Use R!

Solving Differential Equations in R

Springer

Karline Soetaert
Jeff Cash
Francesca Mazzia

Solving Differential Equations in R

Springer

Karline Soetaert
Department Ecosystem Studies
Royal Netherlands Institute for Sea Research
Yerseke
The Netherlands

Jeff Cash
Mathematics
Imperial College
South Kensington Campus
United Kingdom

Francesca Mazzia
Dipartimento di Matematica
University of Bari
Bari
Italy

Series Editors:
Robert Gentleman
Program in Computational Biology
Division of Public Health Sciences
Fred Hutchinson Cancer Research Center
1100 Fairview Avenue, N. M2-B876
Seattle, Washington 98109
USA

Giovanni Parmigiani
The Sidney Kimmel Comprehensive
Cancer Center at Johns Hopkins University
550 North Broadway
Baltimore, MD 21205-2011
USA

Kurt Hornik
Department of Statistik and Mathematik
Wirtschaftsuniversität Wien Augasse 2-6
A-1090 Wien
Austria

R-package diffEq to be downloaded from CRAN URL: http://cran.r-project.org/web/packages/diffEq
In addition R-code of all examples can be downloaded from Extras.Springer.com, also accessible via Springer.com/978-3-642-28069-6

ISBN 978-3-642-28069-6 ISBN 978-3-642-28070-2 (eBook)
DOI 10.1007/978-3-642-28070-2
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2012939126

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To Carlo, Roslyn and Antonello

Preface

Mathematics plays an important role in many scientific and engineering disciplines. This book deals with the numerical solution of differential equations, a very important branch of mathematics. Our aim is to give a practical and theoretical account of how to solve a large variety of differential equations, comprising ordinary differential equations, initial value problems and boundary value problems, differential algebraic equations, partial differential equations and delay differential equations.

The solution of differential equations using R is the main focus of this book. It is therefore intended for the practitioner, the student and the scientist, who wants to know how to use R to solve differential equations.

When writing his famous book, "A Brief History of Time", Stephen Hawking [2] was told by his publisher that every equation he included in the book would cut its sales in half. When writing the current book, we have been mindful of this, and our main desire is to provide the reader with powerful numerical algorithms written in the R programming language for the solution of differential equations rather than considering the theory in any great detail.

However, we also bear in mind the famous statement of Kurt Lewin which is "there is nothing so practical as a good theory". Therefore each chapter that deals with R examples is preceded by a chapter where the theory behind the numerical methods being used is introduced. It has been our goal that non-mathematicians should at least understand the basics of the methods, while obtaining entrance into the relevant literature that provides more mathematical background. We believe that some knowledge of the fundamentals of the underlying algorithms is essential to use the software in an intelligent way, so the principles underlying the various methods should, at least at a basic level, be explained. Moreover, as this book is in the first place about R the discussion of the numerical methods will be skewed to what is actually available in R.

In the sections that deal with the use of R for solving differential equations, we have taken examples from a variety of disciplines, including biology, chemistry, physics, pharmacokinetics. Many are well-known test examples, used frequently in the field of numerical analysis.

R as a Problem Solving Environment

The choice of using R [8] may be surprising to people regularly involved in solving numerical problems. Powerful numerical methods for the solution of differential equations are typically programmed in e.g. Fortran, C, Java, or Python. Whereas these solution methods are often made freely available, it is unfortunately the case that one needs considerable programming expertise to be able to use them. In contrast, easy-to-use software is often in rather expensive programs, such as MATLAB, Maple or Mathematica. In line with this, most books that give practical information about how to solve differential equations make use of these big three problem solving environments, or of one of the free-of-charge variants.

Although still not often used for solving differential equations, R is also very well suited as a Problem Solving Environment. Apart from the fact that it is open source software, there are obvious advantages in solving differential equations in a software that is strong in visualisation and statistics. Moreover, more and more students are becoming acquainted with the language as its use in universities is growing rapidly, both for teaching and for research. This creates a unique opportunity to introduce these students to the powerful scientific methods which make use of differential equations.

The potential for using R to solve differential equations was initiated by the release of the R package **odesolve** by Woody Setzer, a biologist holding a bachelor's degree in mathematics from EPA, US [10]. Years later, a communication in the R-journal by Thomas Petzoldt, a biologist from the university of Dresden, Germany [5] showed the potential of R for solving initial value problems of ordinary differential equations in the field of ecology. Recently a number of books have applied R in the field of environmental modelling [12, 19]. Building upon this initial effort, Karline Soetaert, the first author of this book, (a biologist) in 2008 joined forces with Woody Setzer and Thomas Petzoldt to make an improved version of odesolve that was able to solve a much greater variety of differential equations. This resulted in the R package **deSolve** [17], which contains most of the integration methods available in R. Most of the solvers implemented in the R package **deSolve** are based on well-established numerical codes, programmed in Fortran. By using well tested, robust, reliable and powerful codes, more emphasis can be put on making the existing codes more versatile. For instance, most codes can now be used to solve delay differential equations, or to simulate events. Also, great care was taken to make a common interface that is (relatively) easy to apply from the user's point of view. A set of methods to solve partial differential equations by the method-of-lines was added to **deSolve**, while another package, **rootSolve** [11], was devised to efficiently solve partial differential equations and boundary value problems using root solving algorithms. Finally, solution methods for boundary value problems were implemented in R package **bvpSolve** [15], as a cooperation between the three authors from this book.

Because all these R packages share one common author (KS), there is a certain degree of consistency in them, which we hope to demonstrate here (see also [16]).

Quite a few other R packages deal with the implementation of differential equations [6, 13], with the solution of special types of differential equations [1, 3, 4, 7], with statistical analysis of their outputs [9,14,20], or provide test problems on which the various solvers can be benchmarked [18].

About the Three Authors

Mathematics is the playground not only for the mathematician and engineer who devise powerful mathematical techniques to solve particular classes of problems, but also for the scientist who applies these methods to real-world problems. Both disciplines meet at the level of software, the actual implementation of these methods in computer code.

The three authors reflect this duality and come from different disciplines. Jeff Cash and Francesca Mazzia are experts in numerical analysis in general and the construction of algorithms for solving differential equations in particular. In contrast Karline Soetaert is a biologist, with an additional degree in computer science, whose interest in these numerical methods is mainly due to the fact that she uses these algorithms for application in the field of the marine sciences. Although she originally wrote her scientific programs mainly in Fortran, since she came acquainted with R in 2007 she now performs nearly all of her scientific work in this programming environment.

Acknowledgment Many people have commented on the first versions of this book. We are very thankful for the reviews provided by Filip Meysman, Dick van Oevelen, Tom Cox, Tom van Engeland, Ernst Hairer, Bill Schiesser, Alexander Ostermann, Willem Hundsdorfer, Vincenzo Casulli, Linda Petzold, Felice Iavernaro, Luigi Brugnano, Raymond Spiteri, Luis Randez, Alfredo Bellen, Nicola Guglielmi, Bob Russell, René Lamour, Annamaria Mazzia, and Abdelhameed Nagy.

References

1. Couture-Beil, A., Schnute, J. T., & Haigh, R. (2010). **PBSddesolve***: Solver for delay differential equations*. R package version 1.08.11.
2. Hawking, S. (1988). *A brief history of time*. Toronto/New York: Bantam Books. ISBN 0-553-38016-8.
3. Iacus, S. M. (2009). **sde***: Simulation and inference for stochastic differential equations*. R package version 2.0.10.
4. King, A. A., Ionides, E. L., & Breto, C. M. (2012). **pomp***: Statistical inference for partially observed Markov processes*. R package version 0.41-3.
5. Petzoldt, T. (2003). R as a simulation platform in ecological modelling. *R News, 3*(3), 8–16.
6. Petzoldt, T., & Rinke, K. (2007). **simecol**: An object-oriented framework for ecological modeling in R. *Journal of Statistical Software, 22*(9), 1–31.
7. Pineda-Krch, M. (2010). **GillespieSSA***: Gillespie's stochastic simulation algorithm (SSA)*. R package version 0.5-4.

8. R Development Core Team, (2011). *R: A language and environment for statistical computing*. Vienna: R Foundation for Statistical Computing. ISBN 3-900051-07-0.
9. Radivoyevitch, T. (2008). Equilibrium model selection: dTTP induced R1 dimerization. *BMC Systems Biology, 2*, 15.
10. Setzer, R. W. (2001). *The* **odesolve** *package: Solvers for ordinary differential equations*. R package version 0.1-1.
11. Soetaert, K. (2011). **rootSolve***: Nonlinear root finding, equilibrium and steady-state analysis of ordinary differential equations*. R package version 1.6.2.
12. Soetaert, K., & Herman, P. M. J. (2009). *A practical guide to ecological modelling. Using R as a simulation platform*. Dordrecht: Springer. ISBN 978-1-4020-8623-6.
13. Soetaert, K., & Meysman, F. (2012). Reactive transport in aquatic ecosystems: Rapid model prototyping in the open source software R. *Environmental Modelling and Software, 32*, 49–60.
14. Soetaert, K., & Petzoldt, T. (2010). Inverse modelling, sensitivity and monte carlo analysis in R using package **FME**. *Journal of Statistical Software, 33*(3):1–28.
15. Soetaert, K., Cash, J. R., & Mazzia, F. (2011). **bvpSolve***: Solvers for boundary value problems of ordinary differential equations*. R package version 1.2.2.
16. Soetaert, K., Petzoldt, T., & Setzer, R. W. (2010) Solving differential equations in R. *The R Journal, 2*(2):5–15.
17. Soetaert, K., Petzoldt, T., & Setzer, R. W. (2010). Solving differential equations in R: Package **deSolve**. *Journal of Statistical Software, 33*(9):1–25.
18. Soetaert, K., Cash, J. R., & Mazzia, F. (2011). **deTestSet***: Testset for differential equations*. R package version 1.0.
19. Stevens, M. H. H. (2009). *A primer of ecology with R*. Berlin: Springer.
20. Tornoe, C. W., Agerso, H., Jonsson, E. N., Madsen, H., & Nielsen, H. A. (2004). Non-linear mixed-effects pharmacokinetic/pharmacodynamic modelling in **NLME** using differential equations. *Computer Methods and Programs in Biomedicine, 76*, 31–40.

Contents

1 Differential Equations 1
1.1 Basic Theory of Ordinary Differential Equations 1
1.1.1 First Order Differential Equations 1
1.1.2 Analytic and Numerical Solutions 2
1.1.3 Higher Order Ordinary Differential Equations 3
1.1.4 Initial and Boundary Values 4
1.1.5 Existence and Uniqueness of Analytic Solutions 5
1.2 Numerical Methods 6
1.2.1 The Euler Method 6
1.2.2 Implicit Methods 7
1.2.3 Accuracy and Convergence of Numerical Methods 8
1.2.4 Stability and Conditioning 9
1.3 Other Types of Differential Equations 11
1.3.1 Partial Differential Equations 11
1.3.2 Differential Algebraic Equations 12
1.3.3 Delay Differential Equations 13
References 13

2 Initial Value Problems 15
2.1 Runge-Kutta Methods 15
2.1.1 Explicit Runge-Kutta Formulae 15
2.1.2 Deriving a Runge-Kutta Formula 17
2.1.3 Implicit Runge-Kutta Formulae 22
2.2 Linear Multistep methods 22
2.2.1 Convergence, Stability and Consistency 23
2.2.2 Adams Methods 25
2.2.3 Backward Differentiation Formulae 27
2.2.4 Variable Order – Variable Coefficient Formulae for Linear Multistep Methods 29
2.3 Boundary Value Methods 30
2.4 Modified Extended Backward Differentiation Formulae 31

2.5 Stiff Problems 32
2.5.1 Stiffness Detection 33
2.5.2 Non-stiffness Test 34
2.6 Implementing Implicit Methods 34
2.6.1 Fixed-Point Iteration to Convergence 34
2.6.2 Chord Iteration 35
2.6.3 Predictor-Corrector Methods 36
2.6.4 Newton Iteration for Implicit Runge-Kutta Methods 36
2.7 Codes to Solve Initial Value Problems 37
2.7.1 Codes to Solve Non-stiff Problems 38
2.7.2 Codes to Solve Stiff Problems 38
2.7.3 Codes that Switch Between Stiff and Non-stiff Solvers 38
References 39

3 Solving Ordinary Differential Equations in R 41
3.1 Implementing Initial Value Problems in R 41
3.1.1 A Differential Equation Comprising One Variable 42
3.1.2 Multiple Variables: The Lorenz Model 44
3.2 Runge-Kutta Methods 45
3.2.1 Rigid Body Equations 47
3.2.2 Arenstorf Orbits 49
3.3 Linear Multistep Methods 51
3.3.1 Seven Moving Stars 52
3.3.2 A Stiff Chemical Example 56
3.4 Discontinuous Equations, Events 59
3.4.1 Pharmacokinetic Models 60
3.4.2 A Bouncing Ball 64
3.4.3 Temperature in a Climate-Controlled Room 66
3.5 Method Selection 68
3.5.1 The van der Pol Equation 70
3.6 Exercises 75
3.6.1 Getting Started with IVP 75
3.6.2 The Robertson Problem 76
3.6.3 Displaying Results in a Phase-Plane Graph 76
3.6.4 Events and Roots 78
3.6.5 Stiff Problems 79
References 79

4 Differential Algebraic Equations 81
4.1 Introduction 81
4.1.1 The Index of a DAE 82
4.1.2 A Simple Example 83
4.1.3 DAEs in Hessenberg Form 84
4.1.4 Hidden Constraints and the Initial Conditions 85
4.1.5 The Pendulum Problem 86

4.2 Solving DAEs 87
4.2.1 Semi-implicit DAEs of Index 1 87
4.2.2 General Implicit DAEs of Index 1 88
4.2.3 Discretization Algorithms 89
4.2.4 DAEs of Higher Index 90
4.2.5 Index of a DAE Variable 93
References 94

5 Solving Differential Algebraic Equations in R 95
5.1 Differential Algebraic Equation Solvers in R 95
5.2 A Simple DAE of Index 2 96
5.2.1 Solving the DAEs in General Implicit Form 97
5.2.2 Solving the DAEs in Linearly Implicit Form 98
5.3 A Nonlinear Implicit ODE 98
5.4 A DAE of Index 3: The Pendulum Problem 100
5.5 Multibody Systems 101
5.5.1 The Car Axis Problem 102
5.6 Electrical Circuit Models 106
5.6.1 The Transistor Amplifier 107
5.7 Exercises 111
5.7.1 A Simple DAE 111
5.7.2 The Robertson Problem 111
5.7.3 The Pendulum Problem Revisited 111
5.7.4 The Akzo Nobel Problem 112
References 115

6 Delay Differential Equations 117
6.1 Delay Differential Equations 117
6.1.1 DDEs with Delays of the Dependent Variables 118
6.1.2 DDEs with Delays of the Derivatives 118
6.2 Difficulties when Solving DDEs 119
6.2.1 Discontinuities in DDEs 119
6.2.2 Small and Vanishing Delays 120
6.3 Numerical Methods for Solving DDEs 121
References 121

7 Solving Delay Differential Equations in R 123
7.1 Delay Differential Equation Solvers in R 123
7.2 Two Simple Examples 124
7.2.1 DDEs Involving Solution Delay Terms 124
7.2.2 DDEs Involving Derivative Delay Terms 124
7.3 Chaotic Production of White Blood Cells 125
7.4 A DDE Involving a Root Function 127
7.5 Vanishing Time Delays 128
7.6 Predator-Prey Dynamics with Harvesting 130

7.7 Exercises 132
7.7.1 The Lemming Model 132
7.7.2 Oberle and Pesch 132
7.7.3 An Epidemiological Model 133
7.7.4 A Neutral DDE 134
7.7.5 Delayed Cellular Neural Networks With Impulses 134
References 135

8 Partial Differential Equations 137
8.1 Partial Differential Equations 137
8.1.1 Alternative Formulations 138
8.1.2 Polar, Cylindrical and Spherical Coordinates 140
8.1.3 Boundary Conditions 141
8.2 Solving PDEs 142
8.3 Discretising Derivatives 143
8.3.1 Basic Diffusion Schemes 144
8.3.2 Basic Advection Schemes 145
8.3.3 Flux-Conservative Discretisations 147
8.3.4 More Complex Advection Schemes 148
8.4 The Method Of Lines 152
8.5 The Finite Difference Method 153
References 153

9 Solving Partial Differential Equations in R 157
9.1 Methods for Solving PDEs in R 157
9.1.1 Numerical Approximations 157
9.1.2 Solution Methods 159
9.2 Solving Parabolic, Elliptic and Hyperbolic PDEs in R 160
9.2.1 The Heat Equation 160
9.2.2 The Wave Equation 163
9.2.3 Poisson and Laplace's Equation 166
9.2.4 The Advection Equation 168
9.3 More Complex Examples 170
9.3.1 The Brusselator in One Dimension 170
9.3.2 The Brusselator in Two Dimensions 173
9.3.3 Laplace Equation in Polar Coordinates 174
9.3.4 The Time-Dependent 2-D Sine-Gordon Equation 176
9.3.5 The Nonlinear Schrödinger Equation 179
9.4 Exercises 181
9.4.1 The Gray-Scott Equation 181
9.4.2 A Macroscopic Model of Traffic 182
9.4.3 A Vibrating String 183
9.4.4 A Pebble in a Bucket of Water 184
9.4.5 Combustion in 2-D 184
References 185

10 Boundary Value Problems 187
10.1 Two-Point Boundary Value Problems 187
10.2 Characteristics of Boundary Value Problems 188
10.2.1 Uniqueness of Solutions 188
10.2.2 Isolation of Solutions 189
10.2.3 Stiffness of Boundary Value Problems and Dichotomy 189
10.2.4 Conditioning of Boundary Value Problems 190
10.2.5 Singular Problems 191
10.3 Boundary Conditions 192
10.3.1 Separated Boundary Conditions 192
10.3.2 Defining Good Boundary Conditions 193
10.3.3 Problems Defined on an Infinite Interval 193
10.4 Methods of Solution 194
10.5 Shooting Methods for Two-Point BVPs 194
10.5.1 The Linear Case 194
10.5.2 The Nonlinear Case 195
10.5.3 Multiple Shooting 196
10.6 Finite Difference Methods 197
10.6.1 A Low Order Method for Second Order Equations 197
10.6.2 Other Low Order Methods 198
10.6.3 Higher Order Methods Based on Collocation Runge-Kutta Schemes 199
10.6.4 Higher Order Methods Based on Mono Implicit Runge-Kutta Formulae 200
10.6.5 Higher Order Methods Based on Linear Multistep Formulae 201
10.6.6 Deferred Correction 202
10.7 Codes for the Numerical Solution of Boundary Value Problems 203
References 203

11 Solving Boundary Value Problems in R 207
11.1 Boundary Value Problem Solvers in R 207
11.2 A Simple BVP Example 208
11.2.1 Implementing the BVP in First Order Form 208
11.2.2 Implementing the BVP in Second Order Form 209
11.3 A More Complex BVP Example 210
11.4 More Complex Initial or End Conditions 214
11.5 Solving a Boundary Value Problem Using Continuation 216
11.5.1 Manual Continuation 216
11.5.2 Automatic Continuation 219
11.6 BVPs with Unknown Constants 220
11.6.1 The Elastica Problem 221
11.6.2 Non-separated Boundary Conditions 222
11.6.3 An Unknown Integration Interval 225

11.7 Integral Constraints 228
11.8 Sturm-Liouville Problems 229
11.9 A Reaction Transport Problem 230
11.10 Exercises 233
11.10.1 A Stiff Boundary Value Problem 233
11.10.2 The Mathieu Equation 234
11.10.3 Another Swirling Flow Problem 234
11.10.4 Another Reaction Transport Problem 236
References 237

A Appendix 239
A.1 Butcher Tableaux for Some Runge-Kutta Methods 239
A.2 Coefficients for Some Linear Multistep Formulae 239
A.3 Implemented Integration Methods for Solving Initial Value Problems in R 241
A.4 Other Integration Methods in R 242
References 242

Index 245

Chapter 1
Differential Equations

Abstract Differential equations (DEs) occur in many branches of science and technology, and there is a real need to solve them both accurately and efficiently. There are relatively few problems for which an analytic solution can be found, so if we want to solve a large class of problems, then we need to resort to numerical calculations. In this chapter we will give a very brief survey of the theory behind DEs and their solution. We introduce concepts such as analytic and numerical methods, the order of differential equations, existence and uniqueness of solutions, implicit and explicit methods. We end with a brief survey of the different types of differential equations that will be dealt with in later chapters of this book.

1.1 Basic Theory of Ordinary Differential Equations

Although the material contained in this section is largely of a theoretical nature it is presented at a rather basic level and the reader is advised to at least skim through it.

1.1.1 First Order Differential Equations

The general form taken by a first order ordinary differential equation (*ODE*) is

$$y' = f(x,y), \tag{1.1}$$

which may also be written as

$$\frac{dy}{dx} = f(x,y), \tag{1.2}$$

where f is a given function of x and y and y contained in $\Re^m$ is a vector. Here x is called the independent variable and $y = y(x)$ is the dependent variable.

K. Soetaert et al., *Solving Differential Equations in R*, Use R!,
DOI 10.1007/978-3-642-28070-2_1, © Springer-Verlag Berlin Heidelberg 2012

This equation is called *first order* as it contains no higher derivatives than the first. Furthermore, (1.1) is called an *ordinary* differential equation as y depends on one independent variable only.

1.1.2 Analytic and Numerical Solutions

A differentiable function $y(x)$ is a solution of (1.1) if for all x

$$y'(x) = f(x, y(x)). \tag{1.3}$$

If we suppose that $y(x_0)$ is known, the solution of (1.3), valid in the interval $[x_0, x_1]$, is obtained by integrating both sides of (1.1) with respect to x, to give:

$$y(x) - y(x_0) = \int_{x_0}^{x} f(t, y(t))dt, \quad x \in [x_0, x_1]. \tag{1.4}$$

In some cases this integral can be evaluated exactly to give an equation for y, and this is called an *analytic* solution. For example, the equation

$$y' = y^2 + 1, \tag{1.5}$$

has as analytic solution

$$y = \tan(x + c). \tag{1.6}$$

Note the free parameter c that occurs in the solution. It has been known for a long time that the solution of a first order equation contains a free parameter and that this solution is uniquely defined if for example we impose an initial condition of the form $y(x_0) = y_0$ and we suppose that the function f satisfies some regularity conditions. This is important and we will return to it later.

Unfortunately, it is true to say that many ordinary differential equations which appear to be quite harmless, in the sense that we could expect them to be easy to solve, cannot be solved analytically, i.e. the solution can not be expressed in terms of known functions. An illuminating example of this is given in [4, p. 4] where it is shown how "small changes" in the problem (1.5) may make it much harder (or impossible) to solve analytically. Indeed, if equation (1.5) is changed "slightly" to

$$y' = y^2 + x, \tag{1.7}$$

then the solution has a very complex structure in terms of Airy functions [4]. In view of this, and the fact that most "real-life" applications consist of complicated systems of equations, it is often necessary to approximate the solution by solving equation (1.1) *numerically* rather than analytically.

Undergraduate mathematics courses often give the impression that most differential equations can be solved analytically, with numerical techniques being developed to deal with those few classes of equations that have no analytic solution. In fact, the opposite is true: while an analytic solution is extremely useful if it does exist, experience shows that most equations of practical interest need to be solved numerically.

1.1.3 Higher Order Ordinary Differential Equations

In the previous section, we considered only the first order differential equation (1.1). Ordinary differential equations can include higher order derivatives as well. For example, second order equations of the form:

$$y'' = f(x, y, y'), \tag{1.8}$$

arise in many practical applications.

Normally, in order to deal with the second order equation (1.8), we first convert it to a system of first order equations. This we do by defining an extra dependent variable, which equals the first order derivative of y, in the following way:

$$\begin{aligned} y' &= y_1 \\ y_1' &= f(x, y, y_1). \end{aligned} \tag{1.9}$$

Rather than having one differential equation, we now have a system of two differential equations. Defining $Y = (y, y_1)^T$, (1.9) is of the form (1.1), with $Y \in \Re^2$. As we will see later (Sect. 1.1.4) we need to specify two conditions to define the solution uniquely in this second order case.

As a simple example consider a small stone falling through the air from a tower. Gravity produces an acceleration of $g = 9.8$ ms^{-2}, while the air exerts a resistive force which is proportional to the velocity (v). The differential equation describing this is:

$$v' = g - kv. \tag{1.10}$$

If we are interested in the distance from the top of the tower (x), we use the fact that the velocity $v = x'$, and the equation becomes a second order differential equation:

$$x'' = g - kx'. \tag{1.11}$$

Now, in order to solve (1.11), we rewrite it as two first order equations.

$$\begin{aligned} x' &= v \\ v' &= g - kv. \end{aligned} \tag{1.12}$$

This technique carries over to higher order equations as well. If we are faced with the numerical solution of an nth order equation, it is often advisable to first reduce it to a system of n first order equations using the obvious extension of the technique described in (1.9) above. Consider for example the "swirling flow III problem" [1, p. 23], which comprises a second order and a fourth order equation describing the flow between two rotating, coaxial disks. The original problem definition

$$\begin{aligned} g'' &= (gf' - fg')/\varepsilon \\ f'''' &= (-ff''' - gg')/\varepsilon, \end{aligned} \tag{1.13}$$

needs one intermediate variable to represent the higher order derivative of g, and three to represent the higher order derivatives of f. The corresponding set of first order ODEs is:

$$\begin{aligned} g' &= g_1 \\ g_1' &= (gf_1 - fg_1)/\varepsilon \\ f' &= f_1 \\ f_1' &= f_2 \\ f_2' &= f_3 \\ f_3' &= (-ff_3 - gg_1)/\varepsilon. \end{aligned} \tag{1.14}$$

We will solve this problem in Sect. 11.3.

An exception to the rule is the special case where the first derivative y' is absent from (1.8). In such circumstances it is often better to derive special methods for the solution of

$$y'' = f(x,y), \tag{1.15}$$

rather than to introduce the term y' into the definition of the differential equation [3, p. 261].

1.1.4 Initial and Boundary Values

We saw in Sect. 1.1 that the integration of the ODE (1.5) introduced an arbitrary constant c into the solution. As long as the ODE is specified only by (1.1), then any value of c will give a valid solution. To select a unique solution, one extra condition is needed and this determines the value of c.

Depending on *where* in the integration interval the extra condition is specified we obtain an *initial* or *boundary* value problem. For example, when extending equation (1.5) by introducing the extra condition

$$y(0) = 1, \tag{1.16}$$

then using the general solution (1.6) we obtain $y(0) = \tan(0 + c) = 1$ or $c = \arctan(1) = \pi/4$. Therefore,

$$\begin{aligned} y' &= y^2 + 1 \\ y(0) &= 1, \end{aligned} \tag{1.17}$$

has the unique solution $y = \tan(x + \pi/4)$ providing that we restrict the domain of x suitably. As the extra condition is specified at the initial point of the integration interval, (1.17) is an *initial value problem* (IVP).

The general representation of a first order IVP is:

$$\begin{aligned} y' &= f(x,y) \\ y(x_0) &= y_0, \end{aligned} \tag{1.18}$$

where y can be a vector.

In the case of second order equations such as (1.9) it is necessary to prescribe *two* conditions to define y uniquely. In an initial value problem, both these conditions are prescribed at the initial point (x_0). For example we might have:

$$\begin{aligned} y'' &= f(x,y,y') \\ y(x_0) &= y_0 \\ y'(x_0) &= y'_0, \end{aligned} \tag{1.19}$$

or, in first order form,

$$\begin{aligned} y' &= y_1 \\ y'_1 &= f(x,y,y_1) \\ y(x_0) &= y_0 \\ y_1(x_0) &= y'_0. \end{aligned} \tag{1.20}$$

If instead we prescribe the solution at two different points x_0, x_f in the range of integration, we have a *boundary value problem* (BVP). There are several ways in which to specify these boundary conditions, e.g. :

$$\begin{aligned} y'' &= f(x,y,y') \\ y(x_0) &= y_0 \\ y(x_f) &= y_f. \end{aligned} \tag{1.21}$$

1.1.5 Existence and Uniqueness of Analytic Solutions

An extremely important question concerns the *existence and uniqueness* of solutions of (1.1). This theory is now quite standard and is given for example in [3, Sect. 1.7]. Following the approach of [1] we determine what is required for the IVP solution to exist, be unique and depend continuously on the data, i.e. be well-posed and then ask that the numerical method has similar behaviour.

Basically the main property that we need to ask for if a problem is to be well-posed is that the function $f(x,y)$, appearing in (1.1) should be continuous in a certain region and be Lipschitz continuous [1] with respect to y in that region. An important sufficient condition for Lipschitz continuity is that $f(x,y)$ has bounded partial derivatives df_i/dy_j. A nice summary of this theory is found in [1].

As a simple example of an IVP which does not satisfy the conditions for uniqueness consider

$$y' = -\sqrt{1-y^2}, \qquad y(0) = 1. \tag{1.22}$$

This has at least two solutions: $y = 1$, and $y = \cos(x)$. The uniqueness problem occurs because df/dy is unbounded at $x = 0$. However we can also use our intuition to foresee that there may be difficulties with this problem since if we perturb the initial condition to $y(0) = 1+\varepsilon$ for any positive ε, the solution becomes complex!

The analytic solution of a given second order boundary value problem is rarely possible to obtain (see [3, Sect. 1.3]). Furthermore a proof of the existence and uniqueness of a solution of a given two point boundary value problem is often much harder than for the initial value case and, indeed, boundary value problems are generally much more difficult to solve than initial value problems. We will consider the solution of boundary value problems in detail in Chap. 10.

1.2 Numerical Methods

Having briefly outlined some of the basic theory behind the (analytic) solution of ODEs, we now go on to consider some elementary numerical methods. Basically, in their simplest form, numerical methods start by subdividing the domain of the independent variable x into a number of discrete points, $x_0, x_1 = x_0 + h, \ldots$, and they calculate the approximate values of the dependent variable y and the derivatives of y with respect to x only at these points. These methods are called *finite difference* methods.

Thus, given a series of integration steps $x_0, x_1, \ldots, x_n$, a numerical method constructs a sequence of values $y_0, y_1, \ldots, y_n$, such that

$$y_n \approx y(x_n), \qquad n \geq 0. \tag{1.23}$$

We note the important notation used here namely that $x_n = x_0 + nh$ is a point where the approximate solution will be computed, $y(x_n)$ is the analytic solution at x_n and y_n is the numerical solution obtained at x_n.

1.2.1 The Euler Method

One of the oldest and most simple numerical methods for solving the initial value problem

$$\begin{aligned} y' \quad &= f(x,y) \\ y(x_0) &= y_0, \end{aligned} \tag{1.24}$$

is due to Euler. This method can be derived in several ways and we start by using a Taylor series approach. Supposing that $f(x,y)$ is analytic in the neighborhood of the

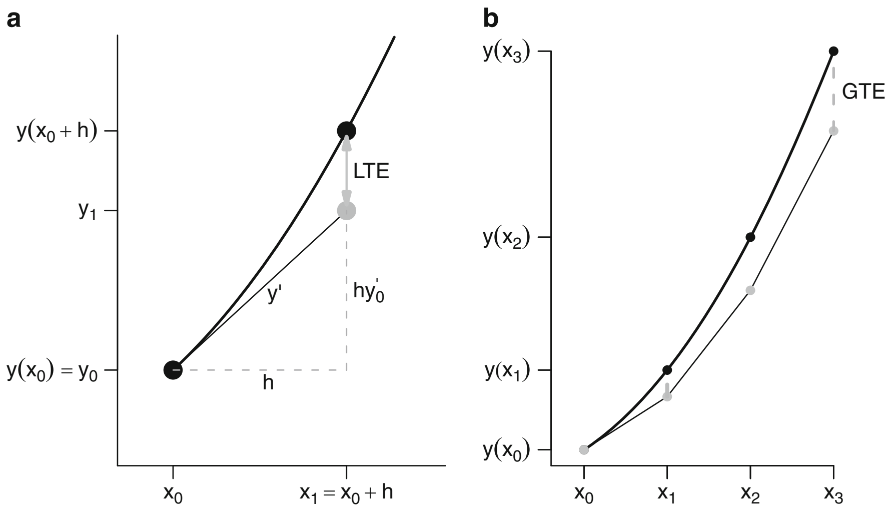

Fig. 1.1 Errors for the Euler method. (**a**) The local truncation error (*LTE*) is the error introduced by taking one Euler step. (**b**) After taking three integration steps, the global truncation error (*GTE*) is, for sufficiently small h, larger than the LTE. This is because for Euler's method the local truncation error is $O(h^2)$ while the global error is $O(h)$

initial value x_0, y_0 so that we can write

$$y(x_0 + h) = y(x_0) + hy'(x_0) + \sum_{r=2}^{\infty} \frac{h^r}{r!} y^{(r)}(x_0), \tag{1.25}$$

where $y^{(r)}$ is shorthand for the rth derivative of y with respect to x. Putting $x_1 = x_0 + h$, ignoring the infinite sum on the right-hand side of (1.25), assuming y_0 is exact and denoting the numerical solution obtained at $x_0 + h$ by y_1, we obtain the (forward) Euler method:

$$y_1 = y_0 + hf(x_0, y_0). \tag{1.26}$$

An alternative way of deriving Euler's method is via a geometric approach. If we evaluate the derivative of y at x_0 and assume that it is constant throughout $[x_0, x_0 + h]$, we have Fig. 1.1a, which immediately gives $y_1 = y_0 + hy'_0$. Of course $y'_0 = f(x_0, y_0)$ so we have again derived Euler's method.

1.2.2 *Implicit Methods*

The Euler formula (1.26) of the previous section expresses the new value y_1 as a function of the known y_0 and the function value $f(x_0, y_0)$. Thus y_1 can be calculated using only known values. Such formulae are called *explicit* methods. When using these methods it is simple to advance to the next integration step.

It is also possible to have *implicit* methods. An example of such a method is the so-called backward Euler method[1]:

$$y_1 = y_0 + hf(x_1, y_1), \tag{1.27}$$

where now the function value f depends on the unknown y_1, rather than solely on y_0. To solve for y_1 is in general not simple: naively we might think that we can just calculate the right-hand side, and estimate y_1, but we can do this only if y_1 is already known! Usually we will need an iterative method to solve for the unknowns. This means extra work per integration step and finding a solution may not always be easy or even possible (see Sect. 2.6).

1.2.3 Accuracy and Convergence of Numerical Methods

An important question concerning equation (1.26) is: how *accurate* is it *locally*? To answer this question, we rewrite (1.25) in the form:

$$y(x_1) - y(x_0) - hf(x_0, y(x_0)) = LTE, \tag{1.28}$$

where LTE is the *local truncation error* introduced by truncating the right-hand side of (1.25) and is given for Euler's method by:

$$LTE = \sum_{r=2}^{\infty} \frac{h^r}{r!} y^{(r)}(x_0). \tag{1.29}$$

Since we do not know the analytic solution $y(x)$ of (1.24), we cannot calculate the local truncation error exactly. The important thing about (1.29) is the power of h in the leading term in the expression for the LTE. For Euler's method this power is 2 and so the LTE is $O(h^2)$ and the method is said to have *accuracy* of order 1. In general, if a method has LTE proportional to h^{p+1}, $p \geq 1$, i.e. $|LTE| \leq Ch^{p+1}$ for sufficiently smooth problems and for h sufficiently small then the method is said to be of order p. The quantity we are interested in is in general not the LTE, but the *global* error $y_n - y(x_n)$ and for Euler's method this is $O(h)$. Hence Euler's method is said to be *convergent* of order 1 (see Fig. 1.1) and the global error for Euler's method is proportional to the constant step size h.

A very similar analysis can be carried out for implicit equations such as (1.27) and it is easy to show that (1.27) is also of order 1.

One strategy to reduce the local truncation error is to reduce the size of the steplength of integration h. In general, the higher the order of accuracy of the

[1]You may wonder why a formula that uses information "forward" in time is called "backward". This will become clear in Sect. 2.2.3.

numerical method, the more effect such step reduction will have on the LTE. On the other hand, higher order methods require more work for one integration step.

The art in devising a good numerical integration method is to achieve a prescribed accuracy with as little work as possible and this usually means with as few function evaluations as possible. Often this involves changing the step size as we perform the integration. If we use Euler's method the global error is proportional to the maximum step size used.

1.2.4 Stability and Conditioning

We complete this introductory chapter with a brief discussion concerning the concepts of *stability* and *conditioning*. The concept of stability is usually applied to initial value problems for differential equations, that of conditioning to boundary value problems. Both concepts relate to the effect small changes in (1.18), either in the function f, or in the initial (or boundary) conditions $y(x_0)$, have on the solution. If small changes induce large effects, the problem is said to be unstable or ill-conditioned. Conversely, a problem which has the desirable property that "small changes in the data produce small changes in the solution" is said to be stable or well-conditioned.

We can put the concept of stability on a firm theoretical basis as follows. Consider the initial value problem (1.18). A solution $y(x)$ is said to be stable with respect to the initial conditions $y(x_0)$ if, given any $\varepsilon > 0$, there is a $\delta > 0$ such that any other solution $\hat{y}(x)$ of (1.18) satisfying

$$|y(x_0) - \hat{y}(x_0)| \leq \delta, \tag{1.30}$$

also satisfies

$$|y(x) - \hat{y}(x)| \leq \varepsilon \qquad \text{for all } x > x_0. \tag{1.31}$$

This definition is usually called Lyapunov stability.

1.2.4.1 Absolute Stability

When we use a numerical method for the solution of a stable initial value problem it is important to require that the numerical solution has the same behavior as the continuous one. What is done in practice to investigate this is to consider a simple test equation

$$y' = \lambda y, \tag{1.32}$$

which is often called Dahlquist's test equation. If we consider λ to be complex with $Re(\lambda) < 0$ we know the true solution of this equation tends to zero as $x \to \infty$. We wish the solution of the numerical method to also behave in this way and if it does we say that $h\lambda$ lies in the stability region of the method. A convenient way to