) (v(t) - u(t)) + \nabla u(t) \cdot \nabla (v(t) - u(t)) \right\} dx \\ &= \int_{\Gamma} \frac{\partial u(t)}{\partial \eta} (v(t) - u(t)) dx \geq 0. \end{aligned}$$

From this we deduce the inequality:

$$\int_{\Omega} \Delta u(t)(v(t) - u(t))dt \geq - \int_{\Omega} \nabla u(t) \cdot \nabla(v(t) - u(t))dt. \quad (21.35)$$

Finally, combining (21.33) and (21.34) produces the *parabolic variational inequality*:

$$\begin{aligned} & \int_{\Omega} \left\{ \frac{\partial u(t)}{\partial t}(v(t) - u(t)) + \nabla u(t) \cdot \nabla(v(t) - u(t)) \right\} dx \\ & \geq \int_{\Omega} f(t)(v(t) - u(t))dx \quad \forall v \in K \end{aligned} \quad (21.36)$$

where $\{K = v \in V; v(t) \in K \text{ for a.a. } t \in (0, T)\}$ and where a.a. denotes ‘for almost all’ in the Lebesgue sense.

This is the so-called continuous formulation of the free boundary problem. For motivational purposes, we return to a one-dimensional case of system (21.5), namely the oxygen absorption problem. This is a good example to use as a model. We set $c(x, t) = u(x, t)$ for notational convenience.

We now discuss the variational formulation of the oxygen absorption problem. In this case we start with the system (21.5). This problem is then formulated as the one-dimensional equivalent of (21.36).

The steps that we execute in this section are:

- Formulate the continuous variational inequality.
- Semi-discretisation in x using linear ‘hat’ functions (finite elements) and piecewise polynomials.
- Full-discretisation using implicit Euler or Crank–Nicolson schemes.
- Assembling the set of discrete inequalities.

We multiply both sides of Equation (21.5) by $(v - c)$ where v belongs to the space of test functions:

$$V = \{v : v \in H^1(0, 1), v(1) = 0\}.$$

Then using the equality:

$$\begin{aligned} & \int_0^1 \left(\frac{\partial c}{\partial t} - \frac{\partial^2 c}{\partial x^2} + 1 \right) (v - c) dx \\ &= \int_0^1 \frac{\partial c}{\partial t} (v - c) dx - \left[(v - c) \frac{\partial c}{\partial x} \right]_0^1 + \int_0^1 \frac{\partial c}{\partial x} \frac{\partial}{\partial x} (v - c) dx + \int_0^1 (v - c) dx \end{aligned} \quad (21.37)$$

and the fact that:

$$\begin{cases} v = c = 0 \text{ on } x = 1 \\ \frac{\partial c}{\partial x} = 0 \text{ on } x = 0 \end{cases}$$

we then get the rearranged form of Equation (21.37), namely:

$$\begin{aligned} & \int_0^1 \frac{\partial c}{\partial t} (v - c) dx + \int_0^1 \frac{\partial c}{\partial x} \frac{\partial}{\partial x} (v - c) dx \\ &= \int_0^1 (v - c) dx + \int_0^1 \left(\frac{\partial c}{\partial t} - \frac{\partial^2 c}{\partial x^2} + 1 \right) dx. \end{aligned} \quad (21.38)$$

The final term on the right-hand side in (21.38) is non-negative because of inequality (21.20), hence we get the variational inequality:

$$\begin{aligned} & \int_0^1 \frac{\partial c}{\partial t} (v - c) dx + \int_0^1 \frac{\partial c}{\partial x} \frac{\partial}{\partial x} (v - c) dx \\ &= - \int_0^1 (v - c) dx + \int_0^1 \left(\frac{\partial c}{\partial t} - \frac{\partial^2 c}{\partial x^2} + 1 \right) dx \\ &\geq - \int_0^1 (v - c) dx \end{aligned} \quad (21.39)$$

or in more compact and general form as:

$$\left(\frac{\partial c}{\partial t}, v - c \right) + a(c, v - c) \geq (-1, v - c)$$

where:

$$(f, g) \equiv \int_0^1 f g dx \text{ (inner product)} \quad (21.40)$$

$$a(u, v) \equiv \int_0^1 \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} dx \text{ (bilinear form).}$$

We now find an approximate solution to a slightly more generalised form of (21.40), namely:

$$\left(\frac{\partial u}{\partial t}, v - c \right) + a(u, v - u) \geq (f, v - u). \quad (21.41)$$

As is common in finite element theory, we seek an approximate solution of (21.41) using combinations of linear polynomials with compact support on the interval $(0, 1)$, namely:

$$u = \sum_{j=1}^n u_j \varphi_j, \quad v = \sum_{j=1}^n v_j \varphi_j$$

where the basis ('linear hat') functions f are defined by the formula:

$$\varphi_j(x) = \begin{cases} (x - (j-1)h)/h, & (j-1)h \leq x \leq jh \\ ((j+1)h - x)/h, & jh \leq x \leq (j+1)h. \end{cases}$$

(Note that $\varphi_j(x_j) = 1$, $\varphi_j(x_{j\pm 1}) = 0$).

If we now insert the above expressions for u and v and choosing $v = \varphi_j$ into inequality (21.41), we get the following expression:

$$\begin{aligned} & \int_0^1 \left(\sum_{i=1}^n \frac{\partial u_i}{\partial t} \varphi_i \right) \left(\sum_{j=1}^n (v_j - u_j) \varphi_j \right) dx \\ & + \int_0^1 \left(\sum_{i=1}^n u_i \frac{\partial \varphi_i}{\partial x} \right) \left(\sum_{j=1}^n (v_j - u_j) \frac{\partial \varphi_j}{\partial x} \right) dx \\ & - \int_0^1 f \left(\sum_{j=1}^n (v_j - u_j) \varphi_j \right) dx \geq 0. \end{aligned} \quad (21.42)$$

We now wish to formulate this problem in matrix form, and to this end we define the so-called the *mass matrix* M , *stiffness matrix* K and inhomogeneous terms as follows:

$$\begin{aligned} M_{ij} &\equiv (\varphi_i, \varphi_j) \\ K_{ij} &= a(\varphi_i, \varphi_j), \quad f_j = (f, \varphi_j). \end{aligned}$$

Some arithmetic shows that:

$$\begin{aligned} & \sum_{i=1}^n \frac{\partial u_i}{\partial t} \sum_{j=1}^n (v_j - u_j) \int_0^1 \varphi_i \varphi_j dx \\ & + \sum_{i=1}^n u_i \sum_{j=1}^n (v_j - u_j) \int_0^1 \frac{\partial \varphi_i}{\partial x} \frac{\partial \varphi_j}{\partial x} dx \\ & - \sum_{j=1}^n (v_j - u_j) \int_0^1 f \varphi_j dx \geq 0 \end{aligned} \quad (21.43)$$

or in shorthand notation (neglecting summation signs) we get:

$$M_{ji} \frac{\partial u_i}{\partial t} (v_j - u_j) + K_{ji} u_i (v_j - u_j) - f_j (v_j - u_j) \geq 0. \quad (21.44)$$

This is a semi-discrete scheme; in other words, the x variable has been discretised while the t variable is continuous. In order to carry out the last step, namely full

discretisation, we replace the t -derivative in (21.44) by a divided difference. In this case we employ an implicit Euler scheme, as follows:

$$M_{ji} \frac{u_i^{n+1} - u_i^n}{k} (v_j - u_j^{n+1}) + K_{ji} u_i^{n+1} (v_j - u_j^{n+1}) - f_j (v_j - u_j^{n+1}) \geq 0 \quad (21.45)$$

or:

$$[M_{ji}/k + K_{ji}] u_i^{n+1} (v_j - u_j^{n+1}) \geq [f + M_{ji}/k] (v_j - u_j^{n+1}). \quad (21.46)$$

This inequality is in the same form as (21.24) and can be solved by the Cryer algorithm, for example. We can carry out the same analysis for convection-diffusion problems. The mathematics become more tedious. We remark that it takes time to learn how to apply the above schemes to practical problems.

21.6 AN INTRODUCTION TO FRONT-FIXING

Free boundary value problems are special because we have to find the solution of a partial differential equation that satisfies auxiliary initial conditions and boundary conditions on a fixed boundary as well as on a free boundary. We now discuss *front fixing*, and in this case we track the free or moving surface by a suitable change of variables. We then use partial differentiation to produce a non-linear partial differential equation on a fixed domain. In the examples in this chapter we have a free boundary somewhere in the interior of the domain of interest. In this case we look specifically at the transformation that was suggested in Landau (1950):

$$x = S/B(t) \quad (21.47)$$

where, for the Black–Scholes equation, S is the underlying and $B(t)$ is the early exercise boundary. We transform the (linear) Black–Scholes equation in the independent variables (S, t) to a non-linear PDE in the new independent variables (x, t) . In order to effect the transformation, we must use partial derivatives and the chain rule as introduced in Chapter 1.

21.6.1 Front-Fixing for the Heat Equation

As a first example, again we examine the one-dimensional Stefan problem (Crank (1984)):

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < s(t), \quad t > 0 \\ u &= 1, \quad x = 0, \quad t > 0 \\ u &= 0, \quad x > 0, \quad t = 0. \\ s(0) &= 0 \\ \left. \begin{aligned} u &= 0 \\ -\frac{\partial u}{\partial x} &= \lambda \frac{ds}{dt} \end{aligned} \right\} &x = s(t), \quad t > 0 \end{aligned} \quad (21.48)$$

where λ = latent heat.

We apply the *Landau transformation* where $s = s(t)$ is the moving boundary:

$$\xi = x/s(t). \quad (21.49)$$

Using the rules for partial differentiation, we can calculate the derivatives in the new variables using the chain rule in two variables for differentiation (set $\tau \equiv t$ identically just to be complete):

$$\begin{aligned} \frac{\partial u}{\partial x} &= \frac{\partial u}{\partial \xi} \frac{\partial \xi}{\partial x} + \frac{\partial u}{\partial \tau} \frac{\partial \tau}{\partial x} = \frac{1}{s(t)} \frac{\partial u}{\partial \xi} + 0 \frac{\partial u}{\partial t} = \frac{\partial u}{\partial \xi} \frac{\partial \xi}{\partial t} + \frac{\partial u}{\partial \tau} \frac{\partial \tau}{\partial t} \\ &= \frac{\partial u}{\partial \xi} \frac{\partial \xi}{\partial s} \frac{ds}{dt} + \frac{\partial u}{\partial \tau} = \frac{\partial u}{\partial \xi} \frac{\partial \xi}{\partial s} \frac{ds}{dt} + \frac{\partial u}{\partial t} \end{aligned} \quad (21.50)$$

or:

$$\begin{aligned} \frac{\partial u}{\partial x} &= \frac{1}{s(t)} \frac{\partial u}{\partial \xi}, \quad \frac{\partial^2 u}{\partial x^2} = \frac{1}{s(t)^2} \frac{\partial^2 u}{\partial \xi^2} \\ \left(\frac{\partial u}{\partial t} \right)_x &= \frac{\partial u}{\partial \xi} \frac{\partial \xi}{\partial t} + \left(\frac{\partial u}{\partial t} \right)_\xi = \frac{-x}{s(t)^2} \frac{ds}{dt} \frac{\partial u}{\partial \xi} + \left(\frac{\partial u}{\partial t} \right)_\xi. \end{aligned} \quad (21.51)$$

We now get a convection-diffusion equation in a fixed domain:

$$\frac{\partial^2 u}{\partial \xi^2} = s^2 \frac{\partial u}{\partial t} - s \xi \frac{ds}{dt} \frac{\partial u}{\partial \xi}, \quad 0 < \xi < 1, \quad t > 0. \quad (21.52)$$

We have thus transformed a diffusion equation with a moving boundary into a non-linear convection-diffusion equation on a fixed domain $(0, 1)$.

Furthermore, the conditions on the free boundary are given by:

$$-\frac{1}{s} \frac{\partial u}{\partial \xi} = \frac{ds}{dt}, \quad \xi = 1, \quad t > 0. \quad (21.53)$$

Looking at this equation, we conclude that we now have a PDE on a fixed interval (no free boundary), but now there are two unknowns, namely the temperature u and the free surface $s = s(t)$. Thus, we have simplified the problem in one direction, but it has become more complex in the other direction! There is no analytic solution for this problem, and we must use numerical methods. The same analysis can be used to price American options.

21.7 PYTHON CODE EXAMPLE: ADE FOR AMERICAN OPTION PRICING

In this section we take a hands-on approach to price an American option by the PDE approach. To this end, we apply the ADE method to the standard Black–Scholes

equation as an illustration. (We could have taken the Crank–Nicolson finite difference method or the binomial method, but they lead to the same conclusions that we arrive at when using ADE.) The method that we discuss is easy to implement, and it basically consists of an extra test in the code to price a European put option. It was originated in Brennan and Schwartz (1977). Taking the case of a put option, the tactic is to first compute the European price V_j^{n+1} , $n \geq 0, j = 1, \dots, J$ for the new time level and for all mesh points corresponding to the discrete values of the underlying stock. Then we make sure that the *following constraint* is satisfied:

$$V_j^{n+1} = \max(V_j^{n+1}, \max(K - S_j, 0)). \quad (21.54)$$

That's it! The code that implements this constraint can be inserted in any numerical method that implements this *early exercise feature*. The method can also be used when pricing two-factor options using splitting or ADI methods; after having computed the European price V_{ij}^{n+1} at each new time level and for all mesh points, we then modify the value against the constraint as it were; for example, here is a preview in the case of a spread option (see Chapter 23):

$$V_{ij}^{n+1} = \max(V_{ij}^{n+1}, g(S_1^i, S_2^j)) \quad 1 \leq i \leq J_1, \quad 1 \leq j \leq J_2$$

where the pay-off function g for a put spread is defined as:

$$g(S_1, S_2) = \max(0, K - (S_1 - S_2)). \quad (21.55)$$

We see that this approach is much easier to program than front-fixing or penalty methods. However, the constraint checker is first-order accurate, and its use introduces a possible performance impact in the body of single or double loops.

We now discuss how to price a one-factor American option in Python using the ADE method. We use the Barakat and Clark scheme for diffusion and the Towler–Yang scheme for convection (an interesting exercise is to use the Roberts–Weiss scheme for convection instead). We adopt a simple modular approach, and we avoid making the code object-oriented at this stage; the goal is to show how the algorithm functions. There is all the time in the world to make the code more reusable.

The main blocks of code are:

1. Option data and PDE data.
2. The pay-off function and boundary conditions (we use domain truncation).
3. The PDE object as a group of functions.
4. The American option constraint.
5. Create mesh and initialisation of arrays and numeric data.
6. Implement the ADE scheme and produce array of option prices.
7. Display the results.

We show the simple Python code corresponding to these seven blocks:

```
import numpy as np

# Option Data
# European put = 5.84628, call = 2.13337
K = 65
T = 0.25
r = 0.08
d = 0.0
sig = 0.3

# Truncated space domain
A = 0.0; B = 4*K

# Initial and boundary conditions
def InitialCondition(x):

    return np.maximum(K-x,0.0)

def bcl(t): #Dirichlet boundary condition at x = A

    return K*np.exp(-(r-d)*t)

def bcr(t): #Dirichlet boundary condition at x = B

    return 0.0

# Functions (coefficients of BS PDE)
def diffusion(x, t):

    return 0.5*sig*sig*x*x

def convection(x, t):

    return (r - d)*x

def reaction(x, t):

    return -r

def rhs(x, t):

    return 0.0

# The early exercise constraint
def AmericanAdjuster (V, S):
    # e.g. early exercise
```

```
# V = V # European case
V = max(V, InitialCondition(S))
return V

import numpy as np
import copy
import matplotlib.pyplot as plt
import BlackScholesPde as pde

def CreateMesh(n, a, b):
# Create a mesh of size n+1 on the closed interval [a,b]

    #result has indexes 0..n
    result = np.linspace(a,b,n+1) # n subintervals on [0,T]
    result[0] = a

    h = (b - a) / n
    for j in range(1, n+1): # 1..n
        result[j] = result[j - 1] + h;

    return result;

##### Alternating Direction Explicit (ADE) Method #####
# See Duffy C++ 2018 book
print("ADE")

# Parameters for FD scheme
a = pde.A; b = pde.B
m = 8
n = m*pde.B
h = (b-a) / n
h2 = h*h;
xarr = CreateMesh(n,a,b)

#ADE is unconditionally stable and accuracy O(h^2 + dt^2)
k = pde.T/601

sz = n+1
U = np.linspace(a,b,sz) # upsweep
V = np.linspace(a,b,sz) # downsweep
VNew =np.linspace(a,b,sz) # (U+V)/2

#initialise the arrays from the pde's initial condition
for j in range(0,sz): #0..sz-1
    U[j] = V[j] = pde.InitialCondition(xarr[j])
```

```

tnow = k

while tnow <= pde.T:

    # Dirichlet boundary conditions
    VNew[0] = U[0] = V[0] = pde.bcl(tnow)
    VNew[sz-1] = U[sz-1] = V[sz-1] = pde.bcr(tnow)

    # Up Sweep, left to right
    for j in range(1,sz-1): #1..n-1
        # Towler-Yang
        t1 = k* pde.diffusion(xarr[j], tnow) / h2
        t2 = (0.5 * k * (pde.convection(xarr[j], tnow))) / h
        t3 = ( 1.0 + t1 - pde.reaction(xarr[j],tnow) * k )
        t4 = -k *pde.rhs(xarr[j],tnow)
        U[j] = ((t1 - t2)*U[j-1] + (1.0 - t1)*U[j]
                 + (t1 + t2)*U[j+1] + t4) / t3

        # Early exercise
        U[j] = pde.AmericanAdjuster (U[j], xarr[j])

    # Down Sweep, right to left
    for j in range(sz-2, 0, -1):      # Down Sweep
        #Towler-Yang
        t1 = k* pde.diffusion(xarr[j], tnow) / h2
        t2 = (0.5 * k * (pde.convection(xarr[j], tnow))) / h
        t3 = ( 1.0 + t1 - pde.reaction(xarr[j],tnow) * k )
        t4 = -k * pde.rhs(xarr[j],tnow)
        V[j] = ((t1 - t2)*V[j-1] + (1.0 - t1)*V[j]
                 + (t1 + t2)*V[j+1] + t4) / t3
        # Early exercises
        V[j] = pde.AmericanAdjuster (V[j], xarr[j])

    for j in range(1,sz-1): # Barakat and Clark averaging
        VNew[j] = 0.5*(U[j] + V[j])
    tnow += k #next time level

print ("finished computing")
print(VNew)

# plot results
plt.plot(xarr,VNew, 'b--')
plt.xlabel('S')
plt.ylabel('Option Price')
plt.show()

```

This code can be used as a basis upon which to implement the other ADE building block schemes from Chapter 19.

In Duffy (2018) we discuss the pricing of American options and their implementation in C++ using lattice and finite difference methods with penalty terms. We also produced C++ code to price perpetual American options.

21.8 SUMMARY AND CONCLUSIONS

In this chapter we gave an introduction to free and moving boundary value problems. These are non-linear problems in which one of the boundaries is unknown. This is a huge area of research, and many mathematical and numerical techniques have been invented to solve these kinds of problems. A comprehensive overview is given in Crank (1984). Popular PDE-based methods for American option pricing are based on front-fixing, penalty methods and the Brennan–Schwartz method.

CHAPTER 22

Splitting Methods, Part II

This is Assembly Control calling all Zero X units. Assembly Phase One – Thunderbirds are go!

22.1 INTRODUCTION AND OBJECTIVES

The goal of this chapter is to document a *defined process* to analyse, design and implement the finite difference method for a generic class of two-factor PDEs that are commonly encountered in finance. In general, we provide answers to the following questions as they are recurring themes in this book:

- A1: PDEs in conservative or non-conservative form?
- A2: Do we solve the PDE in a truncated or transformed domain?
- A3: Do we transform the PDE in some way in order to remove the mixed derivative term?
- A4: Discovery and imposition of (numerical) boundary conditions.
- A5: Which splitting scheme to use (there are several to choose from, depending on the desired accuracy)?
- A6: Monotonicity and exponentially fitted schemes.
- A7: Which tridiagonal matrix solver to use (Thomas, Double Sweep)?
- A8: Special (and pesky) issues: for example, convection-dominance, spurious reflections at downstream boundaries, discontinuous initial condition (pay-off).
- A9: Are we modelling partial integro-differential equations (PIDE)?
- A10: Software framework: Are we creating a software prototype (proof-of-concept) or production code?
- A11: Code debugging and acceptance testing.

We have already discussed most of the prerequisite theory in previous chapters to help us understand and implement the action points A1 to A11. In particular, reading and understanding the hands-on topics in Chapter 18 will help us prepare for the splitting methods that we present in the current chapter.

We shall see how to apply action points A1 to A11 in Chapter 23 when we discuss spread options. This chapter lays the mathematical foundations for splitting methods.

22.2 BACKGROUND AND PROBLEM STATEMENT: THE ESSENCE OF SEQUENTIAL SPLITTING

In this chapter we continue with the analysis of splitting methods that we introduced in Chapter 18. In the current chapter we examine the full lifecycle as it were, from a formulation of the initial boundary value problem to showing how to approximate it using various splitting methods. We focus on two-factor convection-diffusion-reaction equations containing a mixed derivative term. The objective is to approximate these equations by major splitting schemes:

- S1: Basic two-leg scheme. First-order accurate, at least.
- S2: Predictor-corrector scheme. Second-order accurate.
- S3: Marchuk Two-Cycle method. Second-order accurate.
- S4: Strang operator splitting scheme. Second-order accurate.
- S5: Lie-Trotter scheme. First-order accurate.

All of these schemes can and have been applied to a range of PDEs in finance, for example, Davidson and Levin (2014), Sheppard (2007), Wernick (2015) and Chapter 23 of this book, for example. We analyse the schemes S1 to S5 from both a mathematical and numerical viewpoint, and this will allow us to gain more insights into the relative merits of each one.

In this chapter we also discuss how to design the algorithms corresponding to schemes S1 to S5. Since it takes a lot of effort to design and implement algorithms, we should try to write the software in such a way that it can be reused and extended to different kinds of derivatives pricing problems without succumbing to the evil ‘copy, paste and modify’ trick. To be more precise, we would like to design and implement software systems by combining software components into a working computer program. Each component implements an algorithm or part of an algorithm. This is an area of computational finance that has received little attention in the standard literature, possibly because most authors are possibly not (necessarily) professional software developers and may not consider programming to be important enough to warrant a discussion.

Some recent developments in splitting methods are discussed in Geiser (2014).

22.3 NOTATION AND MATHEMATICAL FORMULATION

In this section we give some mathematical background to splitting methods. Much of the literature on these methods tends to focus immediately on the discretised set of equations, while it is important to realise that it is possible to analyse PDE problems from a more abstract viewpoint.

22.3.1 C_0 Semigroups

A *Banach space* B is a normed linear space which is complete under the metric associated with the norm. In other words, each Cauchy sequence in B converges in norm to an element of B (Rynne and Youngson (2000), Kolmogorov and Fomin (1961)).

A C_0 semi-group (also known as a *strongly continuous one-parameter semi-group*) is a generalisation of the exponential function. We have already seen some examples in Chapter 20, in particular the Cauchy problem for a system of ODEs and its solution as a matrix exponential. In this case the symbol is a square matrix. Strongly continuous semigroups provide solutions of linear constant coefficient ODEs in Banach spaces that arise from the study of PDEs, for example. In this case the symbol A becomes a partial differential operator.

Definition 22.1 Define $\mathbb{R}^+ = \{x \in \mathbb{R}; x \geq 0\}$, X a Banach space and $L(X)$ the set of linear operators on X . Then a *strongly continuous semi-group* is a map $T : \mathbb{R}^+ \rightarrow L(X)$ such that:

1. $T(0) = I$ (identity operator on X).
2. $T(t+s) = T(t)T(s) \quad \forall t, s \geq 0$.
3. $\|T(t)x_0 - x_0\| \rightarrow 0$ as $t \downarrow 0 \quad \forall x_0 \in X$.

where $\|\cdot\|$ is the norm in X . The first two axioms are algebraic while the third axiom is topological, and it states that T is continuous in the *strong operator topology* (Hille and Philips (1975)).

We can see that the standard exponential function $T(t) = e^t$, $t \geq 0$ is a simple example of a strongly continuous semi-group.

22.3.2 Abstract Cauchy Problem

In this section we give sufficient conditions under which an ODE in a Banach space has a unique classical solution. The results subsume splitting methods as a special case, and in this way, we hope to put the mathematical theory on a firm footing. To this end, consider the *Cauchy problem*:

$$\begin{aligned} u' &\equiv \frac{du}{dt} = Au + f, \quad 0 < t \leq T, \quad u(0) = u_0 \text{ where } f \in C([0, T]; X), \\ u_0 &\in X. \end{aligned} \tag{22.1}$$

If $A \in L(X)$, then Equation (22.1) admits a unique solution in $C^1(\mathbb{R}; X)$ given by:

$$u(t) = e^{tA}u_0 + \int_0^t e^{(t-s)A}f(s)ds.$$

We can also call (22.1) an *initial value problem* for an ODE. On the other hand, a *weak (mild) solution* of (22.1) is a continuous function $u(t)$ satisfying:

$$\int_0^t u(s)ds \in D(A) \text{ and } A \int_0^t u(s)ds = u(t) - u_0 \tag{22.2}$$

where $D(A)$ is the domain of A . The operator A is closed, but it is not necessarily bounded and $D(A)$ is dense in X .

Any classical solution of (22.1) is also a weak solution of (22.1). A weak solution is a classical solution if and only if it is continuously differentiable.

Finally, the *infinitesimal generator* A of a strongly continuous semi-group T is defined by:

$$Ax = \lim_{t \downarrow 0} \frac{1}{t}(T(t) - I)x$$

whenever this limit exists.

22.3.3 Examples

We give some examples of semigroups related to PDEs. The first example is the Cauchy problem for the heat equation in n -dimensional space:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \Delta u \text{ in } \mathbb{R}^n \times (0, \infty) \\ u &= f \text{ on } \mathbb{R}^n \times \{t = 0\} \\ \text{where } \Delta u &= \sum_{j=1}^n \frac{\partial^2 u}{\partial x_j^2}. \end{aligned} \tag{22.3}$$

The solution of (22.3) is given by:

$$\begin{cases} S(t)f(x) = \frac{1}{(4\pi t)^{n/2}} \int_{\mathbb{R}^n} \exp\left(\frac{-\|x-y\|^2}{4t}\right) f(y) dy, & t > 0 \\ S(0) = I. \end{cases} \tag{22.4}$$

which is also known as the *heat semi-group*. The *solution/evolution operator* $S(t)$ evolves the heat distribution f at time 0 to time t . In other words, evolving the solution from time 0 to time t and then from time t to time $t + t'$ is the same as evolving the solution from time 0 to time $t + t'$:

$$S(t') \cdot S(t) = S(t' + t).$$

If the system is time-dependent ($A = A(t)$ in (22.1)), then the evolution operator depends on a start time and end time, and we then get a somewhat more complex *exponential property*:

$$S(t_3, t_2)S(t_2, t_1) = S(t_3, t_1).$$

For example, $S(t_1, t_2) = e^{t_2 - t_1}$ satisfies this relationship.

The motivation for C_0 semi-group theory is to exploit this exponential property in order to write the solution operator as some kind of actual exponential for an operator A .

The second example is to examine the one-dimensional heat equation on the unit interval with Dirichlet boundary conditions using semi-group theory, and in particular we regard this PDE as an ODE in function space:

$$\begin{cases} \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, & 0 < x < 1, t > 0 \\ u(0, t) = u(1, t) = 0, & t > 0 \\ u(x, 0) = f(x), & 0 < x < 1. \end{cases} \tag{22.5}$$

Let $X = L^2((0, 1); \mathbb{R})$ be the order L^2 space of square integrable real-valued functions with the interval $(0, 1)$ as domain. Let A be the second-derivative operator in (22.5) with domain $D(A) = \{u \in H^2((0, 1); \mathbb{R}); u(0) = u(1) = 0\}$, where the Sobolev space $H^2((0, 1); \mathbb{R})$ is the completion of the space $\{f \in C^2(0, 1) : \|f\|_2 < \infty\}$ where:

$$\|f\|_2 = \left(\sum_{\alpha=0}^2 \left| \frac{d^\alpha f}{dx^\alpha} \right|^2 \right)^{1/2}.$$

See Adams (1975) for an introduction to Sobolev spaces.

22.4 MATHEMATICAL FOUNDATIONS OF SPLITTING METHODS

This section introduces several mathematical and numerical results that serve as a foundation for understanding the algorithmic and implementation details of splitting methods. In particular, we focus on matrix splitting methods, why splitting is important and the consequences of splitting.

22.4.1 Lie (Trotter) Product Formula

This formula allows us to approximate the exponential of a sum $A + B$ of two matrices A and B by a product of uncoupled terms involving metrics A and B :

$$e^{A+B} = \lim_{n \rightarrow \infty} (e^{A/n} e^{B/n})^n.$$

This formula has applications in the path integral formulation of quantum mechanics and in the construction of splitting methods for the numerical solution of differential equations. It is also used when proving the Feynman–Kac formula that we introduced in Chapter 13.

We note that this formula holds even when A and B do not commute.

22.4.2 Splitting Error

We start with the finite-dimensional vector initial value problem:

$$\begin{cases} \frac{du}{dt} = (A_1 + A_2)u, & 0 < t \leq T \\ u(0) = u_0 \end{cases} \quad (22.6)$$

with solution:

$$u(t) = e^{(A_1+A_2)t} u_0, \quad A_1, A_2 \in \mathbb{R}^{n,n}. \quad (22.7)$$

Let us take the (possibly) naïve approach of breaking (22.7) into a sequence of two simpler initial value problems:

$$\begin{cases} \frac{dv}{dt} = A_1 v, & 0 < t \leq \tau \\ v(0) = u_0 \\ \frac{dw}{dt} = A_2 w, & 0 < t \leq \tau \\ w(0) = v(\tau). \end{cases} \quad (22.8)$$

We can see that the solution of (22.8) using the *semi-group property* is given by:

$$w(\tau) = e^{A_2 \tau} v(\tau) = e^{A_2 \tau} e^{\alpha_1 \tau} u_0. \quad (22.9)$$

while the analytical solution is given by:

$$u(\tau) = e^{\tau(A_1 + A_2)} u_0. \quad (22.10)$$

In general, the discrepancy between (22.9) and (22.10) leads to a so-called *local splitting error* defined by:

$$\text{Error } (\tau) = \exp(\tau(A_1 + A_2)) - \exp(\tau A_2) \exp(\tau A_1) u_0. \quad (22.11)$$

We can see by Taylor expanding these exponential terms that:

$$\exp(\tau(A_1 + A_2)) - \exp(\tau A_2) \exp(\tau A_1) = (A_1 A_2 - A_2 A_1)\tau^2 + O(\tau^3). \quad (22.12)$$

We see that the second-order term in τ is zero only if the matrices A and B commute. Two other splitting schemes are:

- *Strang splitting*:

$$C(\tau) = \exp\left(\frac{\tau}{2} A_1\right) \exp(\tau A_2) \exp\left(\frac{\tau}{2} A_1\right). \quad (22.13)$$

- *Symmetrically weighted sequential splitting (SWSS)*:

$$C(\tau) = \frac{1}{2} [\exp(\tau A_2) \exp(\tau A_1) + \exp(\tau A_1) \exp(\tau A_2)]. \quad (22.14)$$

In these cases $C(\tau)$ is the approximate solution to IVP (22.6).

Definition 22.2 A splitting scheme is *order p accurate* if the following error holds for the local splitting error:

$$\| \text{Error } (\tau) \| = O(\tau^{p+1}). \quad (22.15)$$

By Taylor expansions we can prove that (22.9) is first-order accurate and that (22.13) and (22.14) are both second-order accurate.

22.4.3 Component Splitting and Operator Splitting

There is no unique way to split a PDE or matrix into simpler subproblems. There can be several criteria for choosing a suitable *splitting strategy*. Some initial examples are:

- S1: Split an evolution operator into its diffusion and convection components.
- S2: Split an evolution operator into operators based on spatial dimensions (*dimensional splitting*).
- S3: We can discretise a PDE using the finite difference method and then perform *matrix splitting* on the resulting discrete system. This is a common approach with time-independent problems (Varga (1962), Hageman and Young (1981)).
- S4: Separate the Schrödinger evolution operator into alternating increments of kinetic and potential operators.
- S5: Splitting for *multiscale problems*: decompose behaviour into its microscopic and macroscopic aspects.
- S6: *Stochastic splitting schemes*: decouple the deterministic and stochastic terms in an SDE model.

We thus see that splitting is a universal concept and that it is widely used in computational fluid dynamics (CFD) (exponential splitting schemes), rigid body dynamics particle tracking with collision, to name a few applications (for a survey, see Geiser (2014)). In computational finance, strategy S2 (and to a lesser extent S3) seems to be the most widespread strategy.

22.4.4 Splitting as a Discretisation Method

Until now in this chapter we assumed that time was a continuous variable. We now subdivide the interval $[0, T]$ into N equal subintervals $[(n - 1)\Delta t, n\Delta t], n = 1, \dots, N$, $\Delta t = \frac{T}{N}$. The splitting solution is defined at the mesh points $n\Delta t, n = 0, \dots, N$, and the splitting process can be written as:

$$\begin{aligned} u^{n+1} &= C(\Delta t)u^n \\ u^0 &= u_0. \end{aligned} \tag{22.16}$$

The matrix operator $C(\Delta t)$ represents an operator (matrix) that corresponds to the applied discretisation. For example, *sequential first-order splitting* (22.9) corresponds to:

$$C(\Delta t) = \exp(\Delta t A_2) \exp(\Delta t A_1)$$

while Strang splitting and SWS splitting correspond to operators (22.13) and (22.14), respectively.

Definition 22.3 A splitting process that approximates (22.6) is said to be *consistent* if for its solution $C(\Delta t)$ the relation:

$$\lim_{\Delta t \rightarrow 0} \left\| \left(\frac{C(\Delta t) - I}{\Delta t} - (A_1 + A_2) \right) u(t) \right\| = 0 \quad \forall t \in [0, T] \quad (22.17)$$

holds and where $\|\cdot\|$ is the matrix norm.

By the *Lax Equivalence Theorem* (consistency and stability imply convergence), we need to prove that splitting schemes are stable.

Definition 22.4 A splitting scheme with operator $C(\Delta t)$ is *stable* if there exists a constant $K > 0$ such that:

$$\|C(t)^n\| \leq K \quad (22.18)$$

where $\|\cdot\|$ is a matrix norm.

A sufficient condition for stability is $\|C(\Delta t)\| \leq 1 + M\Delta t, M > 0$ is a constant.

Once we have proved consistency and stability, we can then conclude that a given scheme converges to the exact solution of the Cauchy problem (22.6) by appealing to the Lax Equivalence Theorem.

22.5 SOME POPULAR SPLITTING METHODS

We are now ready to introduce specific splitting schemes that have been used in many application areas, including computational finance. For a discussion of pioneering work, see Marchuk (1982), Yanenko (1971) and Strang (1969).

We have already discussed various forms of splitting and how it can introduce *splitting errors*. In this case, the Cauchy problem is one in which time is continuous. We now discretise each step of the splitting scheme by the Crank–Nicolson second-order scheme. Implicit and explicit Euler schemes are first-order accurate, and the latter method is only conditionally stable. Having discretised each leg in time, we get a discrete system of equations to be solved at each time level using the Thomas algorithms or Double Sweep (see Chapter 6), for example. Finally, each leg is an algorithm having input and output (matrices usually), and it is associated with a PDE whose components it needs to access, for example the coefficients of the PDE as well as boundary and initial conditions. A practical challenge is to decide on a suitable notation that is both understandable and facilitates an implementation in C++. We discuss this topic in more detail in Chapter 23.

We now discuss approximations of the initial value problem (22.6).

22.5.1 First-Order (Lie–Trotter) Splitting

For most schemes, we consider one or more legs on an interval $[t_n, t_{n+1}]$. The split form for the Lie–Trotter scheme is:

$$\begin{aligned}\frac{\partial u^*}{\partial t} &= A_1 u^*, u^*(t_n) \text{ given} \\ \frac{\partial u^{**}}{\partial t} &= A_2 u^{**}, u^{**}(t_n) = u^*(t_{n+1}) \\ u(t_{n+1}) &= u^{**}(t_{n+1}).\end{aligned}\tag{22.19}$$

and the splitting error is:

$$\text{Error} = \frac{1}{2} \Delta t (BA - AB) u(t_n) + O(\Delta t^2) \quad (\Delta t = t_{n+1} - t_n).$$

The quantity $u(t_{n+1})$ is the final solution.

The discrete form of (22.19) for the Crank–Nicolson method is:

$$\begin{aligned}\frac{u^{n+\frac{1}{2}} - u^n}{\Delta t} &= A_1^n \frac{u^{n+\frac{1}{2}} + u^n}{2} \\ \frac{u^{n+1} - u^{n+\frac{1}{2}}}{\Delta t} &= A_2^n \frac{u^{n+1} + u^{n+\frac{1}{2}}}{2} \\ \text{where } A_j^n &\equiv A_j \left(t_{n+\frac{1}{2}} \right) j = 1, 2.\end{aligned}\tag{22.20}$$

For completeness, the implicit Euler method is:

$$\begin{aligned}\frac{u^{n+1/2} - u^n}{\Delta t} &= A_1^n u^{n+1/2} \\ \frac{u^{n+1} - u^{n+1/2}}{\Delta t} &= A_2^n u^{n+1}.\end{aligned}\tag{22.21}$$

We discuss the application of this method to spread options in Chapter 23 as model/exemplar for a range of methods and problems.

22.5.2 Predictor-Corrector Splitting

This method is discussed in Marchuk (1982) and Yanenko (1971), and it can be seen as a variation of (22.20):

$$\begin{aligned}\frac{u^{n+1/4} - u^n}{\Delta t/2} &= A_1 u^{n+1/4} \quad (\text{Predictor}) \\ \frac{u^{n+1/2} - u^{n+1/2}}{\Delta t/2} &= A_2 u^{n+1/2} \quad (\text{Predictor}) \\ \frac{u^{n+1/2} - u^n}{\Delta t} &= A u^{n+1/2} \quad (\text{Corrector}).\end{aligned}\tag{22.22}$$

This scheme is second-order accurate in Δt if the operators A_1 and A_2 are positive-definite and *time-independent*. Note that we used implicit Euler for each step. The accuracy is first order if the matrices are time-dependent.

We discuss the application of this method to spread options in Chapter 23.

22.5.3 Marchuk's Two-Cycle (1-2-2-1) Method

This method is discussed in Marchuk (1982), and it is a composition of two instances of scheme (22.20). It is a popular and robust method. It is second-order accurate in Δt even for problems in which *time-dependent* A_1 and A_2 do not commute:

$$\frac{u^{n-1/2} - u^{n-1}}{\Delta t} = A_1^n \frac{u^{n-1/2} + u^{n-1}}{2} \quad (22.23)$$

$$\frac{u^n - u^{n-1/2}}{\Delta t} = A_2^n \frac{u^n + u^{n-1/2}}{2}$$

$$\frac{u^{n+1/2} - u^n}{\Delta t} = A_2^n \frac{u^{n+1} + u^n}{2}$$

$$\frac{u^{n+1} - u^{n+1/2}}{\Delta t} = A_1^n \frac{u^{n+1} + u^{n+1/2}}{2} \quad (22.24)$$

where $A_j^n \equiv A_j(t_n)$, $j = 1, 2$.

In the case of PDEs with nonhomogeneous term f , the Marchuk (Marchuk (1982), p. 240)) takes the form:

$$\begin{aligned} \left(I - \frac{\Delta t}{2} A_1^n \right) u^{n-1/2} &= \left(I + \frac{\Delta t}{2} A_1^n \right) u^{n-1} \\ \left(I - \frac{\Delta t}{2} A_2^n \right) (u^n - \Delta t f^n) &= \left(I + \frac{\Delta t}{2} A_2^n \right) u^{n-1/2} \\ \left(I - \frac{\Delta t}{2} A_2^n \right) u^{n+1/2} &= \left(I + \frac{\Delta t}{2} A_2^n \right) (u^n + \Delta t f^n) \\ \left(I - \frac{\Delta t}{2} A_1^n \right) u^{n+1} &= \left(I + \frac{\Delta t}{2} A_1^n \right) u^{n+1/2}, \text{ where } f^n = f(t_n). \end{aligned} \quad (22.25)$$

Another equivalent scheme for nonhomogeneous problems is given in Marchuk, Rusakov, Zalesny and Diansky (2005), and it includes useful tips on software design of the algorithm. This method can be used in combination with the Yanenko scheme for mixed derivatives.

The first application of Marchuk's scheme in our experience to computational finance is Levin and Duffy (2001) and Davidson and Levin (2014).

We discuss the application of this method to spread options in Chapter 23.

22.5.4 Strang Splitting

Strang splitting is an improvement on Lie–Trotter splitting (Strang (1969)):

$$\begin{aligned}
 \frac{\partial u^*}{\partial t} &= A_1 u^*, \quad t_n \leq t \leq t_{n+1/2} \\
 u^*(t_n) &= u_{sp}^n \quad u_{sp}^n = \text{desired solution at } t = t_n. \\
 \frac{\partial u^{**}}{\partial t} &= A_2 u^{**}, \quad t_n \leq t \leq t_{n+1} \\
 u^{**}(t_n) &= u^*(t_{n+1/2}) \\
 \frac{\partial u^{***}}{\partial t} &= A_1 u^{***}, \quad t_{n+1/2} \leq t \leq t_{n+1} \\
 u^{***}(t_{n+1/2}) &= u^{**}(t_{n+1}). \tag{22.26}
 \end{aligned}$$

where:

$$\begin{aligned}
 u_{sp}^{n+1} &= u^{***}(t_{n+1}) \text{ desired solution at } t_{n+1} \\
 t_{n+1/2} &= t_n + \frac{1}{2}\Delta t \quad (\Delta t = t_{n+1} - t_n).
 \end{aligned}$$

Can you see the relationship between (22.13) and (22.26)?

This method is second-order accurate. It is not (yet) widely known in finance, it would seem.

22.6 APPLICATIONS AND RELATIONSHIPS TO COMPUTATIONAL FINANCE

Splitting (and ADI) methods are popular when approximating the solutions of PDEs that describe two-factor options. Most PDEs in finance are instances of convection-diffusion-reaction equations containing a mixed derivative term and for positive values of the underlying variables (quarter-plane problem). In some cases (for example, two-factor Hull–White model), the PDE is defined in a plane. Before we decide on which scheme to use, we should examine the PDE in question with a view to gaining a better understanding of its qualitative properties. These are issues that are sometimes brushed under the carpet; the consequences are extensive numerical experimentation until a solution is found that bears some resemblance to reality. To avoid ad hoc/trial-and-error approaches, we can consider some ways to *preprocess* the PDE as we have already discussed in this book:

- P1: Domain transformation to the unit square; defining scale factors and hot spots.
- P2: Should we transform the PDE to *canonical form* to remove the mixed derivative term?

- P3: Determining suitable *numerical boundary conditions*; Dirichlet boundary conditions are ideal. We can use the Fichera theory, energy inequalities and Green's function to discover appropriate numerical boundary conditions.
- P4: Is the PDE convection-dominated? In such cases we can employ exponential fitting in order to ensure that the resulting finite difference schemes are monotone and that the resulting matrices are M-matrices.
- P5: Do we work with PDEs in conservative or non-conservative form? The former case may offer certain advantages, such as more compact code (the convection term 'disappears') and the possibility to use non-uniform meshes.

These action points are universal in the sense that they can be used as checklists for any one-factor and two-factor PDE in finance. When starting on a PDE project, you can use them as the driver on how to proceed.

22.7 SOFTWARE DESIGN AND IMPLEMENTATION GUIDELINES

Although this book is not primarily about software design, we do feel it is important to mention some guidelines on creating a software framework that implements splitting methods. This is a daunting task, and it should be clear that a software framework will be created by taking an incremental approach. We see software products as the end result of a series of (incomplete) prototypes (Boehm (1986)). In particular, the first prototype could be code to see if we can get a simple use case up and running:

The management question, therefore, is not whether to build a pilot system and throw it away. You will do that. The only question is whether to plan in advance to build a throwaway, or to promise to deliver the throwaway to customers.

(Brooks (1995))

Some requirements regarding the software framework are:

- R1: It produces accurate results when compared to baseline (exact) results.
- R2: It has good run-time performance.
- R3: It can be adapted and extended to suit new requirements.

We realise requirement R3 by decomposing a software system into a collection of autonomous *modules* as shown in Figure 22.1. This simple *context diagram* results from a defined process that allows us to achieve this end (Duffy (2018), Chapter 9). Each module has a single major responsibility, and it is usually a class or a function. Modules help divide large amounts of code into logical parts, and they are supported in Python and C++20.

We show how Figure 22.1 comes to life in Chapter 23 by taking a concrete example.

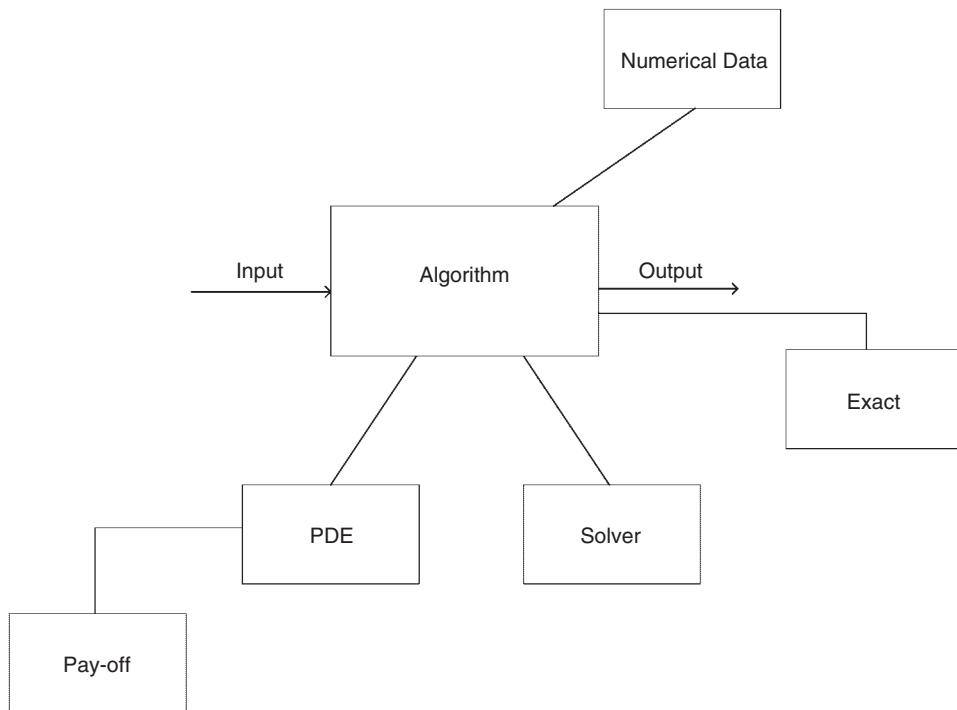


FIGURE 22.1 Block diagram for software framework.

It is also possible to create finance applications in C++ with design patterns. See Ballabio (2020).

22.8 EXPERIENCE REPORT: COMPARING ADI AND SPLITTING

In this section we summarise some of the conclusions in Wernick (2015) (which is an excellent MSc thesis, supervised by the author of this book) that compares ADI and splitting methods with regards to accuracy, stability, ease of application and speed. The corresponding model is a multi-asset PDE with the following pay-offs: exchange option, max of two assets call option and cash-or-nothing option. Both domain truncation (with $S_{max} = 4^*K$) and domain transformation are used, and the Fichera theory helps find numerical boundary conditions. A precise analysis is performed to prove that these schemes are consistent, stable and hence convergent. The mixed derivative term is approximated by the Yanenko scheme (as discussed in Chapter 18; classic ADI uses the Craig–Sneyd scheme). The splitting scheme used in the thesis is the first-order scheme (22.21), which is less accurate and robust than schemes (22.22), (22.23) and (22.24).

The norms to measure the error are:

$$\begin{aligned} RMSE &= \sqrt{\frac{1}{N} \sum_{ij}^N (u_{ij}^e - u_{ij})^2} \text{ (root mean square error)} \\ \|e\|_2 &= \frac{\|[u_{ij}^e] - [u_{ij}]\|_2}{\|[u_{ij}^e]\|_2} \text{ (discrete } l_2 \text{ norm)} \\ \|e\|_\infty &= \frac{\|[u_{ij}^e] - [u_{ij}]\|_\infty}{\|[u_{ij}^e]\|_\infty}, \text{ (maximum norm)} \end{aligned}$$

where u_{ij}^e is the exact solution and u_{ij} is the numerical solution at a given mesh point (i,j) .

We complete this section with a comparison of the differences between ADI and splitting (also called OS (operator splitting)):

- OS outperforms ADI in terms of stability.
- ADI converges faster than OS as the step size decreases until a critical step size is reached, at which stage the ADI solution *blows up*, while OS does not have these problems:

Splitting: $\Delta t = 0.0125$, $\|e\|_2 = 0.0017$, $\|e\|_\infty = 0.0025$, $RMSE = 0.0420$

ADI: $\Delta t = 0.0125$, $\|e\|_2 = 0.0015$, $\|e\|_\infty = 0.0018$, $RMSE = 0.00357$

ADI: $\Delta t = 0.0625$, $\|e\|_2 = 0.0015$, $\|e\|_\infty = RMSE = NA$ (blow – up).

- For-cash-or nothing options, ADI becomes unstable on the lines $S_1 = K_1$, $S_2 = K_2$. Here the pay-off is given by:

$$\text{Payoff}(S_1, S_2) = \begin{cases} \text{Cash if } S_1 \geq K_1 \text{ and } S_2 \geq K_2 \\ 0, \text{ otherwise} \end{cases}$$

where K_1 and K_2 are the strike prices of S_1 and S_2 , respectively.

- The pay-off is like a box with steep edges (see Jeong and Kim (2013)). Finally, classic ADI methods have stability issues for large values of correlation in the mixed derivative term; see, for example, Sheppard (2007) for a discussion of why splitting was used instead of ADI.
- ADI produces an oscillatory solution at the first step, and hence ADI shows a non-smooth numerical solution. OS produces a smooth numerical solution.

In Wernick (2015) the ADE method was also discussed, but the mixed derivative term was approximated by a somewhat unorthodox numerical scheme. Three questions are:

1. Can the Yanenko scheme be used with ADE?
2. What is the ‘ideal’ mixed derivative scheme to use in conjunction with ADE?
3. Transform a PDE to *canonical form* (see Chapter 8) and then apply ADE to this modified PDE.

In Chapter 23 we continue with an analysis of multi-asset options, including implementation of splitting algorithms in C++.

22.9 SUMMARY AND CONCLUSIONS

In this chapter we formalised the mathematical and numerical foundations of splitting methods from Chapter 18. At a high level, each time-dependent convection-diffusion-reaction equation can be posed as an operator equation in Banach space (a special and common case is when the operator is a square matrix). The solution of this problem is called a C_0 semi-group, which can be seen as a generalisation of the exponential function. This approach allows us to analyse operator splitting and time splitting of multidimensional PDEs in a uniform way. In particular, it now becomes easy to prove consistency and stability of splitting schemes.

We have given a detailed mathematical introduction to splitting methods as well as a discussion of several popular splitting methods that are used in practice. We trace a defined process starting with a (possibly preprocessed) PDE and then transforming it in such a way that it can be approximated numerically. This process can be applied to your own PDE problems, and hence this chapter uses all the techniques from the previous chapters of this book.

Test Cases in Computational Finance

CHAPTER 23

Multi-Asset Options

An expert is a person who has made all the mistakes that can be made in a very narrow field.

—Niels Bohr

23.1 INTRODUCTION AND OBJECTIVES

The main goal of this chapter is to benchmark the splitting methods that we introduced in Chapters 18 and 22 by applying them to two-asset option pricing problems. The focus is on showing how to implement the action points A1 to A11 of Chapter 22 to these kinds of problems. In this way we gain real hands-on experience on how to get a C++ framework up and running. This experience can then be used in other derivatives pricing problems. In other words, we create the numerical building blocks, implement them in C++ and then integrate them into the framework. Thus, the four main topics that we wish to accentuate here are:

- A1: Testing and evaluating several splitting schemes.
- A2: Assembling the finite difference equations and solving tridiagonal matrix systems.
- A3: Designing a software framework.
- A4: Acceptance testing. (Is the code accurate, robust and efficient?)

When implementing software systems using PDE/FDM, our approach is to get a software prototype up and running and test the output against known analytical solutions (for example, Haug (2007)), without yet worrying too much about software elegance or maintainability. The next stage could entail adding more functionality to the framework in some way while at the same time trying to ensure that the software design remains flexible.

This chapter uses results from Chapter 9 (Section 9.2.2 on domain transformation, Section 9.4), Chapter 11 (Section 11.5 on Fichera theory and discovery of boundary conditions), Chapter 18 (introduction to splitting methods, in particular Sections 18.4.2 and

18.4.2) and Chapter 22 (in particular, the Lie–Trotter, predictor–corrector and Marchuk Two-Cycle methods in Sections 22.5.1, 22.5.2 and 22.5.3). We will also use the Double Sweep method (Section 6.4) when solving tridiagonal systems of equations. Finally, we use the Yanenko method (Section 18.4.3) for the mixed derivative terms.

23.2 BACKGROUND AND GOALS

The goal of this chapter is to discuss a defined and standardised process to price European and American options on two assets. There are numerous kinds of options with names such as basket, rainbow, product, correlation, exchange and min/max options (see Haug (2007) for a discussion, including analytic solutions). They all satisfy the same two-factor convection-diffusion-reaction PDE, the only difference being the specific pay-off in question and the boundary conditions. Without loss of generality, we examine call and put spread options whose pay-offs are:

$$\begin{aligned} S(t) &= S_1(t) - S_2(t), \quad t \geq 0 \\ f_T(S) &= \max(0, S(T) - K) \equiv (S(T) - K)^+ \text{ (Call)} \\ f_T(S) &= \max(0, K - S(T)) \equiv (K - S(T))^+ \text{ (Put)} \\ \text{where in general } a^+ &\equiv \max(0, a). \end{aligned} \tag{23.1}$$

The PDE of interest is:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{1}{2} \sigma_1^2 S_1^2 \frac{\partial^2 u}{\partial S_1^2} + (r - q_1) S_1 \frac{\partial u}{\partial S_1} \\ &+ \frac{1}{2} \sigma_2^2 S_2^2 \frac{\partial^2 u}{\partial S_2^2} + (r - q_2) S_2 \frac{\partial u}{\partial S_2} + \rho \sigma_1 \sigma_2 S_1 S_2 \frac{\partial^2 u}{\partial S_1 \partial S_2} - ru, \quad 0 < S_1, S_2 < \infty. \end{aligned} \tag{23.2}$$

PDE models and traditional finite difference methods for two-asset option pricing problems are widespread.

The common features of these methods are:

- Using the Alternating Direction Implicit (ADI) method with centred differencing for the diffusion and convection terms in the underlying state variables and the Crank–Nicolson method in time.
- Ad hoc and heuristic domain truncation and then experimentation to discover/concoct numerical boundary conditions at the near-field and far-field boundaries.
- In many cases, the log transformation $x = \log S_1$, $y = \log S_2$ that maps the PDE (23.2) on a quarter plane to a constant-coefficient PDE in the plane, similar to a two-dimensional version of the Cauchy–Euler equation (see Equation (11.2)):

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{1}{2} \sigma_1^2 \frac{\partial^2 u}{\partial x^2} + \nu_1 \frac{\partial u}{\partial x} + \frac{1}{2} \sigma_2^2 \frac{\partial^2 u}{\partial y^2} + \nu_2 \frac{\partial u}{\partial y} + \rho \sigma_1 \sigma_2 \frac{\partial^2 u}{\partial x \partial y} - ru \\ \nu_j &= r - q_j - \frac{1}{2} \sigma_j^2, \quad j = 1, 2 \quad -\infty < x, y < \infty. \end{aligned} \tag{23.3}$$

We truncate both near-field and far-field boundaries in both underlyings. We remark that we were able to remove the mixed derivative term in Equation (23.3), using transforms as described in Section 9.6.

We distance ourselves from the above approach in this chapter because we discuss a more mathematically robust approach to solving Equation (23.2):

- Transform the PDE (23.2) in the positive quarter plane to a PDE with non-constant coefficients on the unit square:

$$\begin{aligned}\frac{\partial u}{\partial t} &= \frac{1}{2}\sigma_1^2x^2\frac{\partial}{\partial x}\left\{(1-x)^2\frac{\partial u}{\partial x}\right\} + rx(1-x)\frac{\partial u}{\partial x} \\ &\quad + \frac{1}{2}\sigma_2^2y^2\frac{\partial}{\partial y}\left\{(1-y)^2\frac{\partial u}{\partial y}\right\} + ry(1-y)\frac{\partial u}{\partial y} + \rho\sigma_1\sigma_2x(1-x)y(1-y)\frac{\partial^2 u}{\partial x\partial y} - ru\end{aligned}$$

where:

$$x = \frac{S_1}{S_1 + \alpha_1}, \quad y = \frac{S_2}{S_2 + \alpha_2}; \quad 0 < x, y < 1. \quad (23.4)$$

(α_1 and α_2 are so-called *scale factors*).

- Apply the Fichera theory to (23.4) to discover the mathematically correct numerical boundary conditions.

To complete this section, we give a preview of how we intend to solve the problem (23.4) and associated pay-off:

- We can approximate (23.4) in its current conservative form or alternatively in its non-conservative form:

$$\begin{aligned}\frac{\partial u}{\partial t} &= \frac{1}{2}\sigma_1^2x^2(1-x)^2\frac{\partial^2 u}{\partial x^2} + \{rx(1-x) - \sigma_1^2x^2(1-x)\}\frac{\partial u}{\partial x} \\ &\quad + \frac{1}{2}\sigma_2^2y^2(1-y)^2\frac{\partial^2 u}{\partial y^2} + \{ry(1-y) - \sigma_2^2y^2(1-y)\}\frac{\partial u}{\partial y} \\ &\quad + \rho\sigma_1\sigma_2x(1-x)y(1-y)\frac{\partial^2 u}{\partial x\partial y} - ru.\end{aligned} \quad (23.5)$$

We work with form (23.5) in this chapter.

- The Fichera theory shows that the PDE (23.5) does not need boundary conditions; for numerical boundary conditions, we use the transformed version of the pay-off function (23.1) in (x, y) space.
- We approximate (23.5) using the Lie-Trotter, predictor-corrector and Marchuk Two-Cycle methods as discussed in Sections 22.5.1, 22.5.2 and 22.5.3.
- Finally, we use the Yanenko method (Section 18.4.3) for mixed derivatives.

23.3 THE BIVARIATE NORMAL DISTRIBUTION (BVN) AND ITS APPLICATIONS

Many two-factor European options have analytic or semi-analytic solutions (Zhang (1998), Haug (2007)). In general, the corresponding formulae are based on either the

cumulative univariate normal distribution or the *bivariate normal density function* (BVN).

The *error function* (also called the *Gaussian error function*) is defined by:

$$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x e^{-t^2} dt = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt. \quad (23.6)$$

In statistics, this function has the following interpretation: let X be a normally distributed random variable with mean 0 and variance 1/2. Then $\operatorname{erf}(x)$ describes the probability of X falling in the range $[-x, x]$ where $x \geq 0$. The *complementary error function* is defined by:

$$\operatorname{erfc}(x) = 1 - \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt. \quad (23.7)$$

C++11 supports both the error function and the complementary error function. Both functions accept `float`, `double`, `long double` and integral arguments:

```
// Error functions
int i = 1;
float f = 3.0F;
double d = 3.14;
long double d2 = 2.71;
std::cout << "i, erf(x): " << i << ", " << std::erf(i) << '\n';
std::cout << "f, erf(x): " << f << ", " << std::erf(f) << '\n';
std::cout << "d, erfc(x): " << d << ", " << std::erfc(d) << '\n';
std::cout << "d2, erfc(x): " << d2 << ", " << std::erfc(d2) << '\n';
```

In Figure 23.1 we give the values of the error functions for special cases of their arguments.

The error functions are related to the *cumulative distribution* Φ (integral of the standard normal distribution) by the formulae:

$$\Phi(x) \equiv N(x) = \frac{1}{2} + \frac{1}{2} \operatorname{erf}(x/\sqrt{2}). \quad (23.8)$$

<i>x</i>	<i>erf</i>	<i>erfc</i>
$+\infty$	+1	+ 0
$-\infty$	-1	2
<i>NaN</i>	<i>NaN</i>	<i>NaN</i>

FIGURE 23.1 Error handling and extreme values for the error functions.

We can check the following relationships:

$$\begin{aligned} \text{erf}(x) &= 2\Phi(x\sqrt{2}) - 1 \\ \text{erfc}(x) &= 2\Phi(-x\sqrt{2}) = 2(1 - \Phi(x\sqrt{2})). \end{aligned} \tag{23.9}$$

The error function has many applications in statistics, probability and partial differential equations.

We continue our discussion of statistical distributions. We create efficient algorithms to compute the *cumulative bivariate normal distribution* (BVN) function defined by:

$$M(a, b; \rho) = \int_{-\infty}^a \int_{-\infty}^b f(x, y; \rho) dx dy \tag{23.10}$$

where in this case:

$$f(x, y; \rho) = \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\left[-\frac{x^2 - 2\rho xy + y^2}{2(1-\rho^2)}\right], \quad -1 < \rho < 1.$$

This distribution has many applications, and our interest is in using it to price options on two assets (see for example, Haug (2007)). In contrast to the univariate case, there seems to be little software available to compute bivariate distributions, and in these cases we resort to a combination of third-party libraries and code written by this author. We propose a number of alternatives, and the final choice will be determined by a number of requirements such as efficiency, accuracy, robustness and maintainability:

- S1: The Drezner method (Drezner (1978)).
- S2: The Genz method (Genz (2004)) and its implementation in *Quantlib* (www.quantlib.org) and by West (West (2004)).
- S3: The formula 26.3.3 in Abramowitz and Stegun (1965) in combination with a Gauss–Legendre numerical quadrature scheme.
- S4: Converting the two-dimensional integral (23.10) defining the cumulative bivariate normal distribution into a *hyperbolic partial differential equation* (PDE) of Goursat type that we approximate using the finite difference method as developed by the author in Duffy (2018).

Summarising, the main goal (in Duffy (2018)) is to propose a number of algorithms to compute the bivariate normal distribution and to compare their relative efficiency, accuracy and usability characteristics. In this way, we choose the specific algorithm from the above list S1 to S4 that best suits our requirements in a given context. We apply these algorithms to price two-factor plain options. The code is implemented using *callable objects* that the option pricing code uses as flexible *plug-ins* (*mixins*), thus allowing us to switch between implementations.

The Genz method for Equation (23.10) is for specific values of its arguments. It is very accurate. We have tested the West and Quantlib C++ implementations against each other, and they are in complete agreement. We now discuss a PDE-based method, originated by the author to approximate (23.10) by writing it as an equivalent PDE. We then approximate the PDE using the finite difference method on a truncated domain (see Duffy (2018) for full details).

23.3.1 Computing BVN by Solving a Hyperbolic PDE

Computing Integrals using Partial Differential Equations (PDEs)

Looking at Equation (23.10), we might naively suspect that we can compute the double integral either exactly or by using numerical quadrature. There seems to be a general consensus that an exact solution is not possible in this case, and computational efficiency and accuracy will be a major concern when we decide which numerical scheme to use.

In this section we take a completely different approach by remarking that certain kinds of integrals can be written as a differential equation and vice versa. In the current context we can differentiate Equation (23.10) with respect to the variables a and b (which we immediately change to x and y , respectively) to produce a partial differential equation of *hyperbolic type*:

$$\frac{\partial^2 u}{\partial x \partial y} = f(x, y; \rho). \quad (23.11)$$

This is an example of a *hyperbolic PDE*, and it can be seen as a second-order *wave equation*. In fact, the transformation:

$$\xi = (x + y)/2, \quad \eta = (x - y)/2 \quad (23.12)$$

reduces Equation (23.11) to a PDE of the form:

$$\frac{\partial^2 u}{\partial \xi^2} - \frac{\partial^2 u}{\partial \eta^2} = f. \quad (23.13)$$

Both Equations (23.11) and (23.13) are special cases of a *Goursat PDE* (Courant and Hilbert (1968)). A full treatment is beyond the scope of this book, because the goal is to use the corresponding techniques to compute the integral in Equation (23.10). To this end, we analyse the *boundary value problem* associated with Equation (23.11), which is different from that used for parabolic PDEs. In the current case we model the problem as a wave travelling in a given direction from a start point. In other words, information is travelling from a single part of the boundary and not from the full boundary, as is the case with elliptic problems. We arbitrarily truncate the infinite domain of integration to a rectangle $(AL, x) \times (BL, y)$ where $AL = BL = -8$ and x and y are the upper values

where we wish to compute the integral (23.10). This leads us to the lower boundary values:

$$u(AL, y) = u(x, BL) = 0, \quad AL \leq x < \infty, \quad BL \leq y < \infty. \quad (23.14)$$

This approach uses *domain truncation*. Incidentally, we have also implemented this problem using domain transformation, and we have produced the same results as in this chapter.

Summarising, Equations (23.11) and (23.14) are the defining equations and starting point for the finite difference method that we employ.

Some of the use cases in which this approach can be used are:

- U1: It is robust and accurate, and we can tune the desired level of accuracy for a range of arithmetic types such as `double`, `float` and multiprecision data types.
- U2: The algorithm is easy to program, and it does not need third-party libraries with the exception of *Boost C++* matrices. It is easy to modify the code to work with user-defined matrix classes.
- U3: We use it as a benchmark to test the accuracy of other algorithms, for example the algorithm in Genz (2004) and the corresponding implementations in *Quantlib*.
- U4: The algorithms compute a matrix of values at discrete mesh points up to and including the point where the value is desired. We can then use this matrix as a *lookup table* in combination with linear and cubic spline interpolation. The matrix is computed once.
- U5: The algorithm can be applied to any bivariate distribution as well as to trivariate distributions without our having to carry out extensive and complex mathematical analysis.
- U6: The algorithm uses methods from PDE and FDM theory.

In general, the requirements for a given problem will determine which particular algorithm to use.

The Finite Difference Method for the Goursat PDE

We now discuss the application of the finite difference method to compute an approximation to the solution of systems (23.11) and (23.14). We define discrete meshes in the x and y directions in the usual way. We store the discrete solution as a *Boost* matrix. The resulting second-order finite difference scheme is given by:

$$\frac{V_{i,j} - V_{i,j-1} - V_{i-1,j} + V_{i-1,j-1}}{h_x h_y} = f_{i-1/2, j-1/2} \quad (23.15)$$

where:

$$f_{i-1/2, j-1/2} = f(x_{i-1} + h_x/2, y_{j-1} + h_y/2), \quad 1 \leq i \leq I, \quad 1 \leq j \leq J$$

and h_x and h_y are the constant mesh sizes in the x and y directions, respectively.

An alternative to (23.11) and (23.15) is to solve the PDE problem on a bounded domain:

$$\begin{aligned} z &= \tanh x, \quad w = \tanh y \\ x &= \frac{1}{2} \log((1+z)/(1-z)), \quad y = \frac{1}{2} \log((1+w)/(1-w)). \end{aligned} \tag{23.16}$$

The transformed PDE becomes:

$$(1-z^2)(1-w^2) \frac{\partial^2 u}{\partial z \partial w} = f(z, w), \quad -1 < z, w < 1.$$

For boundary conditions, we take the same values as in (23.14).

We also mention that this method can be used to compute values of the trivariate normal distribution:

The *trivariate normal distribution* has probability density function defined by:

$$N(b_1, b_2, b_3; \Sigma) = \frac{1}{(2\pi)^{3/2} \sqrt{|\Sigma|}} \int_{-\infty}^{b_1} \int_{-\infty}^{b_2} \int_{-\infty}^{b_3} f(x, y, z; \Sigma) dz dy dx$$

where Σ is the positive-definite covariance and $|\Sigma|$ is its determinant:

$$\Sigma = \begin{pmatrix} 1 & \rho_{12} & \rho_{13} \\ \rho_{12} & 1 & \rho_{23} \\ \rho_{13} & \rho_{23} & 1 \end{pmatrix}, \tag{23.17}$$

$$f(x, y, z; \Sigma) = \exp\left(-\frac{1}{2} w^\top \Sigma^{-1} w\right), \quad w = (x, y, z)^\top, \quad \text{and } \Sigma^{-1} \text{ is the inverse of } \Sigma.$$

We have implemented this function in C++ by explicitly computing the inverse of the covariance matrix. We also used a more explicit form for the probability density function:

$$f(b_1, b_2, b_3; \Sigma) = \frac{e^{-w}/[2(\rho_{12}^2 + \rho_{13}^2 + \rho_{23}^2) - 2\rho_{12}\rho_{13}\rho_{23}]}{2} \sqrt{2\pi^{3/2}} \sqrt{1 - (\rho_{12}^2 + \rho_{13}^2 + \rho_{23}^2) + 2\rho_{12}\rho_{13}\rho_{23}}$$

where:

$$\begin{aligned} w &= b_1^2(\rho_{23}^2 - 1) + b_2^2(\rho_{13}^2 - 1) + b_3^2(\rho_{12}^2 - 1) \\ &+ 2[b_1 b_2 (\rho_{12} - \rho_{13} \rho_{23}) + b_1 b_3 (\rho_{13} - \rho_{12} \rho_{23}) + b_2 b_3 (\rho_{23} - \rho_{12} \rho_{13})]. \end{aligned}$$

Similar to the two-dimensional case, we can differentiate the integral to produce a *third-order hyperbolic* PDE:

$$\frac{\partial^3 u}{\partial x \partial y \partial z} = f. \tag{23.18}$$

We solve (23.18) by a finite difference method that is the three-dimensional analogue of scheme (23.15):

$$\begin{aligned} V_{i,j,k} = & V_{i-1,j,k} + V_{i,j-1,k} - V_{i-1,j-1,k} + V_{i,j,k-1} - V_{i-1,j,k-1} - V_{i,j-1,k-1} + V_{i-1,j-1,k-1} \\ & + h_x h_y h_z f_{i-1/2,j-1/2,k-1/2} \end{aligned}$$

where:

$$x_{i-1/2} = x_{i-1} + h_x/2 \quad (23.19)$$

$$y_{j-1/2} = y_{j-1} + h_y/2$$

$$z_{k-1/2} = z_{k-1} + h_z/2,$$

$$1 \leq i \leq NX, 1 \leq j \leq NY, 1 \leq k \leq NZ.$$

We can create this scheme from first principles, as we did in the two-dimensional case, by composing divided differences in the x , y and z directions. Furthermore, we adopt the heuristic by defining the boundary values (similar to (23.14)):

$$U(AL, y, z) = u(x, BL, z) = u(x, y, CL) = 0, -\infty < x, y, z < \infty$$

where AL , BL and CL are truncation values (typically $AL = BL = CL = -8$).

Finally, quadvariate normal distributions and their PDE approximation are explored in Duffy (2018).

23.3.2 Analytical Solutions of Multi-Asset and Basket Options

We give two examples for completeness; the first case is a *chooser option* that uses (23.10) while the second is a spread option whose analytic solution will be compared against the corresponding solution produced by splitting methods in this chapter.

A *chooser option* is an option where the holder has the right to choose whether it is a call or a put at some point in its life. The value of a chooser option at the point t where the choice is made is given by:

$$w(S, X_c, X_p, t, T_c, T_p) = \max[CBSM(S, X_c, T_c), PBSM(S, X_p, T_p)],$$

where $CBSM(S, X, T)$ and $PBSM(S, X, T)$ are the values of the call and put underlying the option, respectively. These types of options can be priced using lattice methods as discussed in Mun (2002) using the binomial method. A complex chooser option is one that is based on two expirations T_c, T_p and two strikes X_c, X_p . The price of the complex chooser option is then:

$$\begin{aligned} w = & S e^{(b-r)T_c} M(d_1, y_1; \rho_1) \\ & - X_c e^{-rT_c} M(d_2, y_1 - \sigma \sqrt{T_c}; \rho_1) - S e^{(b-r)T_p} M(-d_1, -y_2; \rho_2) \\ & + X_p e^{-rT_p} M(-d_2, -y_2 + \alpha \sqrt{T_p}; \rho_2) \end{aligned}$$

where:

$$d_1 = \frac{\log(S/I) + (b + \sigma^2/2)t}{\sigma\sqrt{t}}, \quad d_2 = d_1 - \sigma\sqrt{t}$$

$$y_1 = \frac{\log(S/X_c) + (b + \sigma^2/2)T_c}{\sigma\sqrt{T_c}}, \quad y_2 = \frac{\log(S/X_p) + (b + \sigma^2/2)T_p}{\sigma\sqrt{T_p}}$$

$$\rho_1 = \sqrt{t/T_c}, \quad \rho_2 = \sqrt{t/T_p}$$

and the value I is the solution of the non-linear equation:

$$Ie^{(b-r)(T_c-t)}N(z_1) - X_c e^{-r(T_c-t)}N(z_1 - \sigma\sqrt{T_c-t})$$

$$+ Ie^{(b-r)(T_p-t)}N(-z_2) - X_p e^{-r(T_p-t)}N(-z_2 + \alpha\sqrt{T_p-t}) = 0$$

$$z_1 = \frac{\log(I/X_c) + (b + \sigma^2/2)(T_c - t)}{\sigma\sqrt{T_c-t}}, \quad z_2 = \frac{\log(I/X_p) + (b + \sigma^2/2)(T_p - t)}{\sigma\sqrt{T_p-t}}.$$

The C++ code for a spread option is:

```
// Q1 and Q2 assumed known
auto ExactSolution = [&](double S1, double S2, double t)
{ // Spread option max(S1 - S2 - X, 0) or max(X - S1 + S2, 0) in this case

    if (K == 0.0)
        return ExactSolutionExchange(S1, S2, t);

    // Haug, not dividends, coc = r
    double quot
        = Q2 * S2 * std::exp((coc2 - r) * t) + K * std::exp(-r * t);
    double F = Q2 * S2 * std::exp((coc2 - r) * t) / quot;
    double S = Q1 * S1 * std::exp((coc1 - r) * t) / quot;
    double sig
        = std::sqrt(sig1*sig1 + sig2 * sig2*F*F - 2.0*rho*sig1*sig2 * F);

    double d1 = (std::log(S) + 0.5 * sig * sig * t) / (sig * std::sqrt(t));
    double d2 = d1 - sig * std::sqrt(t);
    return (Q2 * S2 * std::exp((coc2 - r) * t) + K * std::exp(-r * t))
        * (N(-d2) - S * N(-d1)); // p

    // return (Q2*S2*std::exp((coc2 - r)*t) + K * std::exp(-r * t))
    // * (S*N(d1) - N(d2)); // c
}
```

The problem with non-linear schemes is that they do not always converge in a given number of iterations. In other words, there is no guaranteed ‘quality of service’.

23.4 PDE MODELS FOR MULTI-ASSET OPTION PROBLEMS: REQUIREMENTS AND DESIGN

We now commence our analysis of PDE (23.5) with initial condition (pay-off) of the form (23.1). We immediately note that we will need to model this pay-off and boundary condition in code using (x, y) transformed variables and not state variables (S_1, S_2) . We apply three of the splitting methods from Chapter 22 using centred differencing in space and first-order or second-order differencing in time (in most cases we use Crank–Nicolson). We also note that a first-order or second-order splitting error is incurred when we perform splitting, and this is noticeable, especially with Lie–Trotter splitting.

We recommend paying careful attention to action points A1 to A11 in Section 22.1 (Chapter 22) and to action points P1 to P5 in Section 22.6 before embarking on a finite difference project. For optimal human performance, you need to be familiar with the methods from Chapters 18 and 22.

23.4.1 Domain Transformation

In most cases we prefer to use domain transformation rather than domain truncation because we avoid incurring new approximations, and it allows us to discuss boundary conditions using a defined process. The transformation and its inverse in this case are given by:

$$\begin{aligned} x &= \frac{S_1}{S_1 + \alpha_1}, & y &= \frac{S_2}{S_2 + \alpha_2} \\ S_1 &= \frac{\alpha_1 x}{1 - x}, & S_2 &= \frac{\alpha_2 y}{1 - y}. \end{aligned} \tag{23.20}$$

We remark that the scale factors α_1 and α_2 do not appear in the transformed PDE (23.4) and (23.5) but only in the transformed pay-off function and those boundary conditions that use the transformed pay-off function. The rationale for defining scale factors in (23.20) is that we wish to define hotspots in state space (for example, $(S_1 = 100, S_2 = 100)$) where we wish to know the option price. But since we work in transformed (x, y) space, we wish to know what the corresponding point in the state space is. To this end, we choose the scale factors as follows:

$$\alpha_1 = \frac{S_1(1 - x)}{x}, \quad \alpha_2 = \frac{S_2(1 - y)}{y}. \tag{23.21}$$

In many cases we set $x = y = 1/2$. Finally, we may wish to compute the option price in a box or window centred on the hotspot. These features are incorporated into our software design, and we can test the accuracy of the finite difference method by comparing it with the exact solution in this box.

23.4.2 Numerical Boundary Conditions

The fact that we use domain transformation and that the Fichera theory can be applied to the transformed PDE allows us to conclude that continuous boundary conditions are

not needed for (23.5). (We prove this by showing that the Fichera function (11.21) is identically zero on the boundary of the unit square.) However, we do need *numerical boundary conditions*, and we take them to be equal to the values of the pay-off function on the boundaries (the so-called *pay-off boundary conditions*). We adopt this approach in this chapter. Other possibilities are to extend (23.5) to the boundaries and then to solve them exactly (if an analytic solution is known) or by a finite difference approximation.

In Section 23.7.1 we discuss how other textbooks deal with the topic of domain truncation and the discovery of numerical boundary conditions.

23.5 AN OVERVIEW OF FINITE DIFFERENCE SCHEMES FOR MULTI-ASSET OPTION PROBLEMS

Having described the initial boundary value problem for two-asset option pricing, we now embark on discussing how to approximate its solution using the splitting methods from Chapter 22. The focus is on the following process steps:

- S1: Setting up a modular software framework.
- S2: Designing the algorithms and data structures for the splitting methods.
- S3: Implementing and testing the code for splitting methods.

At this stage in the book, we have enough understanding of the mathematics and the numerics to allow us to proceed to the C++ design and implementation of the corresponding algorithm.

23.5.1 Common Design Principles

Although this is not a book on how to design and implement efficient and maintainable software systems, we do attempt to give some guidelines on how to proceed. (A more complete account is given in Duffy (2018); in particular, in Chapter 9 of that book we give an overview of the *unified software design* (USD) process.)

Some of the principles underlying our design approach can be summarised by the steps that György Pólya describes when solving a mathematical problem (Pólya (1990)):

1. First, you have to *understand the problem*.
2. After understanding what you are trying to solve, *make a plan*.
3. *Carry out the plan*.
4. *Look back at your work. How could it be better?*

We see these steps as being applicable to the software development process in general and to the creation of software for computational finance in particular. Getting each step right saves time and money. In short, we adopt the following tactic: *get it working, then get it right and only then get it optimised* (in that order).

We have applied USD to problems in a number of domains in the past when the author was a requirements analyst and software designer (Duffy (2004a)). In this book we use USD in the background for computational finance applications. The examples in this chapter are generic and have been chosen to suit a wide audience.

We discuss the methods that have shaped and influenced USD. It is a multi-paradigm method to analyse, design and implement applications in a range of domains: for example, process control, logistics and computer-aided design (CAD). A discussion of these kinds of applications is outside the scope of this book. See Duffy 2004 for a discussion of this approach. Instead, we focus on applications related to computational finance and how they are implemented in C++.

USD can best be described as a process to analyse and design software systems. It uses features from a number of well-known system design approaches, while at the same time it tries to avoid features that are more difficult to apply. The main phases in USD reflect the bespoke Pólya's steps:

- Phase I: System Scoping and Initial System Decomposition.
- Phases II: Identifying System Components and Interfaces.
- Phase III: Detailed Design.
- Phase IV: Implementation, Review and Maintenance.

The execution of each phase can be seen as a process that maps input to output, and we describe the *core process* as a sequence of (computable) *activities* that tie the process's output to its input. Ideally, we define a low-risk and seamless process to take user requirements and map them to code. To this end, we have used a number of established methods to analyse and design software systems. We now give a short description of each method to provide the reader with background information.

A *domain architecture* (Duffy 2004a) is a reference model for a range of applications that share similar structure, functionality and behaviour. A domain architecture is a kind of *metamodel* that can be seen as a template for more specific systems.

We discuss five basic forms and one 'composite' form:

- *MIS* (Management Information Systems): Produce high-level and consolidated decision-support data and reports based on transaction data from various sources.
- *PCS* (Process Control Systems): Monitor and control values of certain variables that must satisfy certain constraints. We are primarily interested in *exceptional events* and events that must be handled in the software system.
- *RAT* (Resource Allocation and Tracking) systems: Monitor a request or some other entity in a system. The request is registered, resources are assigned to it, and its status in time and space is monitored. This is probably the most common model that we encounter in applications.
- *MAN* (Manufacturing) systems: Create finished products and services from raw materials and data.
- *ACS* (Access Control Systems): Allow access to passive objects from active subjects. These systems are similar to security systems and the *Reference Model* in large computer systems.

- *LCM* (Lifecycle Model): A ‘composite’ model that describes the full lifecycle of an entity; it is a composition of *MAN*, *RAT* and *MIS* models.

We have modelled the current option pricing problem as a *RAT* system because it is similar to the Monte Carlo application that we discuss elsewhere. We reason by analogy by examining the commonality between Monte Carlo pricing based on SDEs and PDE modelling of multi-asset options.

23.5.2 Detailed Design

The initial *context diagram* (*module design*) is shown in Figure 23.2; we shall elaborate on it in Section 23.7.2. Each dedicated block satisfies the *Single Responsibility Principle* (SRP) in the sense that it encapsulates one specific aspect of the design. For example, we have implemented a module that models the transformed non-conservative PDE (23.5), including early exercise constraint checker functions along rows and columns of the two legs of the splitting algorithms. We note that we have implemented the coefficients (as well as the transformed pay-off and boundary conditions) as free (global) functions in this version of the software. Encapsulation of this functionality in a namespace or a class is a piece of optimisation at this stage, and this step can be executed when the algorithms have stabilised.

The other modules in Figure 23.2 are:

- *OptionData (Source)*: the Black–Scholes-related parameters that are needed by the PDE.
- *Fdm*: Encapsulation of schemes (22.20), (22.22) and (22.23), including code for discrete initial and boundary conditions. In particular, it uses the PDE to populate the matrix data structure at each time level. For example, scheme (22.20) uses three

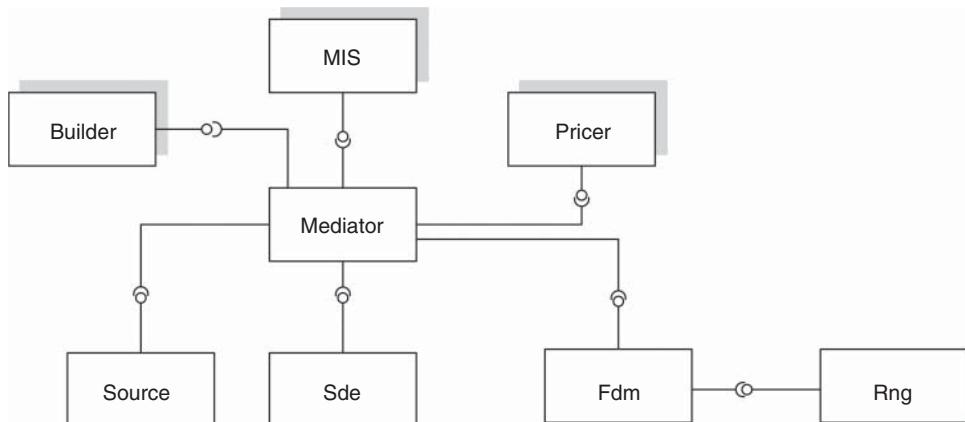


FIGURE 23.2 Context diagram for Monte Carlo engine.

matrices for time levels n , $n + 1/2$ and $n + 1$, and these matrices are updated as we march from $t = 0$ to $t = T$. We implement the legs of the scheme by functions F1 and F2 that we describe later in this section. The code snippet for scheme (22.20) is:

```
for (std::size_t n = 1; n < N3; ++n)
{
    tnow = tMesh[n];

    // Initialise TWO tridiagonal matrices once

    F1(MOld, MMid);
    F2(MMid, MNew);

    // Update to next time level
    MOld = MNew;
}
```

while the two-cycle method (22.23) uses the following update policy:

```
for (std::size_t n = 1; n < N3; ++n)
{
    tnow = tMesh[n];

    // Marchuk's 1-2-2-1 two-cycle model
    F1(MOld, M1);

    // M1 -> M2
    F2(M1, M2);

    // M2 -> M3
    F2(M2, M3);

    // M3 -> M4, final solution at level n+1
    F1(M3, M4);
}
```

- *Mediator*: The central management component in the application. It updates the above matrices at each time level.
- *Pricer*: The module that contains the code to compute the final option price.
- *PostProcessing*: The component that produces statistics on the accuracy of schemes by comparing them with the exact solution, both at hotspots and in a box around the hotspots. We compute the difference between the exact and numerical solutions for the l_2 , l_1 and l_∞ norms and we display the results in Excel using ExcelDriver from Duffy (2018).

- *Builder*: This is a design pattern to separate the construction of a complex object or program from its representation so that the same construction process can create different representations (Gamma, Helm, Johnson and Vlissides (1995)). Not taking this separation of concerns into account will result in a big *ball of mud*, and your code will sooner rather than later become unmaintainable.

23.5.3 Testing the Software

We have extensively tested schemes (22.20), (22.22) and (22.23), and we tested them against each other as well as against Haug (2007) and Chen (2017). We give a summary of the main findings.

The main use cases are:

- U1: Initial example taken from Haug (2007), pp. 214–15 (exact call price = 2.1670).
- U2: Extreme values of correlation $\rho = 0.95$, $\rho = -0.95$, $\rho = 0$.
- U3: Pricing American options.

For U1 we take a call option with $\rho = 0.42$, $\sigma_1 = 0.29$, $\sigma_2 = 0.36$, $r = 0.05$, $q_1 = q_2 = r$, $T = 0.25$, $K = 7$, $Q_1 = Q_2 = 1$. With $NX = NY = 200$, $NT = 500$ we get:

(22.20) with Yanenko scheme for mixed derivatives: 2.16964.

(22.20) with Craig–Sneyd for mixed derivatives: 2.16972.

(22.22) with Yanenko scheme in the corrector step: 2.16951.

(22.23) with Yanenko scheme in the corrector step: 2.1675. This is the best method in general because it is second order and commutativity of the splitted operators is not required.

For U2, we take the same parameters as U1; the case of zero correlation results in a convection-diffusion-reaction equation with no mixed derivative term. The exact value is 2.66068, while (23.23) gives 2.66019 (as expected) while the others give 2.16181 or thereabouts. (An interesting project is to transform a PDE to canonical form, as discussed in Chapter 8, and then to let (22.23) loose on the transformed PDE.) Large negative values of correlation are easily handled, but large positive values of correlation demand that we use larger values of mesh points. For example, in the case $\rho = 0.95$ we get $C = 1.14071$ for $NX = NY = 700$, $NT = 1300$ while the exact solution is 1.1369.

23.6 AMERICAN SPREAD OPTIONS

There is no analytic solution for two-factor American option pricing. We take the case of an American put spread option with $K = 50$, $S_1 = 110$, $S_2 = 60$, $T = 182/365$, $\sigma_1 = 0.4$, $\sigma_2 = 0.2$, $r = 0.1$. The comparisons for $\rho = 0.4$ and $\rho = 0.6$ are:

- Chen (2017): 10.343752, (22.22) gives 10.3437
and
- Chen (2017): 9.593279, (22.22) gives 9.59385, respectively.

For completeness, we show how the code for early exercise is integrated with the splitting algorithm in the case of one leg:

```

auto F1 = [&](const NestedMatrix<double>& matA, NestedMatrix<double>& matB)
{
    DiscreteBC(xMesh, yMesh, tnow - k, matB);
    // MA -> MB
    // Calculate value at n+1/2, coefficient at level n+1/2 (implicit)
    for (std::size_t j = 1; j < N2 - 1; ++j)
    { // For each j, solve a linear system in i

        ym = yMesh[j];
        double BCL = matB(0, j); double BCR = matB(matB.size1() - 1, j);
        for (std::size_t i = 1; i < N1 - 1; ++i)
        {
            xm = xMesh[i];
            lambda1 = (k*a11(xm, ym, tnow) / hx2) / 2;
            conv1 = k * b1(xm, ym, tnow) / (2.0 * hx) / 2;
            reaction = k * c(xm, ym, tnow) / 4;

            // Divide by 2 for average and put old values
            AX[i] = -lambda1 + conv1;
            BX[i] = (1.0 + 2.0 * lambda1) - reaction;
            CX[i] = -lambda1 - conv1;
            double tmp1, tmp2, tmp3;
            tmp1 = matA(i + 1, j);
            tmp2 = matA(i, j);
            tmp3 = matA(i - 1, j);
            FX[i] = lambda1 * (tmp1 - 2.0 * tmp2 + tmp3)
                + conv1 * (tmp1 - tmp3)
                + reaction * tmp2
                + tmp2 + 0.5 * YanenkoMixedDerivative(i, j, tnow - k / 2, matA);

            // Solve system along line y = j
            LUX.updateMatrix(AX, BX, CX, FX, BCL, BCR);

            // Update matrix at time
            updateColumn(j, matB, LUX.solve());

            // Early exercise for a row
            EarlyExerciseConstraintRow(matB, xMesh, j, yMesh[j]);
        }
    };
}

```

where the constraint function along a row is defined by:

```
auto EarlyExerciseConstraintRow = [&](NestedMatrix<double>& m,
                                         std::vector<double> x,
                                         std::size_t j, double yVal)
{ // Update a row of the matrix
    for (std::size_t i = 0; i < m.size1(); ++i)
    {
        m(i, j) = EarlyExerciseConstraint(m(i, j), x[i], yVal);
    }
};
```

The code for the second leg F2() and for checking constraints along a column is similar to the above.

23.7 APPENDICES

We discuss how heuristic techniques are used to find numeric boundary conditions.

23.7.1 Traditional Approach to Numerical Boundary Conditions

In the interest of completeness, we give some references to domain truncation and ad hoc boundary conditions. In general, there are no general a priori guidelines on computing the optimal truncated far field or how to determine what the appropriate boundary conditions should be.

In the case of a spread option, Chen (2017) proposes the following boundary conditions:

- BC1: Use the *analytic solution* as boundary condition (in the case of an exchange option, we could use the Margrabe formula, for example Margrabe (1978) and other examples in Haug (2007)). Of course, this tactic does not work if an analytic solution does not exist or if it cannot be found.
- BC2: *Pay-off boundary conditions*. In this case we use the far-field values in the pay-off function which will then become the boundary condition. This is a reasonable solution, although problems can arise when both state variables are near the far-field boundary.
- BC3: Let the PDE (23.2) be satisfied on the near-field and far-field boundaries. In special cases we may get lucky, and it may be possible to define Dirichlet boundary conditions. For example, the value of a put option is zero at the far field, and its value at the near field can be found in terms of the pay-off function. This is essentially the approach taken in Topper (2005).

- BC4: Using the *linearity boundary condition* $\frac{\partial^2 u}{\partial S^2}(S_{\max}, t) = 0$ in each space variable (including the removal of the mixed derivative term in (23.2)), we then get the first-order hyperbolic PDE (Tavella and Randall (2000), pp. 130–32):

$$\frac{\partial u}{\partial t} = (r - q_1)S_1 \frac{\partial u}{\partial S_1} + (r - q_2)S_2 \frac{\partial u}{\partial S_2} - ru.$$

This scheme is easily discretised using finite difference schemes. Another option is on an S_2 boundary, for example:

$$\frac{\partial u}{\partial t} = \frac{1}{2}\sigma_1^2 S_1^2 \frac{\partial^2 u}{\partial S_1^2} + \rho_1 \sigma_2 S_1 S_2 \frac{\partial^2 u}{\partial S_1 \partial S_2} + (r - q_1)S_1 \frac{\partial u}{\partial S_1} + (r - q_2)S_2 \frac{\partial u}{\partial S_2} - ru.$$

Here we are saying that there is no diffusion in the S_2 direction, while again it is possible to remove the mixed-derivative term. We can discretise this equation using one-sided difference operators in the terms containing S_2 and standard centred differencing for the terms containing S_1 .

Finally, at corner points (involving terms in S_2 and S_1) we have:

$$\frac{\partial u}{\partial t} = \rho \sigma_1 \sigma_2 S_1 S_2 \frac{\partial^2 u}{\partial S_1 \partial S_2} + (r - q_1)S_1 \frac{\partial u}{\partial S_1} + (r - q_2)S_2 \frac{\partial u}{\partial S_2} - ru.$$

In Tavella and Randall (2000), the problem of domain truncation is addressed and justified based on the heuristic that financial models usually contain a dominant diffusion process that attenuates disturbances from an imperfect boundary condition. Boundaries are replaced ‘far enough’ (whatever that means) from the region of interest, and hopefully these imperfections do not adversely affect the solution. Of course, this assumption must be backed up by extensive numerical testing.

In Tavella and Randall ((2000), pp. 120–30), the Cox–Ingessoll–Ross (CIR) PDE interest rate model is discussed. Special topics include truncation of the interest rate domain, linearity boundary condition and the Feller condition. We also address these issues from the perspective of domain transformation and Fichera theory in Chapter 25.

23.7.2 Top-Down Design of Monte Carlo Applications

In the interest of completeness, we take an example of a computer program to price one-factor plain options using the Monte Carlo method. It can be used as a template for modelling in a PDE framework. We discuss the design of this problem in detail in Duffy (2018), but for the moment we list the classes participating in the design (each one satisfies SRP). A decomposition is based on the author’s *Domain Architectures* approach (Duffy (2004)).

The *context diagram* is shown in Figure 23.2. It is a special case of the context diagram for applications that belong to the *Resource Allocation and Tracking* (RAT) category. In general, RAT systems track requests in time and space, and they produce the corresponding reports relating to the status of these requests. In this case the goal is to

compute the price of one-factor plain, barrier, lookback and Asian options using the Monte Carlo method. The request is processed in a series of steps to produce the final output by the modules in the UML component diagram:

- *Source*: The system containing the data relating to the request, for example market and model data. It contains data that is needed by other modules in Figure 23.2.
- *Sde*: The system that models stochastic differential equations (SDEs). In this case we model *geometric Brownian motion* (GBM) and its variants. In particular, we are interested in modelling the drift and diffusion of some underlying variables such as the stock price or interest rate, for example.
- *Fdm*: The family of finite difference schemes that approximate the SDEs in the *Sde* system. In this case we use one-step difference schemes to advance the approximate solution from one time level to the next time level until we reach the desired solution at expiration. The finite difference schemes require the services of a module *Rng* that computes random numbers.
- *Pricer*: This system contains classes to price one-factor options using the Monte Carlo Method. The classes process path information from the *Mediator*, and each class processes this path information in its own way. For example, for a plain option the pricer uses the path data at expiration, uses it to compute the pay-off, adds the result to a running total and then discounts the result to compute the option price.
- *MIS*: This is the statistics-gathering system that receives status information concerning the progress of computation. For example, this system could display how many paths have been processed at any given time.
- *Builder*: This system implements a configuration/creational pattern that creates and initialises the systems and their structural relationships. The newly created objects are encapsulated in a single tuple, which adds to the overall maintainability of the system.
- *Mediator*: This is the central coordinating entity that manages the data flow and control flow in the system. It contains the *state machine* that computes the paths of the SDE. It also informs the other systems of changes that they need to know about. It also plays the role of *client* in the *Builder* pattern.
- *Rng*: a system to generate random numbers. This feature is supported in C++11.

It is possible to use the model architecture in Figure 23.2 as the foundation for a software framework for multi-asset option pricing and indeed other kinds of applications.

23.8 SUMMARY AND CONCLUSIONS

In this chapter we discussed splitting methods for spread options, and we gave general recommendations. We used a number of techniques:

- Three schemes from Chapter 22.
- Handling mixed derivatives using Yanenko and Craig–Sneyd.

- Stress testing for large correlation.
- Software design.

The analysis in this chapter can be used as a template upon which we build other applications such as convertible bond, interest rate and stochastic uncertain volatility models. An interesting project would be to generalise the results of Chapters 16 and 17 for one-factor PDEs (sensitivities) to computing the greeks for two-asset options by solving the PDEs that they satisfy. Another project is to compare ADE and MOL with Splitting.

Chapter 24 discusses more detailed aspects of finite difference methods applied to PDEs.

CHAPTER 24

Asian (Average Value) Options

Whenever a theory appears to you as the only possible one, take this as a sign that you have neither understood the theory nor the problem which it was intended to solve.

Karl Popper

24.1 INTRODUCTION AND OBJECTIVES

In this chapter we discuss the pricing of Asian options using the finite difference method. An *Asian option* (or *average value option*) is one whose payoff is determined by an average (this can be an arithmetic or geometric average) price over some given period of time. Furthermore, there are two types of Asian option:

- *Fixed strike* (average rate): the averaging price is used instead of the underlying price.
- *Floating strike* (floating rate average strike): the averaging price is used instead of the strike price.

Furthermore, these can be of put or call type. Finally, we must consider how averages are computed; with *discrete sampling*, we take reliable data points, for example closing prices, while continuous sampling entails taking the integral of the underlying asset over the averaging period.

The focus in this chapter is on pricing Asian options using PDE/FDM methods. Asian PDEs are similar to multi-asset options in that they depend on two independent factors (the underlying and averaged value), but in contrast to multi-asset options, there is no diffusion term in the averaged value and hence the PDE contains a first-order convection equation only in this term. In particular, we need to prescribe one boundary condition (at most), and we use the methods from Chapter 11 to show how to properly discretise this PDE. Those readers with a background in applied mathematics and fluid mechanics will be familiar with this material, especially with the concept of *outflow boundary condition*.

The approach taken in this chapter is to apply the methods of previous chapters to the pricing of Asian options. One typical example is discussed in Wilmott, Lewis and Duffy (2014).

24.2 BACKGROUND AND PROBLEM STATEMENT

In this chapter we exclusively discuss PDE models for the pricing of Asian options. Thus, analytic and semi-analytic solutions and solutions based on the Monte Carlo method are not included. For a discussion of these methods, see Haug (2007), Glasserman (2004) and Duffy and Kienitz (2009). They are useful when we wish to compare the accuracy of finite difference schemes with known baseline solutions. Furthermore, we focus on continuously monitored arithmetic options.

The main goal of this chapter is to gain a deeper mathematical and numerical understanding of PDEs that model arithmetic Asian options, including applying the Fichera theory to determine the necessity of boundary conditions and to clear up some confusion caused by the fact that there is no diffusion term in one (the average) of the underlying variables. We then have to be careful when we introduce numerical methods for the resulting first-order hyperbolic PDE in the deterministic variable.

An *Asian option* is a contract that gives the holder the right to buy the underlying asset for an average price over some prescribed interval. This kind of option is popular in the currency and commodity markets and there are two ways of averaging the value:

- Arithmetic averaging
- Geometric averaging.

If the underlying asset is assumed to be lognormally distributed, then the geometric average of the asset will also be lognormally distributed. Arithmetic averaging takes the arithmetic average of the underlying asset. We must also determine when this sampling takes place:

- Discretely averaged samples
- Continuously averaged samples.

The formulae for the averaging scenarios are shown in Equations (24.1) and (24.2) below.

Arithmetic averaging:

$$\begin{aligned} \text{Continuous: } I &= \frac{1}{t} \int_0^t S(\tau) d\tau \\ \text{Discrete: } I &= \frac{1}{n} \sum_{j=1}^n S_j. \end{aligned} \tag{24.1}$$

Geometric averaging:

$$\begin{aligned} \text{Continuous: } & \int_0^t \log S(\tau) d\tau \\ \text{Discrete: } & \left(\prod_{j=1}^n S_j \right)^{\frac{1}{n}}. \end{aligned} \quad (24.2)$$

The corresponding *Terminal Boundary Value Problem* (TBVP) are given in Equations (24.3) and (24.4) as follows:

$$\frac{\partial V}{\partial t} + S \frac{\partial V}{\partial I} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0 \quad (24.3)$$

$$I \equiv \int_0^t S(\tau) d\tau.$$

$$\frac{\partial V}{\partial t} + \log S \frac{\partial V}{\partial I} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0 \quad (24.4)$$

$$I \equiv \int_0^t \log S(\tau) d\tau.$$

We note that these equations are of convection-diffusion type in the S variable, while in the I direction they are first-order hyperbolic equations, the latter PDEs having the form:

$$\frac{\partial V}{\partial t} + S \frac{\partial V}{\partial I} = 0 \quad (24.5)$$

$$\frac{\partial V}{\partial t} + \log S \frac{\partial V}{\partial I} = 0$$

and since we only have a derivative of, at most, order one in the I direction we can only accommodate one boundary condition (at most). Furthermore, centred difference schemes are not suitable (they are weakly stable or even unstable), and we must resort to one-sided (upwinded) schemes (See Chapters 14 and 15 of this book) that take the characteristic direction of the first-order equations into account.

24.2.1 Challenges

Most of the problems arise when discretising the convection PDE (24.5) and a possible lack of background regarding which discretisation to use and how to handle boundary conditions. A shortlist of ‘bad’ approaches is:

- Using the Alternating Direction Implicit (ADI) method in combination with five-point discretisation is somewhat of a sledgehammer approach. It necessitates solving band diagonal matrix systems.
- Most approaches tend to use domain truncation, which is mathematically and numerically less robust than domain transformation.

- Using higher-order (for example fourth-order) divided differences to approximate the convection term needs justification. Schemes need to be modified at the boundaries in some way by the introduction of cumbersome *ghost points*, for example, leading to even more ad hoc assumptions.
- In general, too many ad hoc assumptions concerning the behaviour of the solution on the boundaries of the domain. In particular, finding numerical boundary conditions is very much a case of trial and error. In particular, assuming that the second derivative of the option price is zero on the boundaries might work in practice, but it is probably an example of incorrect over-engineering.

These methods will probably work in practice after much tweaking and late-night debugging marathons. In a sense, Asian PDEs are almost one-dimensional, and using heavyweight machinery to solve them will probably introduce unnecessary numerical challenges. In general, a good starting point is to employ first-order upwinding schemes to approximate the convection term in combination with Fichera theory and energy inequalities to determine appropriate numerical boundary conditions (if any).

24.3 PROTOTYPE PDE MODEL

We now focus on the model PDE (based on Wilmott, Lewis and Duffy (2014)):

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} + (S - A) \frac{\partial V}{\partial A} - rV = 0, \quad (0 < S, A < \infty) \quad (24.6)$$

that is based on the following equations:

$$\begin{aligned} dS &= rSdt + \sigma SdW \\ dA &= (S - A)dt. \end{aligned} \quad (24.7)$$

Much has been published on this problem. In general, a plethora of finance difference schemes have been proposed to solve (24.6) using domain truncation and other heuristics (numerical boundary conditions). Our approach here is to transform the quarter-plane problem (24.6) to a new PDE in the unit square. To this end, we apply the *standard transformations* (as in Chapter 23, for example) to produce our ‘working PDE’. We write it as a forward PDE:

$$\begin{aligned} \frac{\partial V}{\partial t} &= \frac{1}{2}\sigma^2 x^2 \frac{\partial}{\partial x} \left((1-x)^2 \frac{\partial V}{\partial x} \right) + rx(1-x) \frac{\partial V}{\partial x} + \left(\frac{x}{1-x} - \frac{y}{1-y} \right) (1-y)^2 \frac{\partial V}{\partial y} - rV \\ &= \frac{1}{2}x^2(1-x)^2 \frac{\partial^2 V}{\partial x^2} + \{rx(1-x) - \sigma^2 x^2(1-x)\} \frac{\partial V}{\partial x} + \left(\frac{x(1-y)^2}{1-x} - y(1-y) \right) \frac{\partial V}{\partial y} - rV. \end{aligned} \quad (24.8)$$

We now investigate this PDE from the perspective of the Fichera theory. Recalling the notation from Chapters 8 and 9, we see that the coefficients of the PDE are:

$$\begin{aligned} a_{11} &= \frac{1}{2}\sigma^2x^2(1-x)^2 \\ a_{22} &= a_{21} = a_{12} = 0 \\ b_1 &= rx(1-x) - \sigma^2x^2(1-x) = b_1(x) \\ b_2 &= \frac{x(1-y)^2}{1-x} - y(1-y) = b_2(x,y) \end{aligned} \tag{24.9}$$

and from the calculations:

$$\begin{aligned} b_1(0) &= b_1(1) = 0 \\ b_2(x, 0) &= \frac{x}{1-x} > 0, \quad b_2(x, 1) = 0 \\ \text{For } y = 1, b_F &= -b_2(x, 1) = 0 \end{aligned} \tag{24.10}$$

we conclude that the Fichera function is zero on the boundary of the unit square. Hence no boundary conditions are needed, and thus PDE (24.8) is satisfied on the boundaries. For example, at $x = 0$ we get:

$$\frac{\partial V}{\partial t} = -rV - y(1-y)\frac{\partial V}{\partial y}. \tag{24.11}$$

Our conclusions differs from the conclusions in Meyer (2015) that claims that boundary conditions are needed at the far field $A = A_{\max}$ based on the application of the Fichera theory. The paradox in our opinion is: a PDE on a quarter plane (with no boundary conditions) is mapped to a PDE on a truncated domain that needs a boundary condition according to the Fichera theory! This way of thinking underlies many numerical methods that use domain truncation. In Wilmott, Lewis, and Duffy (2014) we do not encounter these issues. The main concern is whether it is allowed to apply the Fichera theory to this truncated PDE. This is an open question in our opinion. Of course, we need to prescribe *numerical boundary conditions*, but these can be based on the pay-off as we discussed in Chapter 23.

24.3.1 Similarity Reduction

In some cases it is possible to simplify the PDEs that we have already discussed in this chapter.

We examine the PDE (24.3), and we consider the so-called *similarity reduction* technique by defining a new variable R as $R = I/S$ and the function H by (Wilmott (2006)):

$$V(S, R, t) = SH(R, t).$$

We can check that the function H satisfies the following PDE:

$$-\frac{\partial H}{\partial t} + \frac{1}{2}\sigma^2R^2\frac{\partial^2 H}{\partial R^2} + (1 - rR)\frac{\partial H}{\partial R} = 0. \tag{24.12}$$

The initial/terminal condition for E is now:

$$H(R, 0) = \frac{V(S, R, 0)}{S(0)} = g(S(0), A(0)) \quad (24.13)$$

where g is some function.

Now for the tricky part. At large values of R , the value of H is zero:

$$\lim_{R \rightarrow \infty} H(R, t) = 0 \quad (24.14)$$

while when $R = 0$ the PDE degenerates into the first-order hyperbolic PDE:

$$-\frac{\partial H}{\partial t} + \frac{\partial H}{\partial R} = 0. \quad (24.15)$$

There are many ways to solve system (24.12) to (24.15), as we have already discussed in this book. There are no big surprises with the possible exception of (24.15), which is a first-order hyperbolic PDE.

24.4 THE MANY WAYS TO HANDLE THE CONVECTIVE TERM

We take the following first-order hyperbolic PDE as the prototype model for the convective terms in PDEs (24.3), (24.6) and (24.15), respectively (but with the direction of flow reversed (this is a minor point)):

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0, \quad 0 < x < \infty, a > 0 \quad (24.16)$$

with *inlet boundary* condition.

$$u(0, t) = g(t). \quad (24.17)$$

Let us also assume that the *exit boundary* for problem (24.16) is $x = L$.

Studying and understanding this problem will help when examining Equation (24.8) and avoid many of the mistakes that have been made in the past. We remark that it is also possible to examine schemes for (24.16) in the case $a < 0$, but we then will have *downwinding* instead of *upwinding*.

24.4.1 Method of Lines (MOL)

We examine the three-point *semi-discretisation* of (24.16), and it is probably the first scheme that quants tend to use based on the fact that it is used for diffusion equations (J is the number of subdivisions of the interval $(0, L)$):

$$\frac{du_j}{dt} + a \left(\frac{u_{j+1} - u_{j-1}}{2h} \right) = 0, \quad 1 \leq j \leq J-1, \quad h = \frac{L}{J}. \quad (24.18)$$

This innocuous-looking scheme is formally second-order accurate, but since it uses three points, we see that things become nasty at $x = L$; in short, we do not have enough boundary information. A workaround is to replace this scheme by a two-point first-order accurate upwind approximation:

$$\frac{du_J}{dt} + a \left(\frac{u_J - u_{J-1}}{h} \right) = 0. \quad (24.19)$$

A consequence of this tactic is that this approximation creates *spurious reflections* (Vichnevetsky and Bowles (1982)). What can be done? A scheme with third-order truncation error is:

$$\frac{du_J}{dt} + a \left(\frac{3u_J - 4u_{J-1} + u_{J-2}}{2h} \right) = 0. \quad (24.20)$$

This is an improvement on (24.19) because the amplitude of the reflected solution after passage through the exit boundary $x = L$ is second order in the step length h for (24.19), while in the case of (24.20) it is third order in h .

An important remark is that scheme (24.18) is globally second-order accurate, even though it is only first-order accurate on the boundary $x = L$. This is due to an important result in Gustafsson (1975). This is good to know.

For completeness, we list three popular semi-discrete schemes:

2-point implicit:

$$\frac{1}{2} \left(\frac{du_j}{dt} + \frac{du_{j+1}}{dt} \right) + a \left(\frac{u_{j+1} - u_j}{h} \right) = 0$$

General 3-point family:

$$\frac{\beta}{2} \frac{du_{j-1}}{dt} + (1 - \beta) \frac{du_j}{dt} + \frac{\beta}{2} \frac{du_{j+1}}{dt} + a \left(\frac{u_{j+1} - u_{j-1}}{2h} \right) = 0 \quad (24.21)$$

Linear finite element (FEM): ($\beta = 1/3$)

$$\frac{1}{6} \frac{du_{j-1}}{dt} + \frac{4}{6} \frac{du_j}{dt} + \frac{1}{6} \frac{du_{j+1}}{dt} + a \left(\frac{u_{j+1} - u_{j-1}}{2h} \right) = 0.$$

In the last case we see the emergence of the well-known *mass matrix* M when we assemble the system of equations, that is

$$M = \frac{1}{6} \begin{pmatrix} 4 & 1 & 0 \\ 1 & \ddots & \ddots \\ & \ddots & \ddots & 1 \\ 0 & & & 4 \end{pmatrix}.$$

24.4.2 Other Schemes

We have discussed a number of stable (and not so stable) schemes for (24.16) in Chapter 14, Sections 14.8.3 and 14.8.4:

- FTCS
- BTBS
- FTBS
- Leapfrog
- Thomée (box) scheme.

Finally, the Lax–Wendroff scheme is popular as a means of adding an *artificial diffusion* to a first-order PDE (Lax and Wendroff (1960)):

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{2h} a \left(u_{j+1}^n - u_{j-1}^n \right) + \frac{\Delta t^2}{2h^2} a^2 \left(u_{j+1}^n - 2u_j^n + u_{j-1}^n \right). \quad (24.22)$$

This scheme is consistent and stable if the famous CFL (Courant–Friedrichs–Lewy) condition is satisfied (Thomas (1995)):

$$\frac{|a|\Delta t}{h} \leq 1. \quad (24.23)$$

In other words, we have stabilised the unstable scheme (14.50) (Chapter 14) by adding an artificial diffusion term to it. Furthermore, the stability condition (24.23) is independent of the sign of a , making the method applicable to a large class of problems. Some general remarks on the possible applicability of scheme (24.22) to the Cheyette model are to be found in Kohl-Landgraf (2007), but no definite advice is given. The Lax–Wendroff scheme is second order accurate $O(h^2 + \Delta t^2)$.

It is very easy to motivate the Lax–Wendroff scheme as follows: first, we differentiate (24.16) with respect to t to get:

$$\begin{aligned} \frac{\partial u}{\partial t} &= -a \frac{\partial u}{\partial x} \\ \frac{\partial^2 u}{\partial t^2} &= \frac{\partial}{\partial t} \left(\frac{\partial u}{\partial t} \right) = a^2 \frac{\partial^2 u}{\partial x^2}. \end{aligned}$$

Then we perform a Taylor expansion to give:

$$\begin{aligned} u_j^{n+1} &= u_j^n + \Delta t \frac{\partial u_j^n}{\partial t} + \frac{1}{2} \Delta t^2 \frac{\partial^2 u_j^n}{\partial t^2} + O(\Delta t^3) \\ &= u_j^n - \Delta t a \frac{\partial u_j^n}{\partial x} + \frac{1}{2} \Delta t^2 a^2 \frac{\partial^2 u_j^n}{\partial x^2} + O(\Delta t^3). \end{aligned}$$

We then approximate the first order and second order in x derivatives, and we end up with scheme (24.22).

24.4.3 A Stable Monotone Upwind Scheme

We consider a novel one-sided scheme that the author developed to approximate (24.16), namely:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + a \frac{u_j^{n+1} - u_{j-1}^n}{h} = 0 \quad (a > 0). \quad (24.24)$$

The rationale is that ‘boundary information, coming from the left’ is computed at time level n (where it is known) and not at time level $n + 1$ where it may be more difficult to compute. We can see that scheme (24.24) is *monotone* by writing it in the form:

$$u_j^{n+1} = (u_j^n + \lambda u_{j-1}^n)/1 + \lambda, \quad \left(\lambda = \frac{a\Delta t}{h} \right). \quad (24.25)$$

By a *monotone property* we mean in rough terms (Laney (1998)) that if the initial condition of a numerical condition is non-negative, then the solution at a later time is also non-negative. More generally:

$$u_j^0 \geq v_j^0 \quad \forall j \Rightarrow u_j^n \geq v_j^n \quad \forall j, n. \quad (24.26)$$

How ‘good’ is (24.24)? It is stable because its amplification factor (see formula (14.48) in Chapter 14) is:

$$\gamma = \frac{1 + \lambda e^{i\alpha h}}{1 + \lambda}, \text{ then } |\gamma| \leq 1 \quad (24.27)$$

and has truncation error $O(h + \Delta t + \Delta t/h)$, thus making its conditionally consistent with (24.16). It is also the scheme that we use in Wilmott, Lewis and Duffy (2014) and Duffy (2018) to approximate the convection term.

24.5 ADE FOR ASIAN OPTIONS

In this section we describe how we applied the ADE method to the solution of (24.8). A more complete description is given in Wilmott, Lewis and Duffy (2014), and a C++ implementation is given in Duffy (2018). We get the same results as Alan Lewis, who used Mathematica’s *NDSolve*. *NDSolve* is an implementation of Method of Lines (MOL), which is a semi-discretisation of the space variables to produce a system of ODEs, which is then solved by an adaptive time-stepping.

The ADE algorithm used the Barakat and Clark scheme for the diffusion term in x and both Roberts–Weiss and Towler–Yang for the convection term in x while we used the scheme (24.24) for the convection term in y . Of course, the convection coefficient may change sign, in which case we deploy the scheme:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + a \frac{u_{j+1}^n - u_j^{n+1}}{h} = 0, \quad a < 0. \quad (24.28)$$

In the C++ code, either (24.24) or (24.28) will be executed, depending on the sign of the convection coefficient.

We also tested the accuracy of the Saul'yev scheme by examining the numerical values in Table 2 of Wilmott, Lewis and Duffy (2014) and attempting to reproduce them. We note that the results in that table were computed using the *Mathematica NDSolve* package as well as our C++ code. In general, second- (or higher-) order divided differencing is applied to the spatial derivatives resulting in a system of ordinary differential equations (ODEs), which is then solved in time by an adaptive high-order integrator, such as the *Bulirsch–Stoer* method, for example. In Wilmott, Lewis and Duffy (2014), the mesh sizes $NX = NY = 250$ are specified. In our tests on ADE, we take $NX = NY = 500$, $NT = 2000$, which makes a direct comparison between the methods somewhat difficult in a sense. However, we can get three digits accuracy in the main:

```

NX, NY, NT (put Saul'yev): 500,500,2000
T: 0.25, K: 60, price at hot spot: 0.000439322
T: 1, K: 60, price at hot spot: 0.169892
T: 3, K: 60, price at hot spot: 0.968049
T: 10, K: 60, price at hot spot: 1.94742

T: 0.25, K: 80, price at hot spot: 0.0431205
T: 1, K: 80, price at hot spot: 0.850115
T: 3, K: 80, price at hot spot: 2.3882
T: 10, K: 80, price at hot spot: 3.57465

T: 0.25, K: 100, price at hot spot: 2.83833
T: 1, K: 100, price at hot spot: 4.6868
T: 3, K: 100, price at hot spot: 6.05442
T: 10, K: 100, price at hot spot: 6.06422

T: 0.25, K: 120, price at hot spot: 18.5937
T: 1, K: 120, price at hot spot: 16.2815
T: 3, K: 120, price at hot spot: 13.7188
T: 10, K: 120, price at hot spot: 9.68991

T: 0.25, K: 140, price at hot spot: 38.2958
T: 1, K: 140, price at hot spot: 33.5593
T: 3, K: 140, price at hot spot: 25.4875
T: 10, K: 140, price at hot spot: 14.6057

```

24.6 ADI FOR ASIAN OPTIONS

ADI is the author's least favourite method for two-factor problems in general and for Asian-style PDEs in particular. Some of the objective and subjective reasons for not using it are:

- It originated in the 1960s as a method for solving two-dimensional diffusion problems in the oil and gas industry. In its original form, it was not easy to apply to three-dimensional problems, convection-diffusion equations and problems with mixed derivatives, among others.

- The literature on ADI is scant and sometimes misleading; for example, in Press, Teukolsky, Vetterling and Flannery (2002), p. 859, it is claimed that ADI for convection-diffusion PDE can be implemented by using an explicit scheme for the convection. This statement is ambiguous and in some cases wrong. For example, we can expect numerical stability problems if the FTCS scheme (14.50) (Chapter 14) is unwittingly *embedded* as a component in the main ADI scheme.
- Much of the literature tends to ignore important issues, such as splitting and discretisation errors, and the literature tends to use extensive numerical testing as a surrogate. Furthermore, some ADI schemes are in fact (operator) splitting schemes under a different name. Many ADI schemes are essentially first-order accurate.
- ADI for spread options (for example) produces oscillations in the solution for discontinuous payoffs.
- Reader background: many quants do not have a background in applied mathematics or fluid dynamics, areas in which convection-diffusion equations are common. They tend to be more familiar with diffusion processes, but the numerical methods for these problems do not always map well to convection/drift terms.
- The inventors of ADI probably did not envisage it being used for Asian-style PDEs. In this sense the Strang splitting method is probably more appropriate.

For completeness and backwards compatibility, we discuss how to use ADI for Asian option pricing.

We rewrite the PDE for an Asian option (24.3) and (24.4) in the slightly more general form:

$$-c \frac{\partial V}{\partial t} + b \frac{\partial^2 V}{\partial S^2} + \sigma \frac{\partial V}{\partial S} + \alpha \frac{\partial V}{\partial I} - bV = f. \quad (24.29)$$

We apply ADI by discretising Equation (24.29) in two steps. We first proceed from time level n to $n + \frac{1}{2}$ by using the implicit exponentially fitted scheme in S and explicit centred differencing in the I direction:

$$\begin{aligned} & -c_{ij}^{n+\frac{1}{2}} \frac{V_{ij}^{n+\frac{1}{2}} - V_{ij}^n}{\frac{1}{2}k} + \sigma_{ij}^{n+\frac{1}{2}} \frac{V_{i+1j}^{n+\frac{1}{2}} - 2V_{ij}^{n+\frac{1}{2}} + V_{i-1j}^{n+\frac{1}{2}}}{h^2} \\ & + \alpha_{ij}^{n+\frac{1}{2}} \frac{V_{i+1j}^{n+\frac{1}{2}} - V_{i-1j}^{n+\frac{1}{2}}}{2h} + \alpha_{ij}^{n+\frac{1}{2}} \frac{V_{ij+1}^n - V_{ij-1}^n}{2m} - b_{ij}^{n+\frac{1}{2}} V_{ij}^{n+\frac{1}{2}} = f_{ij}^{n+\frac{1}{2}} \end{aligned} \quad (24.30)$$

where k , h and m are the mesh sizes in the time, S and I variables. This equation can also be written in the form:

$$A_{ij}^{n+\frac{1}{2}} V_{i-1j}^{n+\frac{1}{2}} + B_{ij}^{n+\frac{1}{2}} V_{ij}^{n+\frac{1}{2}} + C_{ij}^{n+\frac{1}{2}} V_{i+1j}^{n+\frac{1}{2}} = f_{ij}^{n+\frac{1}{2}} \quad (24.31)$$

which we can solve, for example, by the use of LU decomposition or by the Double Sweep method.

The next step is to obtain the solution at level $n + 1$ in terms of the solution at level $n + \frac{1}{2}$ by using an explicit scheme in S and the implicit method in the I direction:

$$\begin{aligned} & -c_{ij}^{n+1} \frac{V_{ij}^{n+1} - V_{ij}^{n+\frac{1}{2}}}{\frac{1}{2}k} + \sigma_{1+ij}^{n+1} \frac{V_{i+1j}^{n+\frac{1}{2}} - 2V_{ij}^{n+\frac{1}{2}} + V_{i-1j}^{n+\frac{1}{2}}}{h^2} \\ & + a_{ij}^{n+1} \frac{V_{i+1j}^{n+\frac{1}{2}} - V_{i-1j}^{n+\frac{1}{2}}}{2h} + a_{ij+1}^{n+1} \frac{V_{ij+1}^{n+1} - V_{ij-1}^{n+1}}{2m} - b_{ij}^{n+1} V_{ij}^{n+\frac{1}{2}} = f_{ij}^{n+1}. \end{aligned} \quad (24.32)$$

Again, we can write (24.32) as a tridiagonal system:

$$A_{ij}^{n+1} + V_{ij-1}^{n+1} + B_{ij}^{n+1} V_{ij}^{n+1} + C_{ij}^{n+1} + V_{ij+1}^{n+1} = f_{ij}^{n+1}. \quad (24.33)$$

In Equations (24.30) and (24.32) we have approximated the first-order derivative with respect to the independent variable I by using centred differences. A better approach is to use a one-sided scheme depending on the sign of the coefficient α appearing in Equation (24.29):

$$\begin{aligned} \text{for } \alpha > 0, \quad \alpha \frac{\partial V}{\partial I} & \sim \alpha_{ij+\frac{1}{2}} \frac{V_{ij+1}^n - V_{ij}^n}{m} \\ \text{for } \alpha < 0, \quad \alpha \frac{\partial V}{\partial I} & \sim \alpha_{ij-\frac{1}{2}} \frac{V_{ij}^n - V_{ij-1}^n}{m}. \end{aligned} \quad (24.34)$$

24.6.1 Modern ADI Variations

We now discuss an adaption of the ADI method from the previous section. We produce a scheme of accuracy $O(\Delta x^6 + \Delta y^6 + \Delta t^2)$. This could be an interesting research project. The scheme is called the *Combined Compact Difference* (CCD) ADI method, and it is a combination of the D'Yakonov scheme (see Equation (18.33) in Chapter 18) (Sun and Li (2014)) in combination with the CCD method for one-dimensional boundary value problems as described in Chu and Fan (1998).

Let us begin with the two-dimensional, constant-coefficient convection-diffusion PDE:

$$\begin{aligned} \frac{\partial u}{\partial t} &= (\mathcal{L}_x + \mathcal{L}_y)u + f(x, y, t) \\ \mathcal{L}_x &= a \frac{\partial^2}{\partial x^2} - p \frac{\partial}{\partial x} \\ \mathcal{L}_y &= b \frac{\partial^2}{\partial y^2} - q \frac{\partial}{\partial x}. \end{aligned} \quad (24.35)$$

Discretising (24.35) in time using the Crank–Nicolson around $t = t_n + \Delta t/2$, collecting like terms, factoring and using the fact that $\frac{\Delta t^2}{2} \mathcal{L}_x \mathcal{L}_y (u^{n+1} - u^n) = 0(\Delta t)$ gives us:

$$(1 - \frac{\Delta t}{2} \mathcal{L}_x)(1 - \frac{\Delta t}{2} \mathcal{L}_y)u^{n+1} = (1 + \frac{\Delta t}{2} \mathcal{L}_x)(1 + \frac{\Delta t}{2} \mathcal{L}_y)u^n + \Delta t f^{n+1/2} + O(\Delta t^2). \quad (24.36)$$

We introduce the intermediate term u^* and then we can write (24.36) in the D'Yakonov ADI-like scheme at mesh points in the x and y directions:

$$\begin{aligned} \left(1 - \frac{\Delta t}{2} \mathcal{L}_x\right) u_{j,k}^* &= \left(1 + \frac{\Delta t}{2} \mathcal{L}_y\right) \left(1 + \frac{\Delta t}{2} \mathcal{L}_y\right) u_{j,k}^n + \Delta t f_{j,k} \\ \left(1 - \frac{\Delta t}{2} \mathcal{L}_y\right) u_{j,k}^{n+1} &= u_{j,k}^*. \end{aligned} \quad (24.37)$$

The difference between (18.33) (Chapter 18) and (24.37) is that the former method uses discrete operators, while (24.37) uses continuous operators. It is now the time when the CCD method is applied to (24.37). For the details, see Chu and Fan (1998). This would be an interesting research project.

24.7 SUMMARY AND CONCLUSIONS

In this chapter we examined several finite difference schemes to price Asian options using a PDE approach. The two underlying variables are the stock S and an average A of the stock. The average has no diffusion term, and this fact complicates the numerical analysis somewhat. We have addressed this issue, and we have proposed a number of solutions.

A nice project could be the application of Strang splitting in which leg 1 of Equation (24.8) uses Crank–Nicolson and leg 2 implements the Lax–Wendroff method (24.22). Both legs are second-order accurate, but we need to investigate what the order of the splitting error is. Finally, a brainstorm is to *regularise* the first-order PDE in A to a singularly perturbed PDE of the form:

$$\frac{\partial u}{\partial t} = a \frac{\partial u}{\partial A} + \varepsilon \frac{\partial^2 u}{\partial A^2}, \quad 0 < A < \infty \quad 0 < \varepsilon \ll 1. \quad (24.38)$$

This PDE can be seen as the continuous analogue of the Lax–Wendroff method. We can investigate this PDE using the techniques that the author has developed in this book. In particular, we need to investigate conditions on ε to ensure a good approximation in the limiting case $\varepsilon \rightarrow 0$. This is thus a *singular perturbation problem* and could form the basis for a research project, for it is an open question at the moment of writing.

CHAPTER 25

Interest Rate Models

I would rather have questions that can't be answered than answers that can't be questioned.

Richard Feynman

25.1 INTRODUCTION AND OBJECTIVES

In this chapter we formulate interest-rate problems as PDEs, and we solve them numerically using the finite difference schemes that we have introduced in this book. Historically (and up to the present day), these problems were analysed by (quasi) analytical, Monte Carlo and lattice (binomial, trinomial) methods. A discussion of these methods is outside the scope of this book, and instead we focus on PDE/FDM methods for such problems. It would seem that current PDE/FDM methods used for interest rate problems tend to produce solutions centred on the Crank–Nicolson and ADI methods in combination with domain truncation and linearity boundary conditions. The consequences and results are not always optimal. Furthermore, the discovery of boundary conditions for these problems tends to be more of an art than hard science.

Our contribution in this chapter is to apply PDE/FDM methods to interest rate problems using mathematical and numerical principles. Some novel features that we introduce are:

- Mathematically precise PDEs describing interest rate problems, notably the *Cox–Ingersoll–Ross* (CIR) model. We use domain transformation, allowing us to discover appropriate boundary conditions for these problems. It is also possible to transform these PDEs to conservative (self-adjoint) form.
- A choice of efficient and accurate finite difference schemes for the CIR model. We also discuss the two-factor HW model and related schemes.
- Well-posedness of the initial boundary value problem for the CIR model. We show that this is closely related to the celebrated *Feller condition*, and we show using integration by parts (Green's formula) how this condition can be recovered from first principles.

There is considerable overlap between models and methods to solve interest rate and equity problems. For example, bond options, caps, floors, single caplets and floorlets, and other related interest rate products can all be modelled as PDEs. One major difference, however, is that the interest rate is now a stochastic or independent variable.

25.2 MAIN USE CASES

The main topics that we discuss in this chapter are:

- U1: Introduction to the Cox–Ingersoll–Ross (CIR) model; SDE form.
- U2: Well-posedness of the CIR PDE; deriving the Feller condition from ‘first principles’.
- U3: Domain transformation of CIR PDE. Energy estimates and Fichera theory.
- U4: Finite difference approximation (ADE) of CIR PDE.

The following topics are outside the current scope:

- CIR sensitivities (for example, duration and convexity) using the CSE approach from Chapter 17. This is an open research problem; the presence of the Feller condition can lead to non-uniqueness of solutions (see Robinson (2019), Lewis (2016)).
- CIR sensitivities using simulation and Complex Step Method (CSM) (Kunita (1990), Rasmussen (2016)).
- A new generation of numerical methods for stochastic volatility models: for example, the Heston model. Some methods could be a) reduction to canonical form, b) domain transformations, and c) the holy grail of monotone schemes.
- Rough volatility models.
- A thorough analysis of PDE methods for other kinds of interest rate models: for example, one-factor and two-factor Hull–White models.

25.3 THE CIR MODEL

The *Cox–Ingersoll–Ross* (CIR) model (also known as the *CIR process*) describes the evolution of interest rates. It is a one-factor short-rate model describing interest rate movements that are driven by one source of market risk. The corresponding SDE is defined by:

$$dr(t) = \kappa(\theta - r(t))dt + \sigma\sqrt{r(t)}dW(t) \quad (25.1)$$

where:

$r(t)$ = stochastic interest rate

κ = determines the speed of adjustment ($\kappa > 0$)

θ = central location (long-term value) of $r(t)$ ($\theta > 0$)

σ = volatility of the short rate $r(t)$

$W(t)$ = Wiener process.

This is a *mean-reversion* model. The drift factor $\kappa(\theta - r(t))$ ensures mean reversion of the interest rate to the long-term value θ . The speed of adjustment is governed by the parameter κ . The standard deviation factor $\sigma\sqrt{r(t)}$ avoids the possibility of negative interest rates for all positive values of κ and θ . Furthermore, zero interest rates are not possible if the Feller condition is satisfied:

$$\kappa\theta \geq \sigma^2/2. \quad (25.2)$$

When the rate approaches zero, the evolution becomes dominated by the drift factor, pushing the rate upwards to equilibrium.

Much of the literature uses the following equivalent form instead of Equation (25.1):

$$dr(t) = (a - br(t))dt + \sigma\sqrt{r(t)}dW. \quad (25.3)$$

In computational examples we may need to switch between the various parameter regimes when working with these SDEs. It can be quite annoying. Ideally, a single standard would be better.

25.3.1 Analytic Solutions

A *discount (zero coupon) bond* is one that gives the holder a single unit cash flow (for example, one dollar) at maturity with no intermediate cash flows. We define $P(t, s)$, or more strictly $P(r, t, s)$, to be the price at time t of a discount bond that matures at time s with $t \leq s$ and $P(s, s) = 1$. In Cox, Ingersoll and Ross (1985), the analytical price of a zero coupon bond in the CIR model at time t and maturity T is given by:

$$P(t, T) = A(t, T)e^{-r(t)B(t, T)} \quad (25.4)$$

where:

$$A(t, T) = \left(\frac{2he^{(h+k)(T-t)/2}}{2h + (h+k)(e^{h(T-t)} - 1)} \right)^{2k\theta/\sigma^2}$$

$$B(t, T) = \frac{2(e^{h(T-t)} - 1)}{2h + (h+k)(e^{h(T-t)} - 1)}$$

$$h = \sqrt{k^2 + 2\sigma^2}.$$

The C++ code that implements this formula is:

```
double priceCIR(double r, double t, double T, double kappa,
                 double theta, double sig)
{ // Exact CIR price of a zero-coupon bond P(t, T), t < T

    // Feller condition satisfied?
    if (kappa*theta >= sig * sig / 2)
        std::cout << "Feller condition satisfied\n";
```

```

    else
        std::cout << "Feller condition not satisfied "
        << kappa*theta << ", " << sig*sig/2 << '\n';

    double h = std::sqrt(kappa*kappa + 2.0*sig*sig);

    double expm1 = std::exp(h*(T-t)) - 1.0;
    double hpk = h + kappa;

    double B = (2.0 * expm1) / (2.0*h + hpk*expm1);

    double factor = 2.0 * h * std::exp(0.5*hpk*(T-t))
    / (hpk*expm1 + 2.0*h);
    double A = std::pow(factor, 2.0*kappa*theta / (sig*sig));

    return A * std::exp(-B * r);
}

```

A test case for comparison purposes is:

```

// Discount bond
// dr = (a - br)dt + sig r^1/2 dW
double a = 0.04; double b = 0.8; double sig = 0.275;
double r = 0.08; // Price computed for this value
double T = 0.25;

double kappa = b; double theta = a / b; double t = 0.0;

std::cout << std::setprecision(12) << priceCIR(r, t, T, kappa, theta, sig);

```

The answer is 0.9809004 and the Feller condition (25.2) is satisfied.

The CIR valuation framework can be applied to other securities: for example, options on bonds $C(r, t, T; s, K)$. We denote by:

$$C(r, 0, T; s, K) = \max(P(r, T, s) - K, 0)$$

the pay-off of a call option on a discount bond of maturity date s , exercise price K and expiration date T , where $s \geq T \geq t$. The analytic solution involves the non-central chi-squared (cumulative) density function (which is supported in the Boost C++ mathematical libraries). The formula is given in Cox, Ingersoll and Ross (1985), and we do not repeat it here. However, we do show how the formula is implemented in C++:

```

double priceEuropeanCall(double r, double t, double T, double kappa,
double theta, double sig, double s, double K)
{ // Exact CIR price European call option on a zero-coupon bond C(t, T; s, K)

    // Feller condition satisfied?

```

```

if (kappa*theta >= sig * sig / 2)
    std::cout << "Feller condition satisfied\n";
else
    std::cout << "Feller condition not satisfied\n";

// Compute discount bond
double h = std::sqrt(kappa*kappa + 2.0*sig*sig);

double expm1 = std::exp(h*(T - t)) - 1.0;
double expm2 = std::exp(h*(s - T)) - 1.0;
double hpk = h + kappa;

double B = (2.0 * expm2) / (2.0*h + hpk * expm2);

double factor = 2.0 * h * std::exp(0.5*hpk*(s - T))
    / (hpk*expm2 + 2.0*h);
double A = std::pow(factor, 2.0*kappa*theta / (sig*sig));

// Compute European call, keep notation as in Cox et al. (1985)
// Assume market risk parameter lambda = 0; C(t,T; s,K)
double phi = (2.0*h) / (sig*sig*expm1);
double psi = (kappa + h)/(sig*sig);

double rStar = std::log(A / K) / B;

// Build up the components of the call price
double p1 = priceCIR(r, t, s, kappa, theta, sig);
double p2 = priceCIR(r, t, T, kappa, theta, sig);
double dof1 = 4.0*kappa*theta / (sig*sig);
double lambda1 = 2.0*phi*phi*r*std::exp(h*(T - t)) / (phi + psi + B);
double lambda2 = 2.0*phi*phi*r*std::exp(h*(T - t)) / (phi + psi);
double x1 = 2.0*rStar*(phi + psi + B);
double x2 = 2.0*rStar*(phi + psi);

boost::math::non_central_chi_squared_distribution<double>
    nc1(dof1, lambda1);
boost::math::non_central_chi_squared_distribution<double>
    nc2(dof1, lambda2);

return p1 * boost::math::cdf(nc1,x1) - K * p2*boost::math::cdf(nc2,x2);
}

```

We use these pricing algorithms to compare with various finite difference schemes for the same problems. A test case is:

```

// European call option on a bond
double sig = 0.1; double r = 0.05; // Price computed for this value

```

```

double T = 1.0; double s = 5.0; double K = 0.67;

double kappa = 0.15; double theta = 0.05; double t = 0.0;
double result = priceEuropeanCall(r, t, T, kappa, theta, sig, s, K);
std::cout << "European call: " << std::setprecision(12) << result;

```

The answer is 0.14618 and the Feller condition (25.2) is satisfied.

25.3.2 Initial Boundary Value Problem

The price of a *zero coupon bond* $B(r, t)$ (and other similar securities) is given by the PDE:

$$\frac{\partial B}{\partial t} = \frac{1}{2}\sigma^2 r \frac{\partial^2 B}{\partial r^2} + (a - br)\frac{\partial B}{\partial r} - rB = 0, \quad 0 < r < \infty, \quad t > 0. \quad (25.5)$$

This is a convection-diffusion-reaction equation. We augment it by the heuristic (ad hoc) near-field and far-field boundary conditions when the Feller condition (25.2) holds:

$$\begin{aligned} B(\infty, t) &= 0, t > 0 \\ -\frac{\partial B}{\partial t}(0, t) + a\frac{\partial B}{\partial r}(0, t) &= 0, t > 0. \end{aligned} \quad (25.6)$$

We have already given an introduction to this PDE in Section 11.6.1 (Chapter 11). We can write Equation (25.5) in conservative form:

$$\begin{aligned} \frac{\partial B}{\partial t} &= \alpha(r, t)\frac{\partial}{\partial r} \left(\beta(r, t)\frac{\partial B}{\partial r} \right) + rB \\ \alpha(r, t) &= -\frac{1}{2}\sigma^2 r^{1-\frac{2br}{\sigma^2}}, \text{ and } \beta(r, t) = r^{\frac{2a}{\sigma^2}} e^{-\frac{2br}{\sigma^2}}. \end{aligned} \quad (25.7)$$

This PDE could be used instead of PDE (25.5) as input to a numerical analysis using finite element or finite volume methods (Achdou and Pironneau (2005), Zhang and Yang (2017)). We can choose between domain truncation and domain transformation. Extension of the fitted volume method and Crank–Nicolson to pricing American bond options is discussed in Gan and Xu (2020). Each approach has its advantages and disadvantages. This could form the basis for a research project to compare the different methods.

In the sequel we focus on the domain-transformed version of PDE (25.5).

25.4 WELL-POSEDNESS OF THE CIRPDE MODEL

In this section we prove that the solution of the CIR PDE problem is unique in the case where the Feller condition (25.2) is satisfied. In particular, we compute a priori estimates to bound the solution in a given norm in terms of the data. We shall see that

no boundary conditions are needed precisely when (25.2) is satisfied! In this way we see that the solution is unique. Another way of saying this is that zero data leads to a zero solution.

We repeat the PDE:

$$\frac{\partial B}{\partial t} = \frac{1}{2}\sigma^2 r \frac{\partial^2 B}{\partial r^2} + (a - br) \frac{\partial B}{\partial r} - rB \quad 0 < r < \infty, t > 0. \quad (25.8)$$

We can remove the reaction term completely in Equation (25.8) by a change of dependent variable:

$$u(r, t) = e^{rt}B. \quad (25.9)$$

Then this new variable satisfies (25.8) but without the reaction term. This modified PDE (after domain transformation) will be our starting point when computing energy estimates. We then write the PDE for $u(r, t)$ in compact form for convenience:

$$\frac{\partial u}{\partial t} = Lu \equiv a(y) \frac{\partial}{\partial y} (b(y) \frac{\partial u}{\partial y}) + (c(y) + d(y)) \frac{\partial u}{\partial y} \quad (25.10)$$

where:

$$a(y) = \frac{1}{2}\sigma^2 y(1-y)$$

$$b(y) = (1-y)^2$$

$$c(y) = a(1-y)^2$$

$$d(y) = -by(1-y), \text{ where } y = \frac{r}{r+\alpha}.$$

We prove the well-posedness of this problem after having done some preliminary work.

25.4.1 Gronwall's Inequalities

We state an inequality that is used to prove the stability of solutions to differential, integral and integro-differential equations.

Theorem 25.1 (Gronwall's inequality, continuous case) Let u and β be continuous functions on the interval $[a, b]$ such that $\beta(t) \geq 0 \forall t \in [a, b]$. Assume the inequality:

$$u(t) \leq K + \int_a^t \beta(s)u(s)ds \text{ where } K \text{ is a constant.}$$

$$\text{Then } u(t) \leq K \exp\left(\int_a^t \beta(s)ds\right), \quad t \in [a, b].$$

The following inequality is the discrete analogue of Theorem 25.1. It is used to prove stability of finite difference schemes. We include it for completeness.

Theorem 25.2 (Gronwall's inequality, discrete case) Let $a_n, n = 0, \dots, N$ be a sequence of real numbers such that $|a_n| \leq K + hM \sum_{j=0}^{n-1} |a_j|$, $n = 1, \dots, N$ where K and M are positive constants, then $|a_n| \leq (hM|a_0| + K) \exp(Mnh)$, $n = 1, \dots, N$.

A generalisation of Theorem (25.2) is:

$$\text{Let } |a_n| \leq K + \sum_{r=0}^{n-1} g(r)|a_r| \text{ with } g(r) \geq 0, \text{ then } |a_n| \leq K \exp\left(\sum_{r=0}^{n-1} g(r)\right).$$

The following result can be seen as the differential inequality version of Theorem 25.1.

Lemma 25.1 (Kreiss and Lorenz (2004)). Let $\phi \in C^1[0, \infty)$ and let $y(t)$ and $y_0(t)$ denote non-negative C^1 functions defined for $0 \leq t \leq T$.

If:

$$y'(t) \leq \phi(y(t)), \quad y'_0(t) = \phi(y_0(t)), \quad 0 \leq t \leq T,$$

with

$$y(0) \leq y_0(0).$$

then $y(t) \leq y_0(t)$ in $0 \leq t \leq T$.

25.4.2 Energy Inequalities

In this section we prove that the solution of Equation (25.10) is bounded in a certain norm by its pay-off function. The proof uses integration by parts in combination with Gronwall's inequality. At a certain stage in the analysis, we will see that the Feller condition (25.2) must be satisfied if the solution is to be unique. The proof has been condensed into a number of steps that we now delineate. A good exercise for the reader would be to reproduce and check these results. The maths is easy.

Step 1: we multiply (25.10) on both sides by the solution $u(y, t)$ and integrate in the interval $(0, 1)$ to get:

$$\int_0^1 \frac{\partial u}{\partial t} \cdot u dt = I_1 + I_2 + I_3 \tag{25.11}$$

where:

$$I_1 = \int_0^1 a(y) \frac{\partial u}{\partial y} \left(b(y) \frac{\partial u}{\partial y} \right) dy$$

$$I_2 = \int_0^1 c(y) \frac{\partial u}{\partial y} u dy$$

$$I_3 = \int_0^1 d(y) \frac{\partial u}{\partial y} u dy.$$

Step 2: Integrate the integrals in (25.11) by parts:

$$\begin{aligned} I_1 &= \frac{1}{4}\sigma^2 u^2(0, t) + \int_0^1 f'(y)u^2(y)dy \quad (f(y)) = \frac{1}{2}\alpha(y)b(y) \quad (f' = \frac{df}{dy}) \\ I_2 &= \frac{1}{2}au^2(0, t) - \int_0^1 \frac{1}{2}c'(y)u^2dy \quad (c'(y) \equiv \frac{dc}{dy}) \\ I_3 &= \left[\frac{1}{2}d(y)u^2 \right]_0^1 - \int_0^1 \frac{1}{2}d'(y)u^2 dy \quad (d'(y) \equiv \frac{dd}{dy}). \end{aligned} \quad (25.12)$$

Step 3: Group terms:

$$\begin{aligned} \frac{1}{2} \int_0^1 \frac{du^2}{dt} dy &= \frac{1}{2} \frac{d}{dt} \int_0^1 u^2 dy \leq \frac{1}{4}(\sigma^2 - 2a)u^2(0, t) + \int_0^1 g(y)u^2 dy \\ \left(g(y) \equiv f'(y) - \frac{1}{2}c'(y) - \frac{1}{2}d'(y) \right). \end{aligned} \quad (25.13)$$

We define the quantity $\|u\|^2(t) \equiv \int_0^1 |u(x, t)|^2 dx$.

Step 4: Integrating (25.13) for $0 \leq t \leq \xi$, we get the following inequality while using the Feller condition (25.2):

$$\|u\|^2(\xi) \leq \|u\|^2(0) + \frac{1}{4}(\sigma^2 - 2a) \int_0^\xi |u(0, t)|^2 dt + M \int_0^\xi \|u\|^2(t) dt. \quad (25.14)$$

$$\|u\|^2(\xi) \leq \|u\|^2(0) + M \int_0^\xi \|u\|^2(t) dt \text{ if } \sigma^2 - 2a < 0. \quad (25.15)$$

The Feller condition is jumping out at us in (25.14).

Step 5: Apply Gronwall's inequality (Theorem 25.1) to inequality (25.15):

$$\|u\|^2(\xi) \leq \|u\|^2(0) \exp M\xi \text{ where } M \text{ is a positive constant.} \quad (25.16)$$

This completes the proof.

25.5 FINITE DIFFERENCE METHODS FOR THE CIR MODEL

We now discuss how to approximate the initial boundary value problem for the CIR PDE. The main use cases are:

- U1: Choosing finite difference methods for (25.5), (25.10).
- U2: Approximating the boundary conditions in (25.6).

- U3: Ensuring stable schemes even with discontinuous pay-offs, for example digital options:

$$V(r, \tau = 0) = \begin{cases} 1, & \text{if } P(r, T, s) \geq K \\ 0, & \text{if } P(r, T, s) < K. \end{cases} \quad (25.17)$$

- U4: Domain truncation versus domain transformation.
- U5: Sensitivity analysis (computing duration, convexity and related sensitivities). This important topic is outside the scope of this book.

We have already addressed solutions to these use cases in this book. The most important requirement is probably that the numerical schemes remain second-order accurate for a wide range of parameters and pay-offs. Some issues that can cause performance degradation are:

- Convection dominance and/or large and small values of volatility.
- How to choose the optimal r_{\max} when using domain truncation (typical values are $r_{\max} \in [0.10, 0.50]$) In general, the bigger the value, the better the accuracy.
- Avoiding oscillations due to discontinuous pay-off functions.

We take some initial examples to price a zero coupon bond and we take the values $a = 0.048, b = 0.08, r_0 = 0.08, T = 0.5, \sigma = 0.4$. The analytic solution is 0.97896681. We compare implicit Euler (IE), Crank–Nicolson (CN) and ADE (Barakat and Clark), and we summarise the results as a tuple with elements [fdm_name, NY, NT; value]. We have:

```
[IE, 50,100; 0.978940202], [ADE, 50,100; 0.97895050], [CN, 50,100; 0.978948989]
[IE, 200,200; 0.97896166], [ADE, 200,200; 0.97896307], [CN, 200,200; 0.978966138]
[IE, 500,1000; 0.978965804], [ADE, 500,1000; 0.978965869], [CN, 500,1000;
0.978966702]
```

We now take a stress test with $a = 0.048, b = 0.08, r_0 = 0.08, T = 1.0, \sigma = 1.0$. The analytic solution is 0.97913863.

```
[IE, 50,100; 0.97912121], [ADE, 50,100; 0.97909611], [CN, 50,100; 0.9791261118]
[IE, 200,200; 0.97913775], [ADE, 200,200; 0.979022093], [CN, 200,200; 0.979137979]
[IE, 500,1000; 0.979138469], [ADE, 500,1000; 0.97910949], [CN, 500,1000;
0.979138528]
```

We see that Crank–Nicolson is the most accurate of the three methods. For ADE, Barakat and Clark was used for the diffusion term and Roberts–Weiss for the convection term. In this case, the large value of volatility might affect accuracy. We could use exponential fitting to make it more robust if necessary. The interested reader might like to write down the relevant ADE algorithm for the PDE (25.5) or (25.10).

25.5.1 Numerical Boundary Conditions

It now remains to discuss how to approximate the boundary condition (25.6) at the boundary $r = 0$. This is a first-order hyperbolic PDE, and we need to produce stable and

accurate schemes to approximate it and *integrate it* with the main scheme in the interior of the domain. A number of heuristic methods have been proposed, but we examine the second-order scheme in Thomée (1963). To this end, we take the generic PDE:

$$\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = 0, \quad a > 0. \quad (25.18)$$

Then the *Thomée scheme* is given by:

$$\frac{u_{j+\frac{1}{2}}^{n+1} - u_{j+\frac{1}{2}}^n}{k} + a \frac{u_j^{n+\frac{1}{2}} - u_j^{n+\frac{1}{2}}}{h} = 0 \quad (25.19)$$

where:

$$u_{j+\frac{1}{2}}^n \equiv \frac{1}{2}(u_{j+1}^n + u_j^n), \quad u_j^{n+\frac{1}{2}} \equiv \frac{1}{2}(u_j^{n+1} + u_j^n).$$

In the current case, we wish to apply this scheme to (25.6) for the index $j = 0$ (which corresponds to the boundary $r = 0$). The scheme now corresponds to two unknowns B_0^{n+1} and B_1^{n+1} . We produce a 2×2 system for these unknowns by noting that the main scheme is defined at $j = 1$. We then solve this *mini-system* for B_0^{n+1} and then B_1^{n+1} . This will result in a numerical Dirichlet boundary condition for the full scheme, whether it be Crank–Nicolson or ADE.

We note that we are using domain truncation for scheme (25.18). For domain transformation, we need to solve (25.18) when $y = 0$. We get:

$$\frac{\partial B}{\partial t} = a \frac{\partial B}{\partial y} \text{ at } y = 0. \quad (25.20)$$

The PDE at the near field is the same in both coordinate systems! We can thus apply scheme (25.19) to (25.20) at the near field as before.

25.6 HESTON MODEL AND THE FELLER CONDITION

We now show how splitting techniques can be applied to the Heston PDE that we introduced in Chapter 11, in particular the Feller conditions and the generalisation of the Thomée scheme (25.19) to two space variables. The following analysis holds true whether we use domain truncation or domain transformation as with the CIR model above.

To this end, we define the operators:

$$\begin{aligned} L_S U &\equiv A \frac{\partial^2 U}{\partial S^2} + B \frac{\partial U}{\partial S} + C U \\ L_v U &\equiv D \frac{\partial^2 U}{\partial v^2} + D \frac{\partial U}{\partial v} \end{aligned} \quad (25.21)$$

$$F \equiv \rho \sigma v S \text{ (coefficient of cross term)}$$

where the coefficients A, B, C, E and F are well-known Heston coefficients.

Formally, our splitting scheme is given by the following set of equations:

$$\begin{aligned} -\frac{\partial U}{\partial t} + L_S U + F \frac{\partial^2 U}{\partial S \partial v} &= 0 \\ -\frac{\partial U}{\partial t} + L_v U &= 0. \end{aligned} \quad (25.22)$$

Note: We are now using a forward equation in time in the respective directions. This is why there is a minus sign in front of the derivative with respect to t .

Furthermore, we approximate the elliptic operators in (25.21) by their finite difference equivalents:

$$\begin{aligned} \tilde{L}_S U_{ij}^n &= A_{ij}^n D + D_-^{(S)} U_{ij}^n + B_{ij}^n D_0^{(S)} U_{ij}^n + C_{ij}^n U_{ij}^n \\ \tilde{L}_v U_{ij}^n &= D_{ij}^n D_+^{(v)} U_{ij}^n + E_{ij}^n D_0^{(v)} U_{ij}^n. \end{aligned} \quad (25.23)$$

We are now ready to formulate the splitting scheme. The first leg calculates a solution at level $n + 1/2$ given the solution at level n :

$$\begin{aligned} -\frac{U_{ij}^{n+\frac{1}{2}} - U_{ij}^n}{k} + \tilde{L}_S U_{ij}^{n+1/2} + \frac{1}{2} F_{ij}^n D_0^S D_0^v U_{ij}^n &= 0 \\ n \geq 0, \quad 1 \leq i \leq I-1, \quad 1 \leq j \leq J-1 \end{aligned} \quad (25.24)$$

The second leg brings us from level $n + 1/2$ to level $n + 1$:

$$-\frac{U_{ij}^{n+1} - U_{ij}^{n+\frac{1}{2}}}{k} + \tilde{L}_v U_{ij}^{n+1} + \frac{1}{2} F_{ij}^n D_0^S D_0^v U_{ij}^{n+1/2} = 0. \quad (25.25)$$

with $n \geq 0, 1 \leq i \leq I-1, 1 \leq j \leq J-1$.

The scheme in (25.24) and (25.25) is essentially the Lie–Trotter scheme that we discussed in Chapter 22.

Please note how we have approximated the mixed derivative terms as advocated in Yanenko (1971), namely in an explicit way.

We now come to the approximation of the boundary conditions for this problem. We concentrate on PDE (11.40) (Chapter 11) because it is new and the other conditions have already been discussed in previous chapters. We write Equation (11.40) in the more convenient form:

$$-\frac{\partial U}{\partial t} + \alpha \frac{\partial U}{\partial S} + \beta \frac{\partial U}{\partial v} + bU = 0 \quad (v = 0) \quad (25.26)$$

where the new coefficients α, β and b are defined by:

$$\alpha = rS, \quad \alpha > 0$$

$$b = -r, \quad b < 0$$

$$\beta = K\theta, \quad \beta > 0.$$

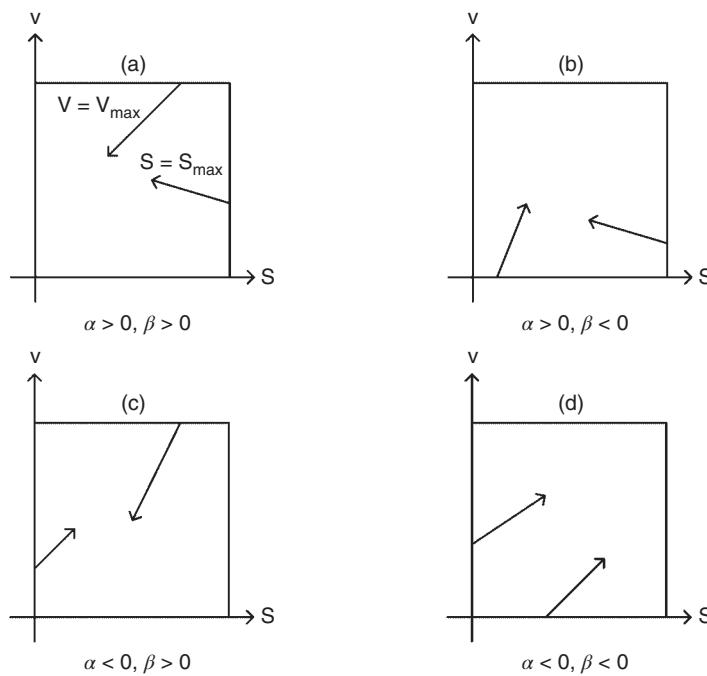


FIGURE 25.1 Direction of information flow (two-dimensional case).

We mention that the signs of the coefficients α and β determine where the information in the system is coming from. This is shown in Figure 25.1 for the four different cases. Our current situation corresponds to case (a). Thus, information at some node (i, j) is coming from ‘upwind’ nodes such as $(i + 1, i)$, $(i, j + 1)$ and $(i + 1, j + 1)$. This regime must be mirrored by the finite difference schemes. We thus choose the correct scheme in space, and we can choose between the following kinds of time marching:

- Explicit Euler scheme (conditionally stable)
- Implicit Euler (unconditionally stable).

Looking at Figure 25.2, we see that we must approximate (25.26) when $j = 0$. Taking into account the upwinding effects, we then propose the following explicit scheme:

$$-\frac{U_{i,0}^{n+1} - U_{i,0}^n}{k} + \alpha_{i,0} \frac{U_{i+1,0}^n - U_{i,0}^n}{h_1} + \beta_{i,0} \frac{U_{i,1}^n - U_{i,0}^n}{h_2} + bU_{i,0}^n = 0. \quad (25.27)$$

Some arithmetic and rearranging shows that:

$$U_{i,0}^{n+1} = (1 - \lambda_1 - \lambda_2 + bk)U_{i,0}^n + \lambda_1 U_{i+1,0}^n + \lambda_2 U_{i,1}^n \quad (25.28)$$

where:

$$\lambda_1 = \frac{\alpha_{i,0}k}{h_1} > 0 \text{ and } \lambda_2 = \frac{\beta_{i,0}k}{h_2} > 0.$$

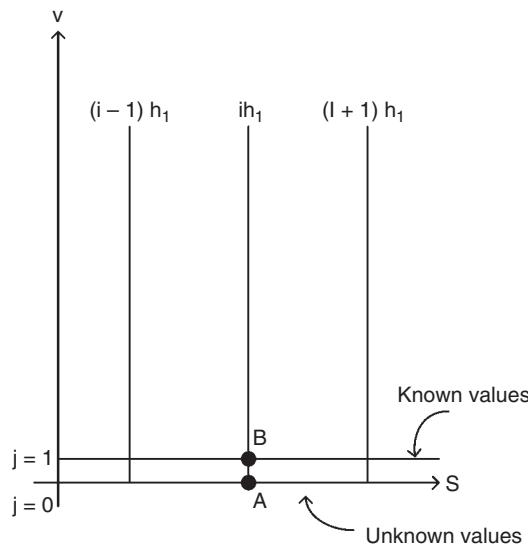


FIGURE 25.2 Approximation on the boundary.

Appealing to the discrete maximum principle by examining the right-hand side of Equation (25.27), we know that the values at level n and at the nodes $(i, 1)$ and $(i + 1, 0)$ are non-negative; we know that there is just one sufficient condition to make the right-hand side positive, namely (taking $b = 0$). Then for convenience:

$$1 - \lambda_1 - \lambda_2 \geq 0 \text{ or } k \leq \frac{1}{\frac{\alpha}{h_1} + \frac{\beta}{h_2}}.$$

When $b \neq 0$ we get a slightly different estimate.

This is the same condition as in Thomas (1995). The scheme (25.27) is thus conditionally stable, and this allows us to define what is essentially the Dirichlet boundary conditions on $v = 0$.

We now consider the implicit Euler scheme. In space it is almost the same as (25.27), except that values are taken at time level $n + 1$:

$$-\frac{U_{i,0}^{n+1} - U_{i,0}^n}{k} + \alpha_{i,0} \frac{U_{i+1,0}^{n+1} - U_{i,0}^{n+1}}{h_1} + \beta_{i,0} \frac{U_{i,1}^{n+1} - U_{i,0}^{n+1}}{h_2} + bU_{i,0}^{n+1} = 0 \quad (25.29)$$

Some arithmetic shows that:

$$-U_{i,0}^{n+1} + U_{i,0}^n + \lambda_1(U_{i+1,0}^{n+1} - U_{i,0}^{n+1}) + \lambda_2(U_{i,1}^{n+1} - U_{i,0}^n) + bkU_{i,0}^{n+1} = 0$$

and thus: (25.30)

$$U_{i,0}^{n+1} = \frac{U_{i,0}^n + \lambda_1 U_{i+1,0}^{n+1} + \lambda_2 U_{i,1}^{n+1}}{1 + \lambda_1 + \lambda_2 - bk}.$$

Appealing to the maximum principle and monotonicity, we see that the solution at time level $n + 1$ is positive because all data on the right-hand side of (25.29) is positive (notice that $b < 0$).

Summarising, we solve this problem using splitting and incorporating the appropriate boundary conditions in S and v .

Finally, an alternative to schemes (25.27) and (25.29) is the two-dimensional generalisation of the upwind monotone scheme that we introduced in Section 24.4.3 (Chapter 24) to approximate the first order PDE (25.26):

$$\frac{u_{ij}^{n+1} + u_{ij}^n}{\Delta t} + \alpha \frac{u_{i+1,j}^n - u_{ij}^{n+1}}{h_1} + \beta \frac{u_{i,j+1}^n - u_{ij}^{n+1}}{h_2} + bu_{ij}^{n+1} = 0. \quad (25.31)$$

This scheme has first-order accuracy on the boundary $v = 0$ ($y = 0$). There is no need to solve a mini-system, because we can solve directly for u_{ij}^{n+1} , but an open question is whether this affects second-order global accuracy. The findings in Gustafsson (1975) would suggest this hypothesis.

25.7 SUMMARY AND CONCLUSION

In this chapter we developed robust finite difference schemes for short-rate interest models, specifically the popular CIR model as introduced in Cox, Ingersoll and Ross (1985). We addressed and resolved a number of open issues concerning heuristic numerical methods to solve the initial boundary value problem for the CIR PDE. We generalised and extended the results to the Heston model.

CHAPTER 26

Epilogue Models Follow-Up Chapters 1 to 25

The end is in the beginning and yet you go on.

Samuel Beckett, *Endgame*

26.1 INTRODUCTION AND OBJECTIVES

As the title suggests, this chapter looks back at the work done in the first 25 chapters of this book and we review the results with a view to adding to them or improving them in some way. In particular, we discuss themes that fall in the following categories:

- More guidelines and schemes on making the content even better.
- Avoiding bad practices and ad hoc solutions.
- Some new methods and schemes that are not well-known in the mainstream quant “folklore”.

To be specific, we look at the following topics:

- Creating monotone schemes for PDEs with a mixed derivative term; the serendipity of finding M-matrices with constant meshes.
- Sensitivities revisited; the Faddeeva function for option greeks; the Kunita stochastic flow for SDE sensitivities.
- MOL and ADE for two-factor Hull-White model.

We discuss these topics at a high level of abstraction. Unfortunately, a discussion of these topics is beyond the scope of the current work.

26.2 MIXED DERIVATIVES AND MONOTONE SCHEMES

In general, we are interested in developing finite difference schemes whose properties mirror those of the PDEs that they approximate. Ideally, the schemes should be accurate approximations to the solutions of PDEs. We have discussed this issue in great detail in the first 25 chapters of this book. In particular, producing monotone schemes and schemes that satisfy the discrete maximum principle is something to strive for. Naïve schemes do not necessarily satisfy this principle, with the result that the approximate solution may oscillate or exhibit undesirable pathological behaviour.

Some common problems, challenges and possible solutions are:

- Convection-dominated problems and their resolution by exponentially fitted and upwinding methods.
- Avoiding numerical oscillations with discontinuous pay-offs (initial conditions).
- Spurious reflections at boundaries, spatial amplification of errors.
- Handling mixed derivatives for two-factor PDEs by transforming a PDE to canonical form to eliminate the mixed derivative term.
- Using the Craig–Sneyd and Yanenko schemes to approximate the mixed derivative term.

We have already addressed the above problems. In this section we introduce a number of methods to find monotone schemes for PDEs containing a mixed derivative term. There is little information available in the finance literature (with the exception of Ma and Forsyth (2017)), even though the methods were developed in the former Soviet Union in the 1960s (Matus (2002)). We discuss these topics and show how to use them on several popular two-factor problems.

The underlying mathematical theory for this section is discussed in Part B of this book, in particular Chapters 8 to 11.

26.2.1 The Maximum Principle and Mixed Derivatives

We reduce the scope by examining the diffusion part of Equation (8.5), namely:

$$\frac{\partial u}{\partial t} = a \frac{\partial^2 u}{\partial x^2} + 2b \frac{\partial^2 u}{\partial x \partial y} + c \frac{\partial^2 u}{\partial y^2}. \quad (26.1)$$

We discuss how to approximate Equation (26.1) by finite differences with the goal of producing monotone schemes, in particular examining the role of the coefficient b of the mixed derivative term. This coefficient is a function of the space and time variables in general, and it may even change sign in the region of integration. The sign of b will influence the finite difference approximation to use, as we shall see. Ideally, the coefficients a , b and c are constant, or failing that, the PDE (26.1) can be transformed to a PDE with constant coefficients. In Samarskii, Matus, Mazhukin and Mozolevski (2002) the authors discuss second-order accurate difference schemes for elliptic equations with mixed derivative terms that can change sign.

In order to proceed, we need to define some notation. We consider the usual stencil of discrete values with constant mesh sizes h_1 and h_2 . We define the primitive difference operators as follows:

$$\left. \begin{array}{l} D_x^+ u_{ij} = h_1^{-1}(u_{i+1,j} - u_{i,j}) \\ D_x^- u_{ij} = h_1^{-1}(u_{i,j} - u_{i-1,j}) \\ D_y^+ u_{ij} = h_2^{-1}(u_{i,j+1} - u_{i,j}) \\ D_y^- u_{ij} = h_2^{-1}(u_{i,j} - u_{i,j-1}) \end{array} \right\} \quad (26.2)$$

and the composite operators:

$$\left. \begin{array}{l} 2\delta_{xy}^+ = D_x^+ D_x^+ + D_x^- D_y^- \\ 2\delta_{xy}^- = D_x^+ D_x^- + D_x^- D_y^+ \end{array} \right\}. \quad (26.3)$$

These composite operators applied to a mesh function deliver the following representations:

$$\left. \begin{array}{l} 2\delta_{xy}^+ u_{ij} = (h_1 h_2)^{-1} [u_{i+1,j+1} - u_{i,j+1} - u_{i+1,j} + u_{i,j} \\ \quad + u_{i,j} - u_{i-1,j} - u_{i,j-1} + u_{i-1,j-1}] \\ 2\delta_{xy}^- u_{ij} = (h_1 h_2)^{-1} [u_{i+1,j} - u_{i,j} - u_{i+1,j-1} + u_{i,j-1} \\ \quad + u_{i,j+1} - u_{i-1,j+1} - u_{i,j} + u_{i-1,j}] \end{array} \right\}. \quad (26.4)$$

We now show how to approximate the mixed derivative term (the outcome depends on the sign of b):

$$b^\pm = \frac{1}{2}(b \pm |b|) \quad (26.5)$$

In order to reduce the scope and to avoid confusion, we assume that $b > 0$ (we leave the case $b < 0$ as an exercise), and we now continue with the ‘full’ discretisation of the PDE (26.1) in the usual way. Grouping like terms on the right-hand side for the finite difference approximation of (26.1) and ignoring time dependence for the moment, we then get the term:

$$\beta_0 u_{0,j} + \beta_1 u_{i+1,j} + \beta_2 u_{i+1,j+1} + \beta_3 u_{i,j+1} + \beta_4 u_{i-1,j} + \beta_5 u_{i-1,j-1} + \beta_6 u_{i,j-1} \quad (26.6)$$

where:

$$\left. \begin{array}{l} \beta_0 = \frac{2a}{h_1^2} - \frac{2c}{h_2^2} + \frac{b}{h_1 h_2} \\ \beta_2 = \beta_5 = \frac{b}{h_1 h_2} \\ \beta_3 = \beta_6 = \frac{c}{h_2^2} - \frac{b}{h_1 h_2} \\ \beta_1 = \beta_4 = \frac{a}{h_1^2} - \frac{b}{h_1 h_2} \end{array} \right\}. \quad (26.7)$$

For monotonicity to hold, we need to satisfy the following inequalities:

$$-\frac{2a}{h_1^2} - \frac{2c}{h_2^2} + \frac{b}{h_1 h_2} \leq 0 \quad (26.8)$$

$$\frac{a}{h_1^2} - \frac{b}{h_1 h_2} \geq 0 \quad (26.9)$$

$$\frac{c}{h_2^2} - \frac{b}{h_1 h_2} \geq 0. \quad (26.10)$$

Equations (26.9) and (26.10) represent the constraints that must be satisfied if we are to preserve ellipticity and produce monotone difference schemes. We can write the constraints as follows:

$$\frac{b}{c} \leq \frac{h_1}{h_2} \leq \frac{a}{b} \quad (26.11)$$

and more generally:

$$\frac{|b|}{c} \leq \frac{h_1}{h_2} \leq \frac{a}{|b|}. \quad (26.12)$$

The next question is to determine if these constraints are useful in practice. The ideal situation is when we can satisfy constraint (26.12) with constant or even non-constant meshes. In general, there may be no solution, but we might get lucky in certain isolated cases. An initial discussion and general results are given in Dunne, O'Riordan and Shishkin (2009), but the claims regarding sufficient conditions to generate an M-matrix are not proved. A counter-example is given in Roos (2019), showing that first-order schemes for elliptic problems with mixed derivatives on highly *anisotropic (non-uniform) meshes* cannot generate an M-matrix. In contrast, *isotropic meshes* can generate M-matrices (see Matus (2002)). Furthermore, this latter article also discusses monotone methods for convection diffusion equations by fitting the diffusion and convection terms (similar to Duffy's exponential fitting) and using schemes (26.3) and (26.5) for the mixed derivative term.

26.2.2 Some Examples

The next question to ask is whether PDEs in finance can be approximated using the above approach. The ideal situation is when the coefficients of the PDE are constant (for example, the two-factor Hull–White model) or when the PDE can be transformed to one with constant coefficients (for example, multi-asset models). Another serendipitous situation is when the ratios of the coefficients in (26.12) are constant (for example, the Heston model) or when the coefficients themselves are non-constant. Then we can compute constant mesh sizes to produce monotone schemes. To this end, we take an example.

The following PDE describes the two-factor Hull–White model:

$$\frac{\partial P}{\partial t} = \frac{1}{2}\sigma_1^2 \frac{\partial^2 P}{\partial r^2} + \frac{1}{2}\sigma_2^2 \frac{\partial^2 P}{\partial u^2} + p\sigma_2\sigma_2 \frac{\partial^2 P}{\partial r \partial u} + [\theta(T - \tau) + u - ar] \frac{\partial P}{\partial r} - bu \frac{\partial P}{\partial u} - rP. \quad (26.13)$$

In this case we have:

$$\begin{aligned} a &= \frac{1}{2}\sigma_1^2, b = \rho\frac{\sigma_1\sigma_2}{2}, c = \frac{1}{2}\sigma_2^2 \\ \rho\left(\frac{\sigma_1}{\sigma_2}\right) &\leq \frac{h_1}{h_2} \leq \rho^{-1}\left(\frac{\sigma_1}{\sigma_2}\right). \end{aligned} \quad (26.14)$$

For the spread options of Chapter 23, for example, we perform log transformations, and we get the same constraints as in (26.14).

Finally, the Heston PDE is:

$$\frac{\partial u}{\partial t} = \frac{1}{2}yS^2\frac{\partial^2 u}{\partial S^2} + \frac{1}{2}\sigma^2v\frac{\partial^2 u}{\partial v^2} + \rho\sigma Sv\frac{\partial^2 u}{\partial S\partial v} + \dots \quad (26.15)$$

and after the log transformation $x = \log S$ we get (Set $y = v$ as notational convenience):

$$\frac{\partial u}{\partial t} = \frac{1}{2}y\frac{\partial^2 u}{\partial x^2} + \frac{1}{2}\sigma^2y\frac{\partial^2 u}{\partial y^2} + \rho\sigma y\frac{\partial^2 u}{\partial x\partial y} + \dots \quad (26.16)$$

leading to:

$$\begin{aligned} a &= \frac{1}{2}y, b = \rho\sigma y/2, c = \frac{1}{2}\sigma^2y \\ \frac{\rho}{\sigma} &\leq \frac{h_1}{h_2} \leq \frac{1}{\rho\sigma}. \end{aligned} \quad (26.17)$$

An interesting project would be using schemes (26.3) to (26.5) instead of the Yanenko method for the multi-asset PDEs of Chapter 23.

26.2.3 Code Sample Method of Lines (MOL) for Two-Factor Hull–White Model

We show the code that implements the above monotone finite difference scheme for the two-factor Hull–White model by the Method of Lines (MOL) via Boost C++ library *odeint*. Recall that we discussed MOL in Chapter 20. We model the PDE:

```
// PDE for zero coupon bond.
//
// dP/dt = a1 P_rr + c P_ru + a2 P_uu + b1 P_r + b2 P_u + d P + F
//
// a1 = 1/2 s1^2, c = rho s1 s2, a2 = 1/2 s2^2,
// b1 = theta(T-tau) + u - ar, b2 = - bu, d = -r, F == 0.
//
```

We use these variable names in order to reduce the *cognitive distance* between the algorithm and C++ code. MOL entails semi-discretisation in space to produce a system of ODEs:

```
struct ZcbPdeSamarski
{
    double h1, h2;
```

```

ZcbPdeSamarski(double stepSize1, double stepSize2)
    : h1(stepSize1), h2(stepSize2) { }

void operator()(const state_type& x, state_type& dxdt, double t) const
{
    // Optimise later
    double h11 = 1.0 / (2 * h1);
    double h21 = 1.0 / (2 * h2);
    double h12 = 1.0 / (h1 * h1);
    double h22 = 1.0 / (h2 * h2);

    double a11tmp, a22tmp, ctmp, b1tmp, b2tmp, dttmp, xijtmp;
    double xa, ya;
    for (std::size_t i = 2; i < x.size1() - 2; ++i)
    {
        for (std::size_t j = 1; j < x.size2() - 1; ++j)
        {
            // For readability reasons
            xa = xarr[i]; ya = yarr[j];
            a11tmp = a11(xa, ya, t);
            a22tmp = a22(xa, ya, t);
            ctmp = crossTerm(xa, ya, t);
            b1tmp = b1(xa, ya, t);
            b2tmp = b2(xa, ya, t);
            dttmp = d(xa, ya, t);
            xijtmp = x(i, j);

            dxdt(i, j) = h12 * a11tmp * (x(i + 1, j)
                - 2.0 * xijtmp + x(i - 1, j))
                + h22*a22tmp*(x(i,j+1)-2.0*xijtmp + x(i,j - 1))
                + h11 * b1tmp * (x(i + 1, j) - x(i - 1, j))
                + h21 * b2tmp * (x(i, j + 1) - x(i, j - 1))
                + dttmp * xijtmp
                + F(xa, ya, t);

            // Test if correlation is positive or negative
            if (ctmp >= 0.0)
            { // Averaging

                dxdt(i, j) += (ctmp / (2 * h1 * h2)) *
                    (x(i+1,j+1)-x(i,j+1)-x(i+1,j) + x(i, j)
                     + x(i,j)-x(i-1,j)-x(i,j-1)+x(i-1,j - 1));
            }
            else
            {
                dxdt(i, j) += (ctmp / (2 * h1 * h2)) *
                    (x(i+1,j)-x(i,j)-x(i+1,j-1)+ x(i,j-1)
                     + x(i,j+1)-x(i-1,j+1)-x(i,j)+x(i- 1, j));
            }
        }
    }
}

```

In this code we see how scheme (26.4) has been implemented. We conclude this section with the snippet of code that realises constraint (26.12). We take specific test data for readability:

```
double T = 1.0;
double P1 = 1.0;
double P2 = 1.0;
double s1 = 0.0168;
double s2 = 0.00462;
double rho = -0.117;
double a = 0.5;
double b = 0.03;
```

The code to compute the mesh size (inverses of the number of steps) is:

```
std::size_t computeSteps(std::size_t I,
double sig1, double sig2, double corr)
{ // Given I steps in x direction, compute number of steps in y direction to
// produce a monotone scheme

    double nI = static_cast<double> (I);
    double L = std::abs(corr) * sig1 / sig2;
    double U = sig1 / (sig2*std::abs(corr));
    double val = (nI * L + nI * U) / 2.0;
    return static_cast<std::size_t> (nI*L);
}
```

26.3 THE COMPLEX STEP METHOD (CSM) REVISITED

In Chapters 16 and 17 we introduced the foundations of CSM. Now we extend it to the computation of Black–Scholes sensitivities by complexifying the cumulative normal distribution function by means of the Faddeeva function. We also show how to implement the gradient vector and Jacobian matrix for vector functions and vector-valued functions, respectively. A nice project based on these results could be to compare CSM and AD in applications, for example machine learning.

26.3.1 Black–Scholes Greeks Using CSM and the Faddeeva Function

In Section 16.3 we gave analytical solutions for first-order option sensitivities (greeks). Arriving at these solutions may entail calculus and drudgery. We may not be disposed to compute these solutions. Then, CSM offers an escape route. To this end, we complexify the normal distribution and the Black–Scholes equation that uses it. All parameters that are used as input to the latter function are complex, and this means that option greeks can be computed by defining a single function.

The normal distribution for complex arguments is defined in terms of the *Faddeeva function* (also known as the *Kramp function*):

$$w(z) := e^{-z^2} \operatorname{erfc}(-iz) = \operatorname{erfcx}(-iz) = e^{-z^2} \left(1 + \frac{2i}{\sqrt{\pi}} \int_0^z e^{t^2} dt \right). \quad (26.18)$$

This is essentially a scaled complex complementary error function. It has applications to various physical problems, such as electromagnetic responses in complicated media, small-amplitude waves and electromagnetic waves of the type used in AM radio, to name a few. Another representation is a convolution of a Gaussian with a simple pole:

$$w(z) = \frac{i}{\pi} \int_{-\infty}^{\infty} \frac{e^{-t^2}}{z-t} dt = \frac{2iz}{\pi} \int_{-\infty}^{\infty} \frac{e^{-t^2}}{z^2 - t^2} dt, \quad \operatorname{Im} z > 0 \quad (26.19)$$

Note also that:

$$w(-z) = 2e^{-z^2} - w(z) \quad (26.20)$$

and

$$w(-\bar{z}) = \overline{w(z)}$$

where \bar{z} is the complex conjugate of z .

The Faddeeva function is implemented in C++ by the Faddeeva package (from *Ab initio*) and in Python in `scipy.special.wofz`. We use the code as follows:

```
double N(double x)
{ // cdf
    return std::erfc(-x / std::sqrt(2.0))/2.0;
}

std::complex<double> N(std::complex<double>& z)
{
    double relErr = 1.0e-16;
    return Faddeeva::erfc(-z / std::sqrt(2.0), relErr) / 2.0;
}
```

We now produce the code to price a call option (the code for puts is similar and we do not show it here) as follows, noting that *all* parameters are complex:

```
std::complex<double> BS(std::complex<double>& K, std::complex<double>& T,
                        std::complex<double>& r, std::complex<double>& b
                        std::complex<double>& v, std::complex<double>& S)
{ // Call option price

    std::complex<double> LOG = std::log(S / K);
    std::complex<double> den = v * std::sqrt(T);
```

```

    std::complex<double> d1 = (LOG + (b + v * v * 0.5) * T) / den;
    std::complex<double> d2 = d1 - den;

    std::complex<double> result =
        S * N(d1) * std::exp((b-r) * T) - K * std::exp(-r * T) * N(d2);

    return result;
}

```

When we wish to compute the sensitivity with respect to one of the above six input parameters, we define it as a complex number with the step size as its imaginary part while setting the imaginary part of the other five parameters to zero. So, for example, to compute the delta we proceed as follows:

```

double h = 1.0e-23;
{
    // Full CSM
    // Haug (2007), page 26, delta = 0.5946
    double h = 1.0e-243;
    std::complex<double> K(100, 0.0); std::complex<double> T(0.5, 0.0);
    std::complex<double> r(0.1, 0.0); std::complex<double> b(0.0, 0.0);
    std::complex<double> v(0.36, 0.0); std::complex<double> S(105, 0.0);

    // Delta
    S.imag(h);

    auto result = BS(K, T, r, b, v, S);
    std::cout << "Delta full CSM: " << std::imag(result) / h << '\n';
}

```

For completeness, the code for the other sensitivities is:

```

{
    // Full CSM
    // Delta by CSM, Gamma by FDM of delta
    // Haug (2007), page 89, delta = 0.5946
    double h = 1.0e-243;
    std::complex<double> K(100, 0.0); std::complex<double> T(0.25, 0.0);
    std::complex<double> r(0.1, 0.0); std::complex<double> b(0.05, 0.0);
    std::complex<double> v(0.3, 0.0); std::complex<double> S(98, 0.0);

    // Delta
    S.imag(h);

    auto result = BS(K, T, r, b, v, S);
    auto dfdS = std::imag(result) / h;

    std::cout << "Delta 2 full CSM: " << dfdS << '\n';
}

```

```
// Gamma; hyrid solution
auto h2 = std::complex<double>(1.0e-5);
auto S2 = S + h2;
auto result2 = BS(K, T, r, b, v, S2);
auto dfdS2 = std::imag(result2) / h;

auto gamma = (dfdS2 - dfdS) / h2;

std::cout << "Gamma 2 full CSM: " << gamma << '\n';
}

{
// Vega, Haug page 50, vega = 18.5027
std::complex<double> K(60, 0.0); std::complex<double> T(0.75, 0.0);
std::complex<double> r(0.105,0.0); std::complex<double> b(0.0695, 0.0);
std::complex<double> v(0.3, 0.0); std::complex<double> S(55, 0.0);

v.imag(h);

auto result = BS(K, T, r, b, v, S);
std::cout << "Vega full CSM: " << std::imag(result) / h << '\n';
}

{
// Rho, Haug page 89, rho = 38.7325
std::complex<double> K(100,0.0); std::complex<double> T(0.25, 0.0);
std::complex<double> r(0.1,0.0); std::complex<double> b(0.05, 0.0);
std::complex<double> v(0.3, 0.0); std::complex<double> S(98, 0.0);

r.imag(h);

auto result = BS(K, T, r, b, v, S);
std::cout << "Rho full CSM: " << std::imag(result) / h << '\n';
}

{
// Carry Rho, Haug page 74, rho = -42.2253
std::complex<double> K(490, 0.0); std::complex<double> T(0.25, 0.0);
std::complex<double> r(0.08, 0.0); std::complex<double> b(0.03, 0.0);
std::complex<double> v(0.15, 0.0); std::complex<double> S(500, 0.0);

b.imag(h);

auto result = BSPut(K, T, r, b, v, S);
std::cout << "Carry Rho full CSM: " << std::imag(result) / h << '\n';
}

{
// Theta, Haug page 64, rho = -31.1924
std::complex<double> K(405, 0.0); std::complex<double> T(0.0833, 0.0);
std::complex<double> r(0.07, 0.0); std::complex<double> b(0.02, 0.0);
```

```

    std::complex<double> v(0.2, 0.0); std::complex<double> S(430, 0.0);

    T.imag(h);

    auto result = BSPut(K, T, r, b, v, S);
    std::cout << "Theta Rho full CSM: " << -std::imag(result) / h << '\n';
}

{
// Strike Delta Call, Haug page 89, value = -0.4386
    std::complex<double> K(100, 0.0); std::complex<double> T(0.25, 0.0);
    std::complex<double> r(0.1, 0.0); std::complex<double> b(0.05, 0.0);
    std::complex<double> v(0.3, 0.0); std::complex<double> S(98, 0.0);

    K.imag(h);

    auto result = BS(K, T, r, b, v, S);
    std::cout << "Strike Delta Call full CSM: " << std::imag(result) / h;
}

```

The output is:

*Delta full CSM: 0.5946286597299961
 Delta 2 full CSM: 0.503105268995501
 Gamma 2 full CSM: (0.02679431030383483,0)
 Vega full CSM: 18.50274408912756
 Rho full CSM: -1.360501690400331
 Carry Rho full CSM: -42.22536336911369
 Theta Rho full CSM: -31.19236705651886
 Strike Delta Call full CSM: -0.4386230959995765*

Summarising, this approach can be used to compute sensitivities of functions with respect to state variables and parameters. But the functions must have an analytical representation, and we must be able to complexify them.

26.3.2 CSM and Functions of Several Complex Variables

We have seen how CSM approximates the first-order derivative of a scalar-valued function of a scalar variable. As we know from Chapters 16 and 17, the trick is to *complexify* the function. The next challenge is to investigate functions of several complex variables for which extensive mathematical results are known (see Osgood (1966), Vladimirov (1966). We can ask ourselves if CSM can be applied to applications in optimisation, machine learning and their applications (see Nocedal and Wright (2006) Goodfellow, Bengio and Courville (2016)). In particular, we define generalised derivatives for vector functions and vector-valued functions. We now discuss some relevant concepts.

We first discuss the gradient vector of a vector function $f : \mathbb{R}^n \rightarrow \mathbb{R}, x = (x_1, \dots, x_n)^\top$ defined by:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right)^\top. \quad (26.21)$$

An example is the Rosenbrock function in two dimensions, which is a non-convex function and is used as a performance test problem for optimisation algorithms. It is also known as Rosenbrock's valley or Rosenbrock's banana function. It has a minimum value of zero at the point (1,1):

$$\begin{aligned} f(x, y) &= 100(y - x^2)^2 + (1 - x)^2 \\ \nabla f &= (-400(y - x^2)x - 2(1 - x), 200(y - x^2)). \end{aligned} \quad \left. \right\} \quad (26.22)$$

For a vector-valued function:

$$\begin{aligned} f : \mathbb{R}^n &\rightarrow \mathbb{R}^m, x = (x_1, \dots, x_n)^\top \\ f &= (f_1(x), \dots, f_m(x))^\top \end{aligned}$$

we define the *Jacobian matrix* as follows:

$$J_f(x) = \left(\frac{\partial f_i}{\partial x_j} \right), \quad 1 \leq i \leq m, 1 \leq j \leq n. \quad (26.23)$$

For completeness, the Hessian (square) matrix is defined by:

$$H = \left(\frac{\partial^2 f}{\partial x_i \partial x_j} \right), \quad 1 \leq i, j \leq n. \quad (26.24)$$

The CSM does not directly apply to this case, and for this reason we do not discuss it here, because CSM is not suitable for higher-order derivative approximation.

We give a concrete example of the Jacobian matrix:

$$\begin{aligned} f : \mathbb{R}^2 &\rightarrow \mathbb{R}^2 \\ f(x, y) &= (f_1(x, y), f_2(x, y))^\top = (x^2 y, 5x + \sin y)^\top. \end{aligned} \quad (26.25)$$

Then:

$$J_f(x, y) = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{pmatrix} = \begin{pmatrix} 2xy & x^2 \\ 5 & \cos y \end{pmatrix}. \quad (26.26)$$

26.3.3 C++ Code for Extended CSM

The C++ code for the basic CSM needs to be extended to support gradient vector and the Jacobi matrix. To this end, we design algorithms to compute these quantities as well as defining types that represent function categories. We use standard C++ vectors, and

we have created our own home-grown matrix class because C++ does not support such a data type. First, function and data types are:

```
// Data types
using value_type = double;
using cmpl_type = std::complex<value_type>;
using Vector = std::vector<cmpl_type>;
using Matrix = NestedMatrix < double>;

// Define some hierarchies of functions
using VectorFunction = std::function < cmpl_type(const Vector& c) >;
using VectorVectorFunction = std::function <cmpl_type(const Matrix& c) >;
using VectorValuedFunction = std::vector < VectorFunction >;
using FunctionType = std::function < std::complex<double>(const
std::complex<double>) >;
```

We now show the code to compute the gradient, directional derivative (one component of the gradient) and the Jacobian:

```
// Scalar step h case for a fn : C(n) -> C
std::vector<double> CsmGradient(const VectorFunction& f,
    Vector& zarr, double h)
{ // Gradient using the Complex Step method, in n dimensions

    auto sz = zarr.size();
    auto eps = std::complex<double>(0.0, h);

    // Gradient vector
    std::vector<double> result(sz);

    zarr[0] += eps;
    result[0] = std::imag(f(zarr)) / h;

    for (std::size_t i = 1; i < sz; ++i)
    { // Compute df/dx_i directional derivative

        zarr[i - 1] -= eps;
        zarr[i] += eps;
        result[i] = std::imag(f(zarr)) / h;
    }

    // Reset the last value in zarr to its original value
    zarr[sz - 1] -= eps;

    return result;
}
```

```

// Scalar step h case for directional derivative
value_type DirectionalDerivative(const VectorFunction& f, Vector& zarr,
                                   double h, std::size_t j)
{ // Component derivative using CSM, in n dimensions

    auto eps = std::complex<double>(0.0, h);

    zarr[j] += eps;
    auto val = std::imag(f(zarr)) / h;
    zarr[j] -= eps;

    return val;
}
// Matrix Jacobian case

Matrix CsmJacobian(const VectorValuedFunction& f, Vector& zarr, double h)
{ // Gradient using the Complex Step method, in n dimensions

    auto NR = f.size(); auto NC = zarr.size();
    // Gradient vector
    Matrix result(NR, NC);

    for (std::size_t i = 0; i < NR; ++i)
    {
        // Capture gradient for f[i] component function
        auto rowSolution = CsmGradient(f[i], zarr, h);

        for (std::size_t j = 0; j < rowSolution.size(); ++j)
        {
            result(i, j) = rowSolution[j];
        }
    }

    return result;
}

```

We take two examples. The first one is the Rosenbrock function and its exact gradient (26.23):

```

std::complex<double> Rosenbrock (const std::vector<std::complex<double>>& x)
{ // Objective function in NL minimax solvers

    std::complex<double> x1 = x[0]; std::complex<double> x2 = x[1];

    // Rosenbrock
    return 100.0*(x2 - x1*x1)*(x2 - x1*x1) + (1.0 - x1)*(1.0 - x1);
}

std::vector<std::complex<double>>
RosenbrockGradient(std::vector<std::complex<double>>& x)

```

```
{
    // Gradient function in NL minimax solvers

    std::vector<std::complex<double>> result(2);

    std::complex<double> x1 = x[0]; std::complex<double> x2 = x[1];
    result[0] = -400.0*(x2 - x1*x1)*x1 + 2.0*(1.0 - x1);
    result[1] = 200.0*(x2 - x1*x1);

    return result;
}
```

The C++ code using CSM is:

```
{
    // Rosenbrock scalar

    std::vector<std::complex<value_type>> seed(2);
    seed[0] = 1.0; seed[1] = 1.0;

    double h = 1.0e-60;

    auto grad = CsmGradient(Rosenbrock, seed, h);
    std::cout << "Rosenbrock CSM scalar " << grad[0] << ", " << grad[1];
    auto grad2 = RosenbrockGradient(seed);
    std::cout << "Rosenbrock exact "
        << std::real(grad2[0]) << ", " << std::real(grad2[1]) << '\n';
}
```

We can run the code and see that the exact and approximate values for the Rosenbrock gradient are the same.

The second example computes the Jacobian matrix based on input from (26.26):

```
{
    std::cout << "2X2 jacobian\n";
    std::vector<std::complex<value_type>> seed(2);
    seed[0] = 1.0; seed[1] = 0.0;

    double h = 1.0e-160;

    auto f4 = [] (const std::vector<std::complex<double>>& x)
    {
        return x[0] * x[0] * x[1]; };
    auto f4A = [] (const std::vector<std::complex<double>>& x)
    { return 5.0*x[0] + std::sin(x[1]); };

    // Matrix: row 1 {0, 1}, row 2 {5, 1}
    VectorValuedFunction mvf{ f4,f4A};
    NestedMatrix<double> jacobian = CsmJacobian(mvf, seed, h);
    jacobian.print();
}
```

26.3.4 CSM for Non-Linear Solvers

Many applications need to compute derivatives of functions. In some cases it may be difficult or inconvenient to compute them analytically, and it may be an option to employ CSM. We take the well-known Newton–Raphson method (Duffy (2018)). The toy code is:

```
#include <functional>
#include <complex>
#include <iostream>
#include <iomanip>
#include <cmath>

using value_type = double;

template <typename T> T f(const T& t)
{ // Solve t - std::exp(5.0) = 0

    return t - std::exp(5.0);
    //return t * t - 2.0;
}

const double h = 1.0e-220;

template <typename T> T dfdx(const T& t)
{ // Complex Step derivative

    std::complex<value_type> z(t, h); // x + ih
    return std::imag(f(z)) / h;
}

int main()
{
    value_type TOL = 1.0e-12;

    value_type x0 = 1000.014; // initial guess
    value_type x1;
    std::size_t counter = 0;
    L0:

        x1 = x0 - f(x0) / dfdx(x0);
        if (std::abs(x1 - x0) > TOL)
        {
            x0 = x1;
            counter++;
            goto L0;
        }
}
```

```

    std::cout << std::setprecision(16) << x1 << ", " << counter << '\n';
    return 0;
}

```

26.4 EXTENDING THE HULL-WHITE: POSSIBLE PROJECTS

In Section 26.2.3 we briefly described the approximation of the two-factor Hull–White PDE (26.13) using MOL and ADE. The background research (which is not described here) was meant as a proof of concept to determine if PDE methods can be applied to interest rate problems. There is no reason why they cannot be used. Just because (almost) no one has tried does not mean that it cannot be done. For example, we have achieved good results in Chapter 25.

At first glance, the Hull–White seemingly differs from the Black–Scholes and CIR in a number of respects:

- It is defined on an infinite interval $-\infty < r < \infty$ (negative interest rates are possible). Most of the models we discussed were defined on a semi-infinite interval, for example $0 < r < \infty, 0 < S < \infty$. Whether this makes analysis more difficult has yet to be seen.
- Since there is no boundary, we conclude that the Fichera theory is not directly applicable to (26.13). We may still need to define asymptotic values for the solution of (26.13).
- One of the specific areas for attention is to compute the time-dependent function $\theta(t)$ that represents the speed of mean reversion. It is given by the formula:

$$\theta(t) = -a \frac{\partial \log P(0, t)}{\partial t} - \frac{\partial^2 \log P(0, t)}{\partial t^2} + \frac{\sigma_1^2}{2a} (1 - e^{-2at})$$

where: (26.27)

$$P(0, t) = \frac{1}{(1 + y(t))^t}$$

$y(t)$ = spot rate at tenor t .

We approximate the spot rate using cubic splines using the current spot curve as input. Cubic splines can overshoot, especially with non-uniform data. In such cases we may wish to choose monotone methods (see, for example, Duffy and Germani (2013)).

There are various use cases that can be used as a rounding-off project for this book. Some project topics and attention points are:

A1: Domain truncation and (26.13). Experiment with how to truncate to a bounded domain and define associated boundary conditions. Compare numerical and analytical solutions. As a special case, implement the one-factor Hull–White model along the same lines.

A2: It might be more effective to analyse and implement the one-factor Hull–White model from all viewpoints before moving to the two-factor model.

A3: Multiple PDE variants 1) domain transformation, 2) domain truncation, 3) PDE in canonical form, 4) PDE in conservative form (see Chapters 8 to 10).

A4: Applicability of the Fichera theory and energy inequalities (Chapters 11 and 12).

A5: There are many finite difference methods that can be applied to (26.13). We have already discussed MOL and ADE. Interesting schemes are Marchuk's two-cycle model (for accuracy) and Strang splitting to split the diffusion and convection terms.

A6: Transforming (26.13) in such a way that we produce monotone schemes with constant meshes as introduced in section 26.2.

A7: Using CSM to compute sensitivities based on the analytic Hull–White formula. You will need to complexify the function.

A8: (probably a research area) Apply the CSE method to (26.13) to compute the Hull–White sensitivities (see Chapter 17). Are the resulting PDEs well-posed, and do they have a unique solution? Are the PDEs coupled, and how will we approximate them numerically?

A9: SDE and stochastic flow-based sensitivity analysis (Kunita (1990), Glasserman (2004)). Consider the SDE:

$$\begin{aligned} dX &= a(\theta, X)dt + b(\theta, X)dW, t > 0 \\ X_0 &= x \\ (X &= X(t), W = W(t)). \end{aligned} \tag{26.28}$$

Define the sensitivity:

$$Y(t) = \frac{\partial X(t)}{\partial \theta} \quad (\theta \text{ is a parameter}) \tag{26.29}$$

which then satisfies the SDE:

$$\begin{aligned} dY &= [a'_\theta(\theta, X) + a'_x(\theta, X)Y] dt + [b'_\theta(\theta, X) + b'_x(\theta, X)Y]dW \\ Y_0 &= \frac{\partial X_0}{\partial \theta} \end{aligned} \tag{26.30}$$

where the primes denote standard derivatives. Simulate SDE (26.30).

A10: Convertible bonds with stochastic interest rates (Wilmott (2016)):

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \rho\sigma Sw \frac{\partial^2 V}{\partial S \partial r} + \frac{1}{2}w^2 \frac{\partial^2 V}{\partial r^2} + rS \frac{\partial V}{\partial S} + (u - w\gamma) \frac{\partial V}{\partial r} - rV = 0$$

$$\gamma(r, S, t): \text{Market price of risk}, -1 \leq \rho(r, S, t) \leq 1: \text{correlation.} \tag{26.31}$$

This problem is addressed in Evans (2015) for one-factor and two-factor convertible bonds using ADE, MOL and the Crank–Nicolson methods. The general conclusion is that ADE is faster than MOL and Crank–Nicolson in the one-factor case, but more time steps are needed to achieve the same level of accuracy as these latter methods.

26.5 SUMMARY AND CONCLUSIONS

This chapter is the epilogue.

Bibliography

- Abramowitz, M. and Stegun, I.A. (1965). *Handbook of Mathematical Functions*. Dover.
- Achdou, Y. and Pironneau, O. (2005). *Computational Methods for Option Pricing*. SIAM.
- Adams, R.A. (1975). *Sobolev Spaces*. Academic Press.
- Ahlberg, J.H., Nilson, E.N., and Walsh, J.L. (1967). *The Theory of Splines and Their Applications*. Academic Press.
- Andersen, L. and Andreasen, J. (2000). Jump-diffusion processes: Volatility smile fitting and numerical methods for option pricing. *Review of Derivatives Research* 4: 231—262.
- Andreasen, J. (2001). *Turbo Charging the Cheyette Model*. Bank of America, London.
- Aubin, J.P. (1980). *Approximation of Elliptic Boundary-Value Problems*. Dover.
- Ballabio, L. (2020). *Implementing Quantlib*. www.leanpub.com
- Barakat, H.Z. and Clark, J.A. (1966). On the solution of diffusion equation by numerical methods. *Journal of Heat Transfer* 88(4): 421–427.
- Bear, J. (1979). *Hydraulics of Groundwater*. McGraw-Hill.
- Bobisud, L. (1967). Second-order linear parabolic equations with a small parameter. *Arch. Ration. Mech. Anal.* 2: 385–397.
- Boehm, B. (1986). A spiral model of software development and enhancement, *ACM SIGSOFT Software Engineering Notes*, 11(4): 14–24.
- Box, M.J. (1966). A comparison of several current optimization methods and the use of transformations in constrained problems. *The Computer Journal* 9(1): 67–77.
- Boyd, J.P. (2000). *Chebyshev and Fourier Spectral Methods*. Dover.
- Brauer, F. and Nohel, J.A. (1969). *Qualitative Theory of Ordinary Differential Equations*. Dover.
- Brennan, M.J. and Schwartz, E.S. (1977). The valuation of American put options. *The Journal of Finance* 32(2): 449–462.
- Brigo, D. and Mercurio, F. (2006). *Interest Rate Models: Theory and Practice with Smile, Inflation and Credit*, second edition. Springer.
- Broadie, M., Detemple, J., Ghysels, E. and Torres, O. (2000). American options with stochastic dividends and volatility: A nonparametric investigation. *Journal of Econometrics* 94(1–2): 53–92.
- Bronson, R. (1989). *Matrix Operations*. Schaum's Outline Series.
- Bronson, R. and Naadimuthu, G. (1997). *Operations Research*. Schaum's Outline Series.
- Brooks, F. (1995). *The Mythical Man-Month*. Addison-Wesley.
- Buchova, Z., Ehrhardt, M. and Guenther, M. (2015). Alternating direction explicit methods for convection diffusion equations. *Acta Math. Univ. Comenianae* 84(2): 309–325.
- Campbell, L.J. and Yin, B. (2007). *On the Stability of Alternating-Direction Explicit Methods for Advection-Diffusion Equations*. Wiley Interscience.
- Carmona, R. and Durleman, V. (2003). Pricing and hedging spread options. *SIAM Review* 45(4): 627–685.
- Carr, P. (2001). Deriving derivatives of derivative securities. *Journal of Computational Finance* 4(2): 5–30.
- Carr, P. and Madan, D. (1998). Option valuation using the fast Fourier transform. *J. Comp. Finance* 2: 61–73.

- Cen, Z., Le, A. and Xu, A. (2013). An alternating direction implicit difference scheme for pricing Asian options. *Journal of Applied Mathematics* (Hindawi Publishing Corporation). DOI: 10.1155/2013/605943
- Chapman, A. (1984). *Heat Transfer*. Macmillan.
- Chen, Y. (2017). Numerical Methods for Pricing Multi-Asset Options. MSc thesis. University of Toronto.
- Chihara, T.S. (1978). *An Introduction to Orthogonal Polynomials*. Dover.
- Chu, P. and Fan, C. (1998). A three-point combined compact difference scheme. *Journal of Computational Physics* 140(2): 370–399.
- Clewlow, L. and Strickland, C. (1998). *Implementing Derivatives Models*. Chichester: John Wiley and Sons.
- Cont, R. and Tankov, P. (2000). *Financial Modelling with Jump Processes*. Chapman and Hall.
- Conte, S.D. and de Boor, C. (1981). *Elementary Numerical Analysis*. McGraw-Hill.
- Courant, R. and Hilbert, D. (1968). *Methoden der mathematischen Physik II* (zweite Auflage). Springer.
- Cox, J.C. and Rubinstein, M. (1985). *Options Markets*. Prentice-Hall.
- Cox, J.C., Ingersoll, J.E. and Ross, S.A. (1985). An intertemporal general equilibrium model of asset prices, *Econometrica* 53(2): 363–384.
- Craig, I.J.D. and Sneyd, A.D. (1988). An alternating direction implicit scheme for parabolic equations with mixed derivatives. *Comput. Math. Applic.* 16(4): 341–350.
- Crank, J. (1984). *Free and Moving Boundary Problems*. Clarendon.
- Cryer, C. (1979). Successive overrelaxation methods for solving linear complementarity problems arising from free boundary value problems. Presented at a seminar held in Pavia (Italy), September–October 1979.
- Dahlquist, G. (1956). Convergence and stability in the numerical integration of ordinary differential equations. *Math. Scand.* 4(1): 33–53.
- Dahlquist, G. and Björck, A. (1974). *Numerical Methods*. Dover.
- Davidson, A. and Levin, A. (2014). *Mortgage Valuation Methods*. Oxford University Press.
- Davis, T.A. (2006). *Direct Methods for Sparse Linear Systems*. Philadelphia: SIAM.
- De Allen, D. and Southwell, R. (1955). Relaxation methods applied to determine the motion, in two dimensions, of a viscous fluid past a fixed cylinder. *Quart. J. Mech. Appl. Math.* 8(2): 129–145.
- Derman, E. and Kani, I. (1996). The ins and outs of barrier options: Part 1. *Derivatives Quarterly* (Winter): 55–67.
- Derman, E. and Kani, I. (1997). The ins and outs of barrier options: Part 2. *Derivatives Quarterly* 3 (Winter): 73–80.
- Drezner, Z. (1978). Computation of the bivariate normal integral. *Mathematics of Computation* 32(141): 277–279.
- Duchateau, P. and Zachman, D.W. (1986). *Partial Differential Equations*. Schaum Outline Series.
- Duffie, D. and Kan, R. (1996). A yield-factor model of interest rates. *Mathematical Finance* 6(4): 379–406.
- Duffy, D.J. (1980). Uniformly Convergent Difference Schemes for Problems with a Small Parameter in the Leading Derivative. PhD thesis. Trinity College Dublin.
- Duffy, D.J. (2004). A critique of the Crank–Nicolson scheme, strengths and weaknesses for financial engineering, *Wilmott Magazine* (July).
- Duffy, D.J. (2004a) *Domain Architectures* Chichester: John Wiley and Sons.
- Duffy, D. J. (2006). *Finite Difference Methods in Finance Engineering*. Chichester: John Wiley and Sons.

- Duffy, D.J. (2009a) Unconditionally stable and second-order accurate explicit finite Difference schemes using domain transformation: Part I one-factor equity problems. *SSRN*. <https://ssrn.com/abstract=1552926>.
- Duffy, D. J. (2018). *Financial Instrument Pricing Using C++*, second edition. Chichester: John Wiley and Sons.
- Duffy, D.J. and Germani, A., (2013). *C# for Financial Markets*. Chichester: John Wiley and Sons.
- Duffy, D.J. and J. Kienitz (2009). *Monte Carlo Frameworks in C++*. Chichester: John Wiley and Sons.
- Dunne, R.K., O'Riordan, E. and Shishkin, G. (2009). Fitted mesh numerical methods for singularly perturbed elliptic problems with mixed derivatives. *IMA Journal of Numerical Analysis* 29(3): 712–730.
- Evans, G. (2015). An Analysis of Finite Difference Methods for Solving One-Factor and Two-Factor PDEs for valuing Convertible Bonds. MSc thesis. University of Birmingham.
- Fang, F. and Oosterlee, W. (2008). A novel pricing method for European options based on Fourier-cosine series expansions. *SIAM Journal on Scientific Computing* 31(2): 826–848.
- Feller, W. (1951). Two singular diffusion problems. *Annals of Mathematics*, second series, 54(1): 173–182.
- Fichera, G. (1956). Sulle equazioni differenziali lineari ellittico-paraboliche del secondo ordine, *Atti Accad. Naz. Lincei. Mem. CI. Sci. Fis. Mat. Nat. Ser. 8* (5): 1–30. MR 19, 658; 1432.
- Friedman, A. (1976). *Stochastic Differential Equations and Applications*. New York: Dover.
- Friedman, A. (1982). *Variational Principles and Free-Boundary Problems*. New York: Dover.
- Friedman, A. (1992). *Partial Differential Equations of Parabolic Type*. New York: Dover.
- Friedman, A. (1999). *Differential Games*. New York: Dover.
- Friedrichs, K.O. (1958). Symmetric positive linear differential equations. *Comm. Pure Appl. Math.* 11: 333–418.
- Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns*. Addison-Wesley.
- Gan, X. and Xu, D. (2020). On the convergence of a Crank–Nicolson fitted finite volume method for pricing American bond options. (*Hindawi Mathematical Problems in Engineering*. | Article ID 1052084. <https://doi.org/10.1155/2020/1052084>)
- Geiser, J. (2014). Recent advances in splitting methods for multiphysics and multiscale: Theory and applications. *Journal of Algorithms & Computational Technology* 9(1): 65–93.
- Geman, H. (2005). *Commodities and Commodity Derivatives*. Wiley.
- Genz, A. (2004). Numerical computation of rectangular bivariate and trivariate normal and t probabilities. *Statistics and Computing* 14: 251–260.
- Gershgorin, S. (1931). Über die Abgrenzung der Eigenwerte einer Matrix. *Izv. Akad. Nauk SSSR Ser. Mat.*, 7: 749–754; 16, 22, 25.
- Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering*. Springer.
- Godunov, S. and Riabenki, V.S. (1987). *Difference Schemes, An Introduction to the Underlying Theory*. Amsterdam: North-Holland.
- Goldberg, S. (1986). *Introduction to Difference Equations*. New York: Dover.
- Golden, S. (1965). Lower bounds for the Helmholtz function. *Phys. Rev.* 137(4B): B1127–B1128.
- Golub, G.H., and van Loan, C.F. (1996). *Matrix Computations*. John Hopkins University Press.
- Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning*. MIT Press.
- Gourlay, A.R. (1970). Hopscotch: A fast second-order partial differential equation solver. *J. Inst. Math. Applic.* 6(4): 375–390.
- Gourlay, A.R. and Morris, J.L. (1980). The extrapolation of first order methods for parabolic partial differential equations II. *SIAM. J. Numer. Anal.* 17(5, October): 641–655.
- Greenspan, D. (1966). *Introductory Numerical Analysis of Elliptic Boundary Value Problems*. New York: Harper & Row.

- Griewank, A. and Walther, A. (2008). *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM.
- Gustafsson, B. (1975). The convergence rate for difference approximations to mixed initial boundary value problems. *Math. Comp.* 29: 396–406
- Haaser, N.B. and Sullivan, J.A. (1991). *Real Analysis*. Dover.
- Hageman, L.A. and Young, D.M. (1981). *Applied Iterative Methods*. New York: Dover.
- Haug, A. N. and Arsenin, V. Y. (1977). *Solutions of Ill-Posed Problems*. New York: Winston.
- Haug, E. (2007). *The Complete Guide to Option Pricing Formulas*. McGraw-Hill.
- Henrici, P. (1962). *Discrete Variable Methods in Ordinary Differential Equations*. John Wiley.
- Heston, S.L. (1993). A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Rev. Financ. Stud.*, 6 (2): 327–343.
- Hildebrand, F.B. (1974). *Introduction to Numerical Analysis*. Dover.
- Hille, E. and Phillips, R.S. (1975). *Functional Analysis and Semi-Groups*. American Mathematical Society.
- Hochstadt, H. (1964). *Differential Equations*. Dover.
- In 't Hout, K.J. and Foulon, S. (2010). ADI finite difference schemes for option pricing in the Heston model with correlation. *International Journal of Numerical Analysis and Modeling* 7(2): 393–320.
- Hull, J.C. (2006). *Options, Futures and Other Derivatives*. Prentice Hall.
- Huyakorn, P.S. and Pinder, G.F. (1983). *Computational Methods in Subsurface Flow*. Academic Press.
- Ikeda, M. and Kunimoto, M. (1992). Pricing options with curved boundaries. *Mathematical Finance* 2: 275–298,
- Il'in, A.M. (1969). Differencing scheme for a differential equation with a small parameter affecting the highest derivative. *Mat. Zam.* 6:237–248.
- Il'lin, A.M., Kalashnikov, A.S. and Oleinik, O.A. (1962). Linear equations of the second order of parabolic type (translation), *Russian Mathematical Surveys* 17(3): 1–143.
- Ingersoll, J.E. (1987). *Theory of Financial Decision Making*. Rowman & Littlefield.
- Isaacson, E. and Keller, H. (1966). *Analysis of Numerical Methods*. New York: John Wiley & Sons.
- Jamshidian, F. (1997). A note on analytical valuation of double barrier options. Sakura Global Capital working paper.
- Jankowski, T. (2002). Differential equations with integral boundary conditions. *Journal of Computational and Applied Mathematics* 147(1): 1–8.
- Jeong, D. and Junseok Kim, J. (2013). A comparison study of ADI and operator splitting methods on option pricing models. *Journal of Computational and Applied Mathematics* 247: 162–171.
- Kangro, R. and Nicolaides, R. (2000). Far field boundary conditions for Black–Scholes equations. *SIAM J. Numer. Anal.* 38(4): 1357–1368.
- Keller, H. (1968). *Numerical Methods for Two-Point Boundary-Value Problems*. Waltham: Blaisdell Publishing Company.
- Keller, H. (1971). A new difference scheme for parabolic problems. In B. Hubbard (ed.), *Numerical Solution of Partial Differential Equations-II*, 327–350. New York: Academic Press.
- Keller, H. (1992). *Numerical Methods for Two-Point Boundary Value Problems*. Dover.
- Kellogg, O.D. (1953). *Foundations of Potential Theory*. Dover.
- Kellogg, R.B. (1964). An alternating direction method for operator equations. *Journal of the Society for Industrial and Applied Mathematics* 12(4): 848–854.
- Kloeden, P. and Platen, E. (1995). *Numerical Solution of Stochastic Differential Equations*. Berlin: Springer.
- Kloeden, P., Platen, E. and Schurz, H. (1997). *Numerical Solution of SDE Through Computer Experiments*. Berlin: Springer.
- Kohl-Landgraf, P. (2007). *PDE Valuation of Interest Rate Derivatives*. Books on Demand GmbH.

- Kolmogorov, A.N. and Fomin, S.V. (1961). *Elements of the Theory of Functions and Functional Analysis*. Dover.
- Kreider, D.L., Kuller, R.G., Ostberg, D.R. and Perkins, F.W. (1966). *An Introduction to Linear Analysis*. Addison-Wesley.
- Kreiss, H.O. and Lorenz, J. (2004). *Initial-Boundary Value Problems and the Navier-Stokes Equations*. SIAM. (<https://pubs.siam.org/doi/pdf/10.1137/1.9780898719130.fm>)
- Kunita, H. (1990). *Stochastic Flows and Stochastic Differential Equations*. Cambridge University Press.
- Ladyženskaja, O.A., Solonnikov, V.A. and Ural'ceva, N.N. (1988). *Linear and Quasi-Linear Equations of Parabolic Type*. American Mathematical Society.
- Lambert, J.D. (1991). *Numerical Methods for Ordinary Differential Equations*. John Wiley and Sons.
- Lamperti, J. (1962). Semi-stable stochastic processes. *Transactions of the American Mathematical Society* 104(1): 62–78.
- Landau, H.G. (1950). Heat conduction in a melting solid. *Quart. Appl. Math.* 8: 81–94.
- Laney, C.B. (1998). *Computational Gasdynamics*. Cambridge University Press.
- Larkin, B.M. (1964). Some stable explicit difference approximations to the diffusion equation, *Math. Comp.* 18: 196–202.
- Larsson, S., Thomée, V. and Wahlbin, L.B. (1998). Numerical solution of parabolic integro-differential equations by the discontinuous Galerkin method. *Math. Comp.* 67: 45–71.
- Lawson, J.D. and Morris, J.L. (1978). The extrapolation of first order methods for parabolic partial differential equations, *I SIAM. J. Numer. Anal.* 15(6, December): 1212–1240.
- Lax, P.D. and Wendroff, B. (1960). Systems of conservation laws. *Communications in Pure and Applied Mathematics*. 13(2): 217–237.
- Leung, S. and Osher, S. (2005). Alternating direction explicit (ADE) scheme for time-dependent evolution equations. Preprint UCLA June 9. Working Paper. <https://www.math.ust.hk/~masyleung/Reprints/leuosh05.pdf>
- Levin, A. and Duffy, D. (2001). Two-factor Gaussian term structure: Analytics, historical fitted and stable fitted finite difference pricing scheme. NYU Seminar, September 2001.
- Lewis, A.L. (2000). *Option Valuation Under Stochastic Volatility*. Newport Beach, CA: Finance Press.
- Lewis, A. L. (2016). *Option Valuation Under Stochastic Volatility II*. Newport Beach, CA: Finance Press.
- Liniger, W. and Willoughby, R.A. (1970). Efficient integration methods for stiff systems of ordinary differential equations. *SIAM J. Numer. Anal.* 7(1): 47–66.
- Lions, J.L. (1971). *Optimal Control of Systems Governed by Partial Differential Equations*. Springer.
- Lotka, A.J. (1956). *Elements of Mathematical Biology*. Dover.
- Lucic, V. (2012). Boundary conditions for computing densities in hybrid models via PDE methods. *Stochastics* 84(5–6): 705–718.
- Lyness, J.N. and Moler, C. (1967). Numerical differentiation of analytic functions. *SIAM Journal on Numerical Analysis* 4(2): 202–210.
- Ma, K. and Forsyth, P.A. (2017). An unconditionally monotone numerical scheme for the two-factor uncertain volatility model. *IMA Journal of Numerical Analysis*, 37(2): 905–944.
- Marchuk, G. (1982). *Methods of Numerical Mathematics*. Springer.
- Marchuk, G., Rusakov, A.S., Zalesny, V.B. and Diansky, N.A. (2005). Splitting numerical techniques with applications to the high resolution simulation of the Indian Ocean circulation. *Pure Appl. Geophys* 162(8): 1407–1429.
- Marchuk, G. I. and Shaidurov, V.V. (1983). *Difference Methods and Their Extrapolations*. New York: Springer-Verlag.

- Margrabe, W. (1978). The value of an option to exchange one asset for another. *Journal of Finance* 33(1): 177–186.
- Matus, P. (2002). *The Maximum Principle and Some of Its Applications*. De Gruyter. Published online.
- Merton, R. (1976). Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics* 3(1–2): 125–144.
- Meyer, G.H. (2006). The Black Scholes Barenblatt equation for options with uncertain volatility and its application to static hedging, *International Journal of Theoretical and Applied Finance* 9(5): 673–703.
- Meyer, G.H. (2015). *The Time-Discrete Method of Lines for Options and Bonds*. World Scientific.
- Mil'shtein, G. N. (1974). Approximate integration of stochastic differential equations, *Teor. Veroyatnost. i Primenen* 19(3): 583–588.
- Moler, C. and Van Loan, C. (2003). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review* 45(1): 3–49.
- Mori, M. and Sugihara, M. (2001). The double-exponential transformation in numerical analysis *Journal of Computational and Applied Mathematics* 127(1–2): 287–296.
- Morton, K. (1996). *Numerical Solution of Convection-Diffusion Equations*. London: Chapman & Hall.
- Morton, K.W. and Mayers, D.F. (1996). *Numerical Solution of Partial Differential Equations*. Cambridge University Press.
- Mun, J. (2002). *Real Options Analysis*. John Wiley & Sons, Inc.
- Nehari, Z. (1975). *Conformal Mapping*. Dover.
- Nocedal, J. and Wright, S. (2006). *Numerical Optimisation*. Springer.
- Øksendal, B. (1998). *Stochastic Differential Equations*. Springer.
- Oleinik, O.A. and Radkevic, E.V. (1973). *Second Order Equations with Nonnegative Characteristic Form*, translated from Russian by Paul C. Fife. American Mathematical Society.
- Osgood, W.F. (1966). *Topics in the Theory of Functions of Several Complex Variables*. Dover.
- Patie, P. and Winter, C. (2008). First exit time probability for multidimensional diffusions: A PDE-based approach. *Journal of Computational and Applied Mathematics* 222(1): 42–53.
- Peaceman, D.W. (1977). *Fundamentals of Numerical Reservoir Simulation*. Elsevier.
- Pealat, G. and Duffy, D. (2011). The alternating direction explicit (ADE) method for one-factor problems, *Wilmott Magazine* (July).
- Perko, L. (2001). *Differential Equations and Dynamical Systems*. Springer.
- Petrovsky, I.G. (1991). *Lectures on Partial Differential Equations*. New York: Dover.
- Piacsek, S.A. and Williams, G.P. (1970). Conservation properties of convection difference schemes. *Journal of Computational Physics* 6(3): 392–405.
- Pólya, G. (1990). *How to Solve It*. Penguin Books.
- Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. (2002). *Numerical Recipes in C++*. Cambridge. University Press.
- Rannacher, R. (1984). Finite element solution of diffusion problems with irregular data. *Numerische Mathematik* 43: 309–327.
- Rasmussen, L. (2016). Computational Finance – On the Search for Performance. PhD thesis. University of Copenhagen.
- Ren, Y., Madan, D. and Qian, M.Q. (2007). Calibrating and pricing with embedded local volatility models. *Risk* (September).
- Roache, P. J. (1998). *Fundamentals of Fluid Dynamics*. Albuquerque: Hermosa Publishers.
- Roberts, K.V. and Weiss, N.O. (1966). Convective difference schemes. *Mathematics of Computation*. 20(94, Apr.): 272–299.
- Robinson, M. (2019). Sensitivities: A Numerical Approach. Master's degree thesis. University of Birmingham.

- Ron, U. (2000). A practical guide to swap curve construction. Working paper, Bank of Canada.
- Roos, H.G. (2019). Monotone discretization of elliptic problems with mixed derivatives on anisotropic meshes: a counterexample. <https://arxiv.org/pdf/1901.05321.pdf>.
- Roscoe, D.F. (1975). New methods for the derivation of stable difference representations for differential equations. *J. Inst. Math. Applic.* 16(3): 291–301.
- Rothe, E. (1930). Zweidimensionale parabolische Randwertaufgaben als Grenzfall eindimensionaler Randwertaufgaben, *Mathematische Annalen* 102: 650–670. doi: 10.1007/BF01782368 | MR 1512599
- Rudin, W. (1964). *Principles of Mathematical Analysis*. McGraw-Hill.
- Rudin, W. (1970). *Real and Complex Analysis*. McGraw-Hill.
- Rynne, B.P. and Youngson, M.A. (2000). *Linear Functional Analysis*. Springer.
- Saad, Y. (1996). *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company.
- Saad, Y. and Schultz, M.H. (1986). GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear system. *SIAM J. Sci. Stat. Comput.*, 7: 856–869.
- Samarskii, A.A., Matus, P.P., Mazhukin, V.I. and Mozolevski, I.E. (2002). Monotone difference schemes for equations with mixed derivatives. *Computers and Mathematics with Applications* 44(3–4): 501–510.
- Saul'yev, V. K. (1964). *Integration of Equations of Parabolic Type by the Method of Nets*. Pergamon Press.
- Scales, L.E. (1985). *Introduction to Non-Linear Optimization*. London: Macmillan.
- Schölkopf, B. and Smola, A.J. (2002). *Learning with Kernels*. MIT Press.
- Sharp, N.J. (2006). Advances in Mortgage Valuation: An Option-Theoretic Approach. PhD thesis. Manchester University.
- Sheppard, R. (2007). Pricing Equity Derivatives under Stochastic Volatility: A Partial Differential Equation Approach. Master of Science thesis. University of the Witwatersrand, Johannesburg, South Africa.
- Shilov, G.E. (1977). *Linear Algebra*. Dover.
- Skorokhod, A. (1982). *Studies in the Theory of Random Processes*. Dover.
- Smith, G.D. (1978). *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford: Clarendon Press.
- Spiegel, M. (1959). *Vector Analysis*. Schaum's Outlines.
- Spiegel, M. (1999). *Complex Variables*. Schaum's Outlines.
- Squire, W. and Trapp, G. (1998). Using complex variables to estimate derivatives of real functions. *Siam Review* 40(1): 110–112.
- Stakgold, I. (1998). *Green's Functions and Boundary Value Problems*. New York: John Wiley & Sons.
- Stoer, J. and Bulirsch, R. (1980). *Introduction to Numerical Analysis*. New York: Springer-Verlag.
- Strang, G. (1969). On the construction and comparison of difference schemes. *SIAM Journal on Numerical Analysis* 5(3): 506–517.
- Strang, G. and Fix, G. (1973). *An Analysis of the Finite Element Method*. Prentice Hall.
- Stroud, A.H. (1974). *Numerical Quadrature and Solution of Ordinary Differential Equations*. Springer.
- Sun, H.W. and Li, L.Z. (2014). A CCD-ADI method for unsteady convection-diffusion equations. *Computer Physics Communications* 3(3): 790–797.
- Tannehill, J.C., Anderson, D.A. and Pletcher, R.H. (1997). *Computational Fluid Mechanics and Heat Transfer*. Taylor and Francis.
- Tavella, D. and Randall, C. (2000). *Pricing Financial Instruments, The Finite Difference Method*. New York: Wiley.
- Thomas, J.W. (1995). *Numerical Partial Differential Equations, Volume I: Finite Difference Methods*. Springer.

- Thomas, J.W. (1999). *Numerical Partial Differential Equations Volume II; Conservation Laws and Elliptic Equations*. Springer.
- Thomas, L.H. (1949). Elliptic Problems in Linear Difference Equations over a Network. Technical report, Watson Sci. Comput. Lab. Rept, New York: Columbia University.
- Thomée, V. (1963). A stable difference scheme for the mixed boundary problem for a hyperbolic, first-order system in two dimensions. *Journal of the Society for Industrial and Applied Mathematics* 10(2): 229–245.
- Tikhonov, A. N. and Arsenin, V. Y. (1977). *Solutions of ill-Posed Problems*. New York: Winston.
- Tolstov, G.P. (1962). *Fourier Series*. Dover.
- Topper, J. (2005). *Financial Engineering with Finite Elements*. Chichester: John Wiley & Sons.
- Towler, B.F. and Yang, R.Y.K. (1978). Numerical stability of the classical and the modified Saul'yev's finite-difference methods. *Computers and Chemical Engineering* 2(1): 45–51.
- Tricomi, F.G. (1985). *Integral Equations*. Dover.
- Van Moerbeke, P. (1974). On the optimal stopping and free boundary problems. *Rocky Mountain Journal of Mathematics* 4(3): 539–578. doi: 10.1216/RMJ-1974-4-3-539
- Varga, R.S. (1962). *Matrix Iterative Analysis*. Englewood Cliffs, NJ: Prentice Hall.
- Vichnevetsky, R. and Bowles, J.B. (1982). *Fourier Analysis of Numerical Approximations of Hyperbolic Equations*. Siam Publications.
- Vladimirov, V.S. (1966). *Methods of the Theory of Functions of Many Complex Variables*. Dover.
- Wahba, G. (1990). *Spline Models for Observational Data*. Society for Industrial and Applied Mathematics.
- Watson, G.N. (2006). *A Treatise on the Theory of Bessel Functions*. Cambridge University Press.
- Wernick, A. (2015). A Comparison of the Alternating Direction Explicit, Operator Splitting and Alternating Direction Implicit Methods when solving the Two-Factor Black–Scholes Partial Differential Equation. MSc thesis, University of Birmingham.
- West, G. (2004). Better Approximation to Cumulative Normal Functions. Report, Financial Modelling Agency.
- Widder, D.V. (1989). *Advanced Calculus*. Dover.
- Wilmott, P. (2006). *Paul Wilmott on Quantitative Finance*. Wiley.
- Wilmott, P., Lewis, A., and Duffy, D. (2014). Modelling volatility and valuing derivatives under anchoring. *Wilmott Magazine* 73: 48–57.
- Yanenko, N.N. (1971). *The Method of Fractional Steps*. Springer.
- Zeemanian, A.H. (1987). *Generalized Integral Transformations*. Dover.
- Zhang, K. and Yang, X.Q. (2017). Pricing European options on zero-coupon bonds with a fitted finite volume method. *International Journal of Numerical Analysis and Modeling* 14(3): 405–418.
- Zhang, P. (1998). *Exotic Options: A Guide to the Second Generation Options*, second edition. Singapore: World Scientific.

Index

- A-stable** 38
absorbing barrier conditions 219
access control systems (ACS) 437
activation function 48
AD *see* automatic differentiation
Adams–Bashforth scheme 36
addition, vectors 65
addition formula 367
ADE method *see* alternating direction explicit method
adjoint equations 132–135
advection
 alternating direction explicit method 356–358
 first-order partial differential equations
 177–178
 Fourier transform 229–230
 Roberts–Weiss method 357–358, 455
 Towler–Yang approximation 356–357, 455
affine term structure 46
alternating direction explicit (ADE) method
 120–125, 349–364
 American options 124–125, 400–405
 applicability 351–352
 Asian options 455–456
 background 351
 Barakat and Clark method 120–125, 353–354
 Black–Scholes equation 359–360
 boundary conditions 360–361
 call pay-off at far field 362
 conditional consistency 362
 constant elasticity of variance 123
 convection equations 356–358
 convection-dominated scenario 122–123
 convection–diffusion equations 358–362
 diffusion equations 352–356
 European put option 121–122
 general formulation 362–363
 Larkin method 355
 option sensitivities 124
 overview 351–352
 Roberts–Weiss method 357–358, 455
 Saul’yev method 354–355
 stress testing 122
Towler–Yang approximation 356–357, 455
two-dimensional diffusion 355–356
alternating direction implicit (ADI) method
 328–330, 339–340
 Asian options 456–459
 modern variations 458–459
 vs. splitting 419–421
American options
 alternating direct explicit method 400–405
 alternating direction explicit method 124–125
 boundary conditions 442–443
 early exercise 391
 free/moving boundary value problems 391,
 400–405
Method of Lines 384
Python coding 400–405
sensitivity analysis 384
analytic functions 154–155
applicability, alternating direction explicit method 351–352
approximate factorisation 334–336
arithmetic averaging 448–449
artificial diffusion 454
Asian options 241, 447–459
 alternating direction explicit method 455–456
 alternating direction implicit method
 456–459
 arithmetic averaging 448–449
 challenges 449–450
 Fichera theory 451
 geometric averaging 449
 Lax–Wendroff scheme 454
 Method of Lines 452–453, 455
 monotone upwind scheme 455
 similarity reduction 451–452
 upwinding 453, 455
 associative 65
 asymptotic behaviour, Landau symbols
 10–11
automatic differentiation (AD) 314–319
C++ 318–319
 dual numbers 317–318
 modes 316

- autonomous 35
 auxiliary equation 254
 average value options *see* Asian options
- B&C *see* Barakat and Clark method
 backward differencing 245, 272
 backward induction steps 115
 backward in time, backward in space (BTBS)
 scheme 245–246, 252–253
 backward-difference formula 283
 backwards heat equation 185–187
 Banach spaces 408–410
 Banach's fixed point theorem 19
 band matrices 89, 94
 Barakat and Clark (B&C) method 120–125,
 353–354, 455
 bases, metric spaces 69
 Bernoulli ordinary differential equation 45–46
 Big O notation 10–11
 bivariate normal distribution (BVN) 216,
 427–434
 computing via hyperbolic PDE 430–433
 cumulative 429
 first-order partial differential equations
 430–433
 Black–Scholes equation 109–126
 adjoint equation 133
 alternating direction explicit method
 121–125, 359–360
 American option early exercise 124–125
 constant elasticity of variance 123, 146–147
 continuous sensitivity equation 310–313
 convection-dominated scenario 122–123
 Crank–Nicolson method 111–114
 European put option 121–122
 exponential fitting 253–254
 fully implicit method 110–111, 114
 Greeks 279–282
 linearity boundary condition 241
 option sensitivities 124
 trinomial method 115–119
- BLAS 90–91
 block tridiagonal matrices 89, 97–99
 boundary conditions 159
 alternating direction explicit method 360–361
 Cox–Ingersoll–Ross model 470–471
 cubic splines 291–292
 double-barrier options 376–377
 Heston model 173–176
 multi-asset options 435–436, 442–443
 multidimensional Fokker–Planck equation
 219–220
 splitting 339–340
- boundary layers 114
 boundary surface 389
 boundary value problems 87–108
 BLAS 901
 Cholesky decomposition 94
 conjugate gradient method 105–106
 direct methods, linear systems 92–94
 elliptic PDEs 106–107, 153–162
 free 387–405
 Gauss–Seidel method 104–105
 iterative methods 89, 103–107
 Jacobi method 104
 Laplace equation 154–159
 linear complementarity problem 106
 LU decomposition 92–94, 96–97
 matrix types 88–89
 moving 387–405
 projective successive over-relaxation 106
 successive over-relaxation 105
 tridiagonal systems 94–99
 two-point 99–103
 bounded domains 143–151
 constant elasticity of variance 146–147
 constrained optimisation 145
 double-exponential rule 145
 hotspots 148
 integration of functions 144–145
 log transformation 149–150
 truncation 148–149
 box scheme 27, 247
 BTBS *see* backward in time, backward in space
 scheme
 builder 438, 440, 444
 BVN *see* bivariate normal distribution
- C++
 American spread options 440–442
 automatic differentiation 318–319
 complementary error function 539
 complex step method 300–301, 484–493
 Cox–Ingersoll–Ross model 463–466
 cubic splines 289
 double-barrier options 378–384
 error functions 539
 European option sensitivity analysis 381–384
 Faddeeva function 483–487
 Jacobian matrices 488–490
 Lie–Trotter splitting 439
 Marchuk's two-cycle splitting method 439
 matrix ODEs 59–62
 non-linear solvers 492–493
 one-factor stochastic differential equation
 205–209

- Riccati equation 56–59
Rosenbrock functions 490–491
spread options 434
tridiagonal matrices 439
two-factor Hull–White model 481–483
 C_0 semigroups 408–409
calculus
 branches of 3–4
 chain rules 9
 differential 8–11
 implicit forms 13–14
 Landau symbols 10–11
 metric spaces 15–16
 partial derivatives 11–13
 Taylor’s theorem 9–10
call options
 alternating direction explicit method 362
 Heston model 174
 Method of lines 381–384
callable objects 429–430
canonical form, partial differential equations 147–148
Cauchy sequences 16–17, 409–410
Cauchy–Euler equation 163–165, 426–427
Cauchy–Goursat theorem 296–297
Cauchy–Riemann (CR) equations 155–156
Cauchy’s integral formula 294
Cauchy’s residue theorem 298
Cayley transform 84
CDR *see* convection-diffusion reaction
centred approximation 25
centred-difference formula 283
chain (discrete state) processes 48, 196
chain rule for differentiation 150
chain rules 9
charm 282, 311
Cholesky decomposition 83, 94
chooser options 433–434
chord length 293
CIR *see* Cox–Ingersoll–Ross
classes, discontinuous functions 7–8
closed curves 293
colour 311
column matrix 78
Combined Compact Difference (CCD) method 458–459
commutative 65
compatibility conditions, heat equation 189
complementary error function 428
complementary function 37
complete metric space 17
complex function theory 292–299
Cauchy–Goursat theorem 296–297
Cauchy’s integral formula 297
Cauchy’s residue theorem 298
curves 293
Gauss’s mean value theorem 299
Laurent’s theorem 295
regions 293–294
Taylor’s theorem 294–295
complex line integral 293
complex step method (CSM) 299–302, 310, 483–493
C++ code 300–301, 484–493
extended 488–492
Faddeeva function 483–487
non-linear solvers 492–493
component form, Method of Lines 372–373
component splitting 413
composite functions 9
composition 74
compound interest 367
computational graph 315
conditionally stable 234
conjugate functions 155–156
conjugate gradient method 105–106
consistency 33–34, 36, 237–238
constant elasticity of variance (CEV) 123, 146–147
constrained optimisation 145
context diagrams 438
continuity
 formal definition 5–6
 uniform 6–7, 18
continuity at a point 17–18
continuous curves 293
continuous functions 4–8, 17–19
continuous sensitivity equation (CSE) 305–313
 Black–Scholes equation 310–313
 initial value problem 306–307
 population dynamics 307–309
continuous time Markov chains 61
continuous-parameter/continuous time processes 48, 196
contour 293
contraction 18
convection *see* advection
convection-diffusion reaction (CDR) equations 249–276
 alternating direction explicit method 358–362
 auxiliary equation 254
 backward in time, backward in space scheme 252–253
boundary conditions 360–361

- convection-diffusion reaction (CDR) equations
(Continued)
- Crank–Nicolson scheme 253–254, 260, 261, 268, 271, 276
 - discrete Fourier transform 236–237
 - explicit Euler method 268, 271, 272–273, 276
 - exponential fitting 253–254, 257–258
 - forward in time, centred in space scheme 251–252
 - fundamental equations 351
 - Heston model 172–176
 - implicit Euler scheme 259, 271, 275
 - infinite domain 150
 - initial boundary value problems 260–263
 - maximum-minimum principles 189–190
 - numerical methods 250–257
 - Padé matrix approximation 269–274
 - principal symbols 131–132
 - semi-discretisation 264–269
 - spatial amplification errors 361–362
 - spatial stability 251–257
 - splitting 346–347
 - temporal stability 251–257
 - time-dependent 275–276
 - total symbol 131–132
 - convection-dominance 122–123, 252
 - convergence 35–36, 239, 257
 - radius of 366
 - cost of carry 280
 - Courant–Friedrichs–Lowy (CFL) condition 244–245, 454
 - covariance matrices 203–204
 - Cox–Ingersoll–Ross (CIR) model 462–471
 - boundary conditions 470–471
 - C++ coding 463–466
 - Fichera theory 171–172
 - finite difference methods 469–471
 - Fokker–Planck equation 220
 - initial boundary value problem 466
 - well-posedness 466–469
 - zero coupon bonds 242, 463–466
 - CR *see* Cauchy–Riemann
 - crack spread options 210
 - Crank–Nicolson scheme 27, 327–328
 - Black–Scholes equation 111–114
 - convective-diffusion reaction 253–254, 260, 261, 268, 271, 276
 - exponential fitting 253–254, 260
 - initial value problems 246
 - Method of Lines 374
 - Padé matrix approximation 271
 - stability 250–251
 - Cryer projective successive over-relaxation method 394
 - CSM *see* complex step method
 - cubic splines 286–292
 - boundary conditions 291–292
 - C++ code 289
 - option Greeks 290–291
 - sparse data 289–290
 - cumulative bivariate normal distribution 429
 - cumulative normal (Gaussian) distribution functions 280, 428–429
 - curves, complex function theory 293–294
 - data sources 438, 444
 - degenerate elliptic equations 135
 - delta 280, 285, 311
 - derivatives
 - definition 8
 - mixed 331–333, 478–483
 - see also* American options; Asian options; double-barrier options; European call options; European put options
 - DFT *see* discrete Fourier transform
 - diagonal matrix 78
 - diagonally dominant matrices 83, 97
 - differential calculus 8–11
 - diffusion
 - alternating direction explicit method 122–123, 352–356
 - artificial 454
 - Barakat and Clark method 353–354
 - Fourier transform 230–231
 - fundamental variants 351
 - Larkin method 355
 - of oxygen 390
 - Saul’ev method 354–355
 - stochastic differential equations 51
 - two-dimensional 355–356
 - see also* convection-diffusion reaction equations
 - dilation 70
 - dimensional splitting 413
 - Dirichlet condition 159
 - discontinuous functions 5, 7–8
 - discrete Fourier transform (DFT) 232–237
 - discrete maximum principle 28–29
 - discrete time approximations 32–37
 - consistency 33–34
 - stability 34

- discrete von Neumann stability criterion 235
discrete-parameter/discrete time processes (random sequences) 48, 196
discrete-state (chain) processes 48, 196
discretisation consistency 33–34, 36
Crank–Nicolson scheme 27
explicit Euler method 26, 51, 53, 57–59
exponential fitting 29–31, 37–39
Heun’s method 52, 53–54
implicit Euler method 27, 36
ordinary differential equations 25–39
predictor–corrector model 31
Richardson extrapolation 31–32
root condition 34–36
splitting as 413–414
spurious oscillations 28, 29, 101
stability 34, 36
stable difference schemes 28–29
Theta method 27, 37–38
trapezoidal method 28
two-point boundary value problems 99–101
discretisation error 284
dissipative 375
distance function 15
divergence 178–179
divergence (Gauss–Ostrogradsky) theorem 179
divided differences 25, 282–286
dividends, stochastic partial differential equation 140–141
domain transformation 358–359, 435
domain truncation 148–149, 427
dot product 178
double sweep method 94–96
double-barrier options boundary conditions 376–377
C++ coding 378–384
exponential fitting 377–378
Method of Lines 375–384
sensitivity analysis 381–384
double-exponential rule 145
Douglas–Rachford scheme 344
drift alternating direction explicit method 122–123
exponential affine term structures 46
stochastic differential equations 51
drift-adjusted predictor–corrector methods 202–203
D’Yakonov method 333–334
dynamic instability 252
early exercise, American options 124–125, 391
eigenvalues 72
elliptic operators 131
Gershgorin’s circle theorem 262–263
eigenvectors 72
elliptic equations adjoint equations 132–135
boundary conditions 159
boundary value problems 153–162
classification 135–138
degenerate 135
Fichera theory 168–178
harmonic functions 154–156
iterative solvers 107–108
Laplace equation 154–159
maximum–minimum principles 159–162
multi-asset options 139–140
multi-index notation 131–132
operator 131–132
separation of variables 156–158
stochastic dividends 140–141
elliptic operator 131
end-point singularities 144–145
energy estimates 166–167
energy inequalities 185, 468–469
energy-integral arguments 160–162
entire functions 154–155
epsilon-delta approach 5, 6
error function 428
Euclidean space 75
Euler–Maruyama method 199–200, 205–209
Euler’s identity 367
Euler’s number 366–367
European call options Heston model 174
Method of Lines 381–384
sensitivity analysis 381–384
European put options alternating direction explicit method 121–122
Heston model 175
existence theorems ordinary differential equations 43–45
stochastic differential equations 48–51
explicit difference schemes 26
explicit Euler method 26, 51, 53, 57–59, 212–213, 326
convection–diffusion reaction 268, 271, 272–273, 276

- explicit Euler method (*Continued*)
 interest rate models 473–474
 Method of Lines 374
 Padé matrix approximation 271, 272–273
- explicit finite difference method *see* forward in time, centred in space
- explicit solutions 39–40
- exponential affine 46
- exponential fitting 29–31, 37–39
 alternating direction explicit method 122–123
 convection-diffusion reaction equations 253–254, 257–258
- Crank–Nicolson scheme 260
 double-barrier options 377–378
- exponential function 10, 366
- exponential of a matrix 367–370
- extrapolation 31–32
- Faddeeva function 483–487
- far-field boundary 144, 148–149
- fast transients 37
- FDM *see* finite difference method
- feature space 75
- Feller condition 149, 469, 471
- FEM *see* finite element method
- Feynman–Kac formula 217
- Fibonacci recurrence relation 16–17
- Fichera function 169
- Fichera theory 163, 168–178
 Asian options 451
 Cox–Ingersoll–Ross 171–172
 Heston model 176
 multi-asset options 435–436
- fields 65
- finite difference method (FDM) 25–29, 223–319
 alternating direction explicit method 349–364
 automatic differentiation 314–319
 auxiliary equation 254–255
 backward in time, backward in space scheme 245–246, 252–253
 box scheme 247
 complex function theory 292–299
 complex step method 299–302
 consistency 237–238
 continuous sensitivity equation 305–313
 convergence 239, 257
 core concepts 227
 Cox–Ingersoll–Ross model 469–471
 Crank–Nicolson scheme 246, 253–254, 260, 261, 268, 271, 276
 cubic spline interpolation 286–292
 discrete Fourier transform 232–237
- divided differences 282–286
- explicit Euler method 268, 271, 272–273, 276
- exponential fitting 253–254, 257–258
- first-order PDEs 239–240, 239–248
- forward in time, backward in space scheme 244, 248
- forward in time, centred in space scheme 245, 251–252
- forward in time, forward in space scheme 244–245
- Fourier analysis 227–237
- free boundary value problems 387–406
- front-fixing 399
- Gershgorin's circle theorem 261–263
- Goursat PDE 216, 431–433
- Greeks 279–282, 285–286, 310–314
- implicit Euler method 259, 271, 275
- initial boundary value problems 260–263
- leapfrog scheme 246–247
- method of lines splitting 365–386
- moving boundary value problems 387–406
- multi-asset options 436–440
- notation 226
- Padé matrix approximation 269–274
- parabolic variational inequalities 392–399
- semi-discretisation 264–269
- sensitivity analysis 277–319
- simple explicit scheme 243–245
- software implementation 438–439, 444
- spatial stability 251–257
- splitting methods 323–348, 407–421
- stability 238–239, 251–257
- temporal stability 251–257
- time-dependent convection-diffusion problems 275–276
- von Neumann stability analysis 235
- finite dimensional vector spaces 63–72
- finite element method (FEM) 178
- first exit-time problems, stochastic differential equations 221–222
- first-order (Lie–Trotter) splitting 415
- first-order partial differential equations 114
- advection, Heston model 177–178
- bivariate normal distribution 430–433
- difficulties 242–243
- explicit scheme 243–245
- finite difference method 239–248
- initial boundary value problems 184–185
- fitted centred difference equation 256

- fitting factor 30
fixed point theorems 18–19
fixed strike 447
floating strike 447
flux (Neumann) condition 159, 160
Fokker–Planck equation (FPE) 218–220
forward differencing 245, 272
forward induction steps 115
forward mode, automatic differentiation 316
forward in time, backward in space (FTBS)
 scheme 244–245, 248
forward in time, centred in space (FTCS) scheme
 245, 251–252, 344–345
forward in time, forward in space (FTFS) scheme
 244–245
forward-difference formula 283
Fourier series, Weierstrass function 7
Fourier stability analysis 235
Fourier transform 227–237
 advection equation 229–230
 diffusion equation 230–231
 discrete 232–237
fourth-order Runge–Kutta method 52, 53
FPE *see* Fokker–Planck equation
Fredholm integral equations 186
free boundary value problems
 387–405
 American options 391, 400–405
 definition 388
 front-fixing 399–400
 melting ice 389–390, 392
 oxygen diffusion 390
 parabolic variational inequalities
 392–399
 front-fixing 399–400
FTBS *see* forward in time, backward in space
 scheme
FTCS *see* forward in time, centred in space
 scheme
FTFS *see* forward in time, forward in space
 scheme
full matrices 88–89
fully implicit method, Black–Scholes equation
 110–111, 114
function of a function 12–13
functions
 analytic 154–155
 composite 9
 conjugate 155–156
 continuous 4–8
 discontinuous 5, 7–8
 entire 154–155
 exponential 10
 implicit forms 13–14
 Landau symbols 10–11
 Taylor's theorem 9–10
gamma 285–286, 311
Gaussian error functions 428
Gauss–Ostrogradsky theorem 179
Gauss's mean value theorem 299
Gauss–Seidel method 104–105
generalisations
 alternating direction explicit method 362–363
 ordinary differential equations 24–25
generating polynomials 33
generator matrices 62
geometric averaging 449
Gershgorin's circle theorem 261–263
global properties 6–7
Golden Ratio 18
Goursat partial differential equation 216,
 431–433
gradient functions 178
graph theory 315–316
Greeks
 alternating direction explicit method 124
 continuous sensitivity equation 310–313
 cubic splines 290–291
 divided differences 285–286
 double-barrier options 381–384
 Faddeeva function 483–487
 higher-order 282, 310–311
 mixed 282, 311
 operator calculus 313–314
 principles 279–282
Greens first/second identities 180
Green's theorem 180
Gronwall's inequality 166, 467–468
harmonic conjugates 155–156
heat equation 120–125, 182
 alternating direction implicit method
 328–330
 Cauchy problem 410–411
 compatibility conditions 189
 D'Yakonov method 333–334
 front-fixing 399–400
 initial boundary value problem 182–184
 maximum-minimum principles 190
 melting ice 389–390
 method of lines 370–373
 predictor-corrector methods 337
 separation of variables 184
 smoothness conditions 188–189
 Soviet splitting 330–331

- heat equation (*Continued*)
 three-dimensional 332–339, 343–344
 two-dimensional 328–339
 uniqueness 193
 Yanenko scheme 331–333
- Heaviside function 5
- Hermitian matrices 81
- Heston model 172–178
 boundary conditions 173–176
 European call options 174
 European put options 175
 Fichera theory 176
 first-order advection PDEs 177–178
 stochastic volatility 211–214
- Heun's method 52, 53–54
- higher-order Greeks 282
- HMOL *see* horizontal method of lines
- homogeneous ordinary differential equations
 24–29
- hopscotch method 344–345
- horizontal method of lines (HMOL) 365
- hyperbolic partial differential equations *see*
 first-order partial differential equations
- hypercomplex numbers 317–318
- ice, melting 389–390, 392
- idempotent matrix 80
- identity matrix 78
- ill-posed problems 185–187
- implementation
 splitting 346
 see also C++; Python; software design
- implicit Euler method 27, 36, 327
 convection-diffusion reaction equations 259,
 271, 275
 interest rate models 474–475
 Method of Lines 374
 Padé matrix approximation 271
 implicit finite difference schemes 26
 implicit forms 13–14
 implicit Milstein scheme 201, 325
 incompatibility 189
 increment function 51
 index sets 196
 infinite domain 150
 infinitesimal generator 221
 inflow boundaries 177, 452
 inhomogeneous forcing term 22
 initial boundary value problems (IBVP)
 alternating direction implicit method
 328–330
 first-order hyperbolic equations 184–185
 first-order PDEs 240–241
- general linear 248
- heat equation 182–184
- parabolic variational inequalities 395–399
- Soviet splitting 330–331
- stability 260–263
- time-dependent, one-dimensional 190–192
- two-dimensional convection 343
- Yanenko scheme 331–333
- zero coupon bonds 466
- initial condition 22
- initial value problems (IVP) 22, 109
 advection equation 229
 continuous sensitivity equation
 306–307
 discretisation 25–29
 existence 43–45
 uniqueness 44–45
- inner product spaces 74–75
- integral equations 44–45, 144–145
- integral relations 178–180
- integrating factor method 359, 369
- integration of functions 144–145
- interest rate models 461–475
 Cox–Ingersoll–Ross 462–471
 explicit Euler method 473–474
 Feller condition 469, 471
 Heston 471–475
 implicit Euler method 474–475
 Thomée scheme 471
- interpolation, cubic splines 286–292
- invariant subspaces 70–71
- inverse heat equation 185–187
- involutory matrix 81
- inward unit normal 168
- irreducible matrices 83
- isomorphism 71
- iterative solvers 89, 103–107
 elliptic PDEs 106–107
 Gauss–Seidel method 104–105
 general methods 103–104
 Jacobi method 104
- Ito formula 215
- Ito–Milstein scheme 201
- IVP *see* initial value problems
- Jacobi matrices *see* tridiagonal matrices
- Jacobi method 104
- Jacobian determinants 14
- joint cumulative distribution functions (joint
 cdf) 215–216
- joint probability density functions (joint pdf)
 216
- Jordan curve 294

- k-step method 33
kernel 71, 75
Kolmogorov backward equation (KBE) 61–62, 220, 221
Kolmogorov equation 218
Kolmogorov forward equation 61, 218–220
Kolmogorov models 47
Kramp function 484–487
- L-matrices 84
Landau symbols 10–11
Laplace equation 12–13, 154–159
 harmonic functions 154–156
 maximum-minimum principles 158–159
properties 156–159
uniqueness 192–193
- Laplace operator 130
Larkin method 355
Laurent's theorem 295
Lax Equivalence Theorem 239, 414
Lax–Wendroff scheme 245, 454
LCM *see* lifecycle model
LCP *see* linear complementarity problem
leapfrog scheme 36, 246–247
Lie product formula 411
Lie–Trotter splitting 415, 439
lifecycle model (LCM) 438
line integral 293
linear algebra 87–92
linear complementarity problem (LCP) 106, 394
linear independence 68–69
linear transformations
 composition 74
 metric spaces 69–72
 scalar product 74
 sum of 74
linearity boundary condition 149, 241
Lipschitz condition 49, 197, 375
Lipschitz continuous functions 17–19
little o notation 10–11
local splitting error 412
log transformation 149–151, 426–427
logistic functions 48
Lotka–Volterra equations 47
lower triangular matrices 89
LU decomposition 92–94, 96–97
- M-matrices 84
magnification 70
magnitude of a vector 178
management information systems (MIS) 437, 444
- manufacturing (MAN) systems 437
Marchuk's two-cycle splitting method 416, 439
marginal distribution functions 216
Markov chains 61–62
 see also stochastic differential equations
mass matrix 398–399
matrices
 band 89, 94
 boundary value problems 87–108
 Cayley transform 84
 Cholesky decomposition 83, 94
 eigenvalues 262–263
 exponential of 367–370
 full 88–89
 fundamental properties 77–80
 Gershgorin's circle theorem 261–263
 Hermitian 81
 idempotent 80
 involutory 81
 irreducible 83
 L-type 84
 lower triangular 89, 91
 LU decomposition 92–94, 96–97
 M-type 84
 Metzler 84
 nilpotent 80–81
 non-negative 83
 normal 81
 operations 78–79
 orthogonal 82
 patterned 89
 positive definite 82–83
 skew-Hermitian 81
 sparse 89
 spectral norms 262
 symmetric 81
 Toeplitz 94, 263
 tridiagonal 89, 94–99, 288
 types 80–84
 unitary 82
 upper triangular 89, 91
 Z-type 84
matrix, definition 76–77
matrix addition 78–79
matrix decomposition 82
matrix multiplication 79
matrix operator splitting 24
matrix ordinary differential equations
 C++ 59–62
 operator splitting 24
matrix solvers 88–108
 BLAS 90–91
 block tridiagonal systems 97–99

- matrix solvers (*Continued*)
 Cholesky decomposition 94
 conjugate gradient method 105–106
 direct methods, linear systems 92–94
 double sweep method 94–96
 elliptic PDEs 106–107
 Gauss–Seidel method 104–105
 iterative 89, 103–107
 Jacobi method 104
 linear complementarity problem 106
 LU decomposition 92–94, 96–97
 matrix types 88–90
 non-linear 492–493
 projective successive over-relaxation 106
 successive over-relaxation 105, 106
 Thomas algorithm 96–97
 tridiagonal systems 94–99
 two-point boundary value problems 99–103
- matrix theory 73–86
 inner product spaces 74–75
 operations 78–79
 orthonormal basis 75
 types of matrices 80–84
- matrix trace 79
- matrix-matrix operations, BLAS 91–92
- matrix-vector operations, BLAS 90–91
- maximum-minimum principles 158–162, 478–481
- mediators 438, 439, 444
- melting ice 389–390, 392
- mesh functions 28
- mesh point 25, 28
- mesh points 199, 205
- mesh size 51
- Method of Lines (MOL) 362–363, 365–385
 American options 384
 Asian options 452–453, 455
 component form 372–373
 double-barrier options 375–384
 European call options 381–384
 exponential function 366–367
 exponential of a matrix 367–370
 one-dimensional heat equation 370–373
 semi-discretisation 368–369, 371–373
 semi-linear problems 373–375
 Theta method 372
 two-factor Hull–White model 481–483, 493–495
- metric spaces 15–19
 bases 69
 Cauchy sequences 16–17
 definition 67
 eigenvalues/eigenvectors 72
- invariant subspaces 70–71
 linear independence 68–69
 linear transformations 69–72, 73–74
 Lipschitz continuous functions 17–19
 nullity 71
 rank 71
 Metzler matrices 84
 Milstein method 201, 213
 minimal subspaces 68–69
 minimum-maximum principle 158–159
MIS see management information systems
 mixed derivatives 331–333, 478–483
 mixed Greeks 282, 311
 mixed (Robin) condition 159, 160
 modified Euler method 54
 modified trapezoidal rule 31
 MOL see Method of Lines
 monotone upwind schemes, Asian options 455
 monotonicity result 22–24
 Monte Carlo applications, design 438–440, 443–444
 Monte Carlo options pricing 204–209
 moving boundary value problems 387–405
 American options 400–405
 definition 388
 front-fixing 399–400
 melting ice 389–390
 oxygen diffusion 390
 parabolic variational inequalities 392–399
 multi-asset options 425–445
 American spread options 440–442
 bivariate normal distribution 427–434
 boundary conditions 435–436, 442–443
 domain transformation 435
 Fichera theory 435–436
 finite difference schemes 436–440
 Goursat PDE 431–433
 integral computation 430–431
 model requirements 435–436
 software design 436–440
 two-factor model 139–140
- multi-dimensional Fokker–Planck equation 219–220
- multi-index notation 131–132
- multiplication
 matrices 79
 scalars 65–66, 79
- multiply connected 294
- multistep method 26
- natural boundary conditions 219
NDSolve 455–456
 near-field boundary 144

- Neumann (flux) condition 159, 160
nilpotent matrices 80–81
non-autonomous 35
non-homogeneous Dirichlet boundary conditions 158
non-linear initial value problem 48
non-linear solvers 492–493
non-negative matrices 83
non-uniform mesh 51, 199
normal linear spaces 17
normal matrices 81
norms 66, 232–233
notation, vector spaces 64–65
nullity 71
numerical approximation
 adjoint-form PDEs 134–135
 stochastic differential equations 199–203
numerical linear algebra 87–92
- odeint* library, C++ 55–62, 378–384
ODEs *see* ordinary differential equations
one-factor problems
 Black–Scholes equation 109–126
 boundary value problems 87–108
 finite dimensional vector spaces 63–72
 foundations 3–19
 matrix theory 73–86
 ordinary differential equations 21–62
one-phase problems
 definition 388
 melting ice 389–390
 oxygen diffusion 390
one-sided differences 25
one-sided Lipschitz condition 375
one-step method 26, 28
one-step trapezoidal method 34
operations, matrices 78–79
operator splitting 330–331, 413
optimal exercise boundary 391
options pricing
 Asian 447–459
 chooser options 433–434
 Monte Carlo method 204–209
 multi-asset 425–445
 see also American options; Asian options;
 Black–Scholes model; European call
 options; European put options
ordinary differential equations (ODEs) 21–62
 Bernoulli-type 45–46
 C++ coding 55–62
 consistency 33–34, 36
 discrete maximum principle 28–29
 discrete time approximations 32–37
 discretisation 25–39
 existence 43–45
 explicit Euler method 26, 51, 53, 57–59
 explicit solutions 39–40
 exponential fitting 29–31, 37–39
 generalisations 24–25
 inhomogeneous forcing term 22
 initial condition 22
 initial value problem 22, 24–29
 integration factor 22
 logistic functions 48
 matrix 24, 59–62
 numerical methods 51–55
 Picard iterative method 44–45
 positivity 22–24
 predator-prey models 47
 predictor-corrector model 31, 52, 54
 Python coding 52–55
 qualitative properties 22–24
 rationale 24
 Riccati-type 46, 55–59
 Richardson extrapolation 31–32, 57–59
 root condition 34–36
 stability 34, 36
 stiff 37–39
 transition rate matrices 61–62
 uniqueness 44–45
Ornstein–Uhlenbeck process 173
orthogonal matrices 82
orthogonal vectors 75
orthonormal basis 75
oscillatory behaviour 28, 29, 101, 255, 263
outflow boundaries 177
oxygen diffusion 390
- Padé matrix approximation 269–274
parabolic initial boundary value problems 114
parabolic partial derivative equations
 heat equation 182–190
 maximum-minimum principles 189–190
 time-dependent boundaries 190–192
parabolic variational inequalities 392–399
 initial boundary value problems 395–399
 problem formulation 392–394
parameter sets 48, 196
partial derivatives 11–13
partial differential equations (PDEs)
 adjoint equations 132–135
 Asian options 241, 447–459
 background 129–130
 bivariate normal distribution 430–433

- partial differential equations (PDEs) (*Continued*)
- boundary conditions 159
 - boundary value problems 159–160
 - bounded domains 143–151
 - canonical form 147–148
 - Cauchy–Euler equation 163–165
 - convection-diffusion reaction equation 129–130
 - Cox–Ingersoll–Ross model 462–471
 - degenerate 135
 - eigenvalues 131
 - elliptic operators 130–135
 - energy estimates 166–167
 - Fichera theory 168–178
 - first-order 177–178, 239–248
 - Goursat-type 216, 431–433
 - Heston model 176
 - hotspots 148
 - Laplace operator 130
 - log transformation 149–151
 - maximum-minimum principles 159–162, 189–190
 - multi-asset options 139–140
 - numerical approximation 134–135
 - principal symbols 131–132
 - second-order equations classification 135–138
 - self-adjoint operators 133–135
 - semi-elliptic 135
 - stochastic dividends 140–141
 - stochastics representations 195–222
 - time-dependent 181–193
 - total symbols 131–132
 - two-dimensional convection 341–343
 - well-posed 165–166
- partial integro-differential equations (PIDE) 186, 338–339
- particular integral 37
- partitions 293
- pasting conditions 390, 391
- path 48
- path evolution 203–209
- patterned matrices 89
- Pauli matrices 80
- Picard iterative method 44–45
- PIDE *see* partial integro-differential equations
- Poisson's equation 106–107
- population dynamics 47, 307–309
- positive definite matrices 82–83
- positivity 22–24
- postprocessing 438, 439
- Power method 38–39
- power series 366
- predator-prey models 47
- predictor-corrector methods 31, 52, 54
- drift-adjusted 202–203
 - Method of Lines 374–375
 - stochastic differential equations 201–203, 214
 - three-dimensional heat equation 337
- predictor-corrector splitting 415–416
- price jumps 186
- pricers 438, 439, 444
- principal symbols 131–132
- probability density functions 215–220
- process control systems (PCS) 437
- projective successive over-relaxation (PSOR) method 106, 394
- Put options
- alternating direction explicit method 121–122
 - Heston model 175
- Python
- American options 400–405
 - complex step method 300–301
 - Faddeeva function 484–487
 - ordinary differential equations 52–55
- qualitative properties, ordinary differential equations 22–24
- radius of convergence 366
- random processes 196
- random sequences 48
- rank 71
- realisation 48
- rectangular matrix 76, 78
- rectified linear unit (ReLU) 4–5
- recurrence relations 40
- reducible matrices 83
- reflecting barrier conditions 219, 453
- region of absolute stability 34, 38
- regions, complex function theory 293–294
- regularisation 185
- resource allocation and tracking (RAT) systems 437, 443–444
- reverse mode, automatic differentiation 316
- Reynold's number 113
- Riccati equation 46, 55–59
- Richardson extrapolation 31–32, 57–59
- Riemann–Stieltjes integral 293
- Roberts–Weiss method 357–358, 455
- Robin (mixed) condition 159, 160
- root condition 34–36
- Rosenbrock functions 490–491
- rotation of the axis 149–150
- row matrix 78
- row-major strategy 107

- sample functions 48
sample space 196
Saul'yev method 354–355
scalar fields 178
scalar first-order linear ordinary differential equations 22
scalar multiplication 65–66, 79
scalar product 74
scalars 65, 76–77
scale factors 148
Schrödinger equation 85–86
sealed surfaces 393
second-order Greeks 282, 310–311
second-order Ralston method 52, 54
sectionally smooth curve 293
self-adjoint operators 133–135
semi-discretisation 368–369, 371–373
 Asian options 452–453
 convection-diffusion reaction equations 264–269
semi-elliptic partial differential equations 135
semi-implicit methods 213, 374
semi-linear problems 373–375
semi-norm 232
semigroups 408–410
sensitivity analysis 277–319
 alternating direction explicit method 124
 American options 384
 automatic differentiation 314–319
 complex function theory 292–299
 complex step method 299–302, 310
 continuous sensitivity equation 305–313
 cubic spline interpolation 286–292
 divided differences 282–286
 double-barrier options 381–384
 dual numbers 317–318
 European call options 381–384
 Faddeeva function 483–487
 Greeks 279–282, 285–286
 operator calculus 313–314
 overview 278–279
separation of variables 156–158, 370
sequential first-order splitting 413–414
similarity measures 75
similarity reduction 451–452
simple closed curve 293
simply connected 294
single responsibility principle (SRP) 438
single-phase problems
 definition 388
 melting ice 389–390
 oxygen diffusion 390
singular perturbation problems 29, 114
skew-Hermitian matrices 81
slow transients 37
smooth pasting condition 390
smoothness conditions, heat equation 188–189
software design
 American spread options 440–442
 chooser options 433–434
 multi-asset options 436–440
 splitting 346, 418–419
 stochastic differential equations 436–440, 443–444
testing 440
unified 436–440
 see also C++; Python
Soviet (operator) splitting 330–331, 413
spark spread options 210
sparse data, cubic splines 289–290
sparse matrices 89
spatial stability
 alternating direction explicit method 361–362
 convection-diffusion reaction equations 251–257
spectral norms 262
speed 282, 311
splitting 323–421
 alternating direction explicit method 349–364
 alternating direction implicit method 328–330
 applications 417–418
 approximate factorisation 334–336
 Asian options 455–456
 background 324–325
 Banach spaces 408–410
 boundary conditions 339–340
 Cauchy problem 409–410
 compared to ADI 419–421
 components 413
 as discretization 413–414
 D'Yakonov method 333–334
 errors 411–413
 hopscotch method 344–345
 implementation 346
 interest rate models 471–475
 Lie product formula 411
 Lie-Trotter-type 415, 439
 Marchuk's two-cycle method 416, 439
 model problems 325–328
 notation 325
 operators 330–331, 413
 partial integro-differential equations 338–339
 popular methods 414–417
 predictor-corrector 415–416
semigroups 408–410

- splitting (*Continued*)
 software design 346, 418–419
 Strang-type 412, 417
 three-dimensional equations 344–345
 two-dimensional convection equations
 341–343
 Yanenko scheme 331–333
- spot price, Heston model 173
- spread call options, stochastic volatility
 209–211
- spread options
 boundary conditions 442–443
 C++ coding 434
- spurious oscillations 28, 29, 101, 255
- spurious reflections 453
- square matrix 76
- SRP *see* single responsibility principle
- stabilising corrections space 337
- stability 34, 36, 238–239
 Crank–Nicolson scheme 250–251
 finite difference method 251–257
- stable difference schemes 28–29
- standard logistic function 48
- state space 48
- static instability 252
- steady-state solutions 37
- Stefan conditions 390, 392
- Stefan problems 388
 see also moving boundary value problems
- stiff systems 29, 37–39
- stiffness matrix 398–399
- stochastic differential equations (SDEs) 18–19,
 195–222
 Black–Scholes equation 115–119
 C++ code 205–209
 Cox–Ingersoll–Ross (CIR) model 462–471
 drift-adjusted predictor-corrector methods
 202–203
 Euler–Maruyama method 199–200, 205–209
 existence theorems 48–51
 Feynman–Kac formula 217
 first exit-time problems 221–222
 Fokker–Planck equation 218–220
 implicit Milstein scheme 201, 214
 Ito formula 215
 Ito–Milstein scheme 201
 Kolmogorov backward equation
 220, 221
 Kolmogorov equations 218–220
 Milstein method 201, 213
 Monte Carlo options pricing 204–209
 numerical approximation 199–203
 one-dimensional processes 196–199
 one-factor 205–209
 overview 196
 parameter sets 48
 path evolution 203–209
 predictor-corrector methods 201–203, 214
 software design 436–440, 443–444
 Stratonovich–Milstein scheme 201
 two-factor problems 209–214
 stochastic splitting 413
 stochastic volatility
 Heston model 211–214
 spread options 209–211
 stopping time 221–222
 Strang splitting 412, 417
 Stratonovich–Milstein scheme 201
 stress testing, alternating direction explicit
 method 122
 strike delta 311
 strike gamma 282, 311
 strongly continuous one-parameter semi-groups
 408–409
 strongly dissipative 375
 Sturm–Liouville system 157
 subintervals 51
 subspaces 67–68, 70–71
 subtraction, vectors 66
 successive over-relaxation (SOR) 105, 106
 sum of two linear transformations 74
 symmetrically weighted sequential splitting
 (SWSS) 412
 symmetric matrices 81
 target set 220
 Taylor’s theorem 9–10, 294–295
 temporal stability, convection-diffusion reaction
 251–257
 terminal value problems 109
 theta 280, 311
 Theta method 27, 37–38, 326–328, 372
 third-order Greeks 282, 311
 Thomas algorithm 96–97
 Thomée schemes 27, 247, 471
 three-dimensional heat equation 332–339,
 343–344
 time, discrete approximations 32–37
 time-dependent convection-diffusion reaction
 equations 275–276
 time-dependent partial differential equations
 181–193
 initial boundary value problem
 182–184
 separation of variables 184
 well-posed problems 184–188

- time-marching schemes 326
Toeplitz matrices 94, 263
top-down design, Monte Carlo applications 443–444
total symbols 131–132
Towler–Yang approximation 356–357, 455
TPBVP *see* two-point boundary value problems
transient solutions 37
transition probability density function 218
transition rate matrices 61–62
transposal, matrices 79
trapezoidal method 28, 34
trapezoidal scheme 36
tridiagonal (Jacobi) matrices 89, 94–99
 block 89, 97–99
 C++ code 489–491
 complex step method 489–491
 cubic splines 288
 double sweep method 94–96
 oscillations 263
 Thomas algorithm 96–97
 trinomial method 115–119
Trotter product formula 411
truncation error 239
turning point problems 255
turning points 255
two-dimensional convection equations 341–343
two-factor Hull–White model 480–481, 493–495
two-factor problems
 adjoint operators 132–135
 alternating direction explicit method 349–364
 bounded domains 143–152
 Cauchy–Euler equation 163–165
 convection-diffusion reaction equation 129–130
 elliptic equations 130–135
 elliptic PDE boundary value problems 153–162
 energy estimates 166–167
 Fichera theory 163, 168–178
 free boundary value problems 387–405
 Heston model 172–178, 211–214
 Laplace equation 154–159
 method of lines 365–385
 moving boundary value problems 387–405
 multi-asset options 139–140
 second-order equations classification 135–138
 splitting methods 323–348, 407–421
 stochastic dividends 140–141
 stochastics representations 195–222
 time-dependent PDEs 181–194
two-phase problems, melting ice 392
two-point boundary value problems (TPBVP) 99–103
UDR *see* unified difference representation
unconditionally stable 234
unified difference representation (UDR) 255
unified software design (USD) 436–440, 443–444
uniform continuity 6–7, 18
uniform limit theorem 7
uniform mesh 51, 199
uniqueness
 ordinary differential equations 44–45
 two-dimensional boundary value problems 192–193
unitary matrices 82
unitary space 75
upper operator 335
upper triangular matrices 89
upwind schemes 177
upwinding 253, 453, 455
USD *see* unified software design
vanna 282, 311
variance, Heston model 173
vector fields 178–180
vector spaces
 concepts 65–67
 finite dimensional 63–72
 inner product spaces 74–75
 matrices 76–77
 notation 64–65
 orthonormal basis 75
 subspaces 67–68, 70–71
vector subspaces 67–68
vector-valued transformations 70
vector–vector operations, BLAS 90
vectors
 addition 65
 dot product 178
 magnitude 178
 norms 232–233
 orthogonal 75
 subtraction 66
vega 280–281, 311
vertical method of lines (VMOL) 365

- viscosity, Lax–Wendroff scheme PDEs 245
VMOL *see* vertical method of lines
volatility 46, 280–281, 311
volga 311
von Neumann stability analysis 235

wave equations 236
Weierstrass function 7
well-posed problems 23, 165–166, 184–188,
 466–469

Yanenko scheme 331–333

Z-matrices 84
zero coupon bonds 46
 Cox–Ingersoll–Ross model 242,
 463–466
 initial boundary value problems 466
zero element 65
zero matrix 78
zomma 311