NUMERICAL SOLUTION OF HYPERBOLIC PARTIAL DIFFERENTIAL EQUATIONS

This is a new type of graduate textbook, with both print and interactive electronic components (on CD). It is a comprehensive presentation of modern shock-capturing methods, including both finite volume and finite element methods, covering the theory of hyperbolic conservation laws and the theory of the numerical methods.

Classical techniques for judging the qualitative performance of the schemes, such as modified equation analysis and Fourier analysis, are used to motivate the development of classical higher-order methods (the Lax–Wendroff process) and to prove results such as the Lax Equivalence Theorem.

The range of applications (shallow water, compressible gas dynamics, magnetohydrodynamics, finite deformation in solids, plasticity, polymer flooding and water/gas injection in oil recovery) is broad enough to engage most engineering disciplines and many areas of applied mathematics.

The solution of the Riemann problems for these applications is developed, so that the reader can use the theory to develop test problems for the methods, especially to measure errors for comparisions of accuracy and efficiency. The numerical methods involve a variety of important approaches, such as MUSCL and PPM, TVD, wave propagation, Lax–Friedrichs (aka central schemes), ENO and discontinuous Galerkin; all of these are discussed in one and multiple spatial dimensions. Since many of these methods depend on Riemann solvers, there is extensive discussion of the basic design principles of approximate Riemann solvers, and several computationally useful techniques. The final chapter contains a discussion of adaptive mesh refinement via structured grids.

The accompanying CD contains a hyperlinked version of the text which provides access to computer codes for all of the text figures. Through this electronic text students can:

- See the codes and run them, choosing their own input parameters interactively
- View the online numerical results as movies
- Gain an appreciation for both the dynamics of the problem application, and the growth of numerical errors
- Download and modify the code for use with other applications
- Study the code to learn how to structure their programs for modularity and ease of debugging.

John A. Trangenstein is Professor of Mathematics at Duke University, North Carolina

NUMERICAL SOLUTION OF HYPERBOLIC PARTIAL DIFFERENTIAL EQUATIONS

JOHN A. TRANGENSTEIN
Department of Mathematics, Duke University
Durham, NC 27708-0320

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

Cambridge University Press
The Edinburgh Building, Cambridge CB2 8RU, UK

Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9780521877275

First published 2007

Printed in the United Kingdom at the University Press, Cambridge

A catalog record for this publication is available from the British Library

ISBN 978-0-521-87727-5 hardback

To James A. Rowe

Contents

Preface

Hyperbolic conservation laws describe a number of interesting physical problems in diverse areas such as fluid dynamics, solid mechanics, and astrophysics. Our emphasis in this book is on nonlinearities in these problems, especially those that lead to the development of propagating discontinuities. These propagating discontinuities can appear as the familiar shock waves in gases (the "boom" from explosions or super-sonic airplanes), but share many mathematical properties with other waves that do not appear to be so "shocking" (such as steep changes in oil saturations in petroleum reservoirs). These nonlinearities require special treatment, usually by methods that are themselves nonlinear. Of course, the numerical methods in this book can be used to solve linear hyperbolic conservation laws, but our methods will not be as fast or accurate as possible for these problems. If you are only interested in *linear* hyperbolic conservation laws, you should read about spectral methods and multipole expansions.

This book grew out of a one-semester course I have taught at Duke University over the past decade. Quite frankly, it has taken me at least 10 years to develop the material into a form that I like. I may tinker with the material more in the future, because I expect that I will never be fully satisfied.

I have designed this book to describe both numerical methods and their applications. As a result, I have included substantial discussion about the analytical solution of hyperbolic conservation laws, as well as discussion about numerical methods. In this course, I have tried to cover the applications in such a way that the engineering students can see the mathematical structure that is common to all of these problem areas. With this information, I hope that they will be able to adapt new numerical methods developed for other problem areas to their own applications. I try to get the mathematics students to adopt one of the physical models for their computations during the semester, so that the numerical experiments can help them to develop physical intuition.

I also tried to discuss a variety of numerical methods in this text, so that students could see a number of competing ideas. This book does not try to favor any one particular numerical scheme, and it does not serve as a user manual to a software package. It does have software available, to allow the reader to experiment with the various ideas. But the software is not designed for easy application to new problems. Instead, I hope that the readers will learn enough from this book to make intelligent decisions on which scheme is best for their problems, as well as how to implement that scheme efficiently.

There are a number of very good books on related topics. LeVeque's *Finite Volume Methods for Hyperbolic Problems* [97] is one that covers the mathematics well, describes several important numerical methods, but emphasizes the wave propagation scheme over all. Other books are specialized for particular problem areas, such as Hirsch's *Numerical Computation of Internal and External Flows* [73], Peyret and Taylor's *Computational Methods for Fluid Flow* [131], Roache's *Computational Fluid Dynamics* [137] and Toro's *Riemann Solvers and Numerical Methods for Fluid Dynamics* [159]. These books contain very interesting techniques that are particular for fluid dynamics, and should not be ignored.

Because this text develops analytical solutions to several problems, it is possible to measure the errors in the numerical methods on interesting test problem. This relates to a point I try to emphasize in teaching the course, that it is essential in numerical computation to perform mesh refinement studies in order to make sure that the method is performing properly. Another topic in this text is that numerical methods can be compared for accuracy (error for a given mesh size) and efficiency (error for a given amount of computational time). Sometimes people have an inate bias toward higher-order methods, but this may not be the most cost-effective approach for many problem. Efficiency is tricky to measure, because subtle programming issues can drive up computational time. I do not claim to have produced the most efficient version of any of the schemes in this text, so the efficiency comparisons should be taken "with a grain of salt."

The numerical comparisons produced some surprises for me. For example, I was surprised that approximate Riemann problem solvers often produce better numerical results in Godunov methods than "exact" Riemann solvers. Another surprise is that there is no clear best scheme or worst scheme in this text (although I have omitted discussions of schemes that have fallen out of favor in the literature for good reasons). There are some schemes that generally work better than most and some that often are less efficient than most, but all schemes have their niche in which they perform well. The journal literature, of course, is full of examples of the latter behavior, since the authors get to choose computational examples that benefit their method.

During the past ten years, I have watched numerical methods evolve, computers gain amazing speed, and students struggle harder with programming. The evolution of the methods lead me to develop the course material into a form that students could access online. In that way, I could insert additional text for ready access by the students. The speed of current desktop machines allows us to make some reasonably interesting computations during the semester, seeing in a few minutes what used to require overnight runs on supercomputers. During that time, however, the new operating systems have separated the students ever farther from programming details.

As I gained experience with online text generation, I started to ask if it would be possible to develop an interactive text. First, I wanted students to be able to view the example programs while they were reading the text online. Next, I wanted students to be able to examine links to information available on the web. Then, I decided that it would be really nice if students could perform "what if" experiments within the text, by running numerical methods with different parameters and seeing the results immediately. Because I continue to think that only "real" programming languages (*i.e.*, C, C^{++} and Fortran) should be used for the material such as this, I rejected suggestions that I rewrite the programs in Matlab or Java. Eventually, our department systems programmer, Andrew Schretter, found a way to make things work for me, provided that I arrange for all parameter entry through graphical user interfaces. Our senior systems programmer, Yunliang Yu, did a lot of the development of the early form of the graphical user interface. One of my former graduate students, Wenjun Ying, programmed carefully the many cases for the marching cubes algorithm for visualizing level surfaces in three dimensions. I am greatly indebted to Andrew, Wenjun and Yunliang for their help.

This text is being published in two forms: traditional paper copy and a PDF file on a companion CD. The electronic form of the text contains links between equation or theorem references and the original statements. Similar links lead to bibliography citations or to occurrences of key words in the index. There are electronic links in the online text to source code and executables on the CD. This allows students to view computer implementations of the algorithms developed in the book, and to perform "what if" experiments with program and model parameters. However, since the text is the same for both versions of the book, this means that the paper text contains instructions to click on electronic links.

The graphical user interface (GUI) makes it easy for students to change parameters (and, in fact, to see all of the input parameters). The GUI also complicates the online programs. There is a danger that students may think that they have to program GUI's in order to solve these problems. That is not my intent. I have provided several example programs in the online version of chapter 2 to show students how they can write simple programs (that produce data sets for post processing) or slightly more complex programs (that display numerical results during the computation to look like movies), or very sophisticated programs (that use GUI's for input parameters). I would be happy if all students could program successfully in the first style. After all, CLAWPACK is a very successful example of that simple and direct style of programming.

It is common that students in this class are taking it in order to learn programming in Fortran or C^{++}, as much as they want to learn about the numerical methods. Both of these languages have advantages and disadvantages. Fortran is very good with arrays (subscripts can start at arbitrary values, which is useful for "ghost cells" in many methods) and has a very large set of intrinsic functions (for example, max and min with more than two arguments for slope limiters). Fortran is not very good with memory allocation, or with pointers in general. I use C^{++} to perform all memory allocation, and for all interactive graphics, including GUIs. When users select numerical methods through a GUI, then I set values for function pointers and pass those as arguments to Fortran routines. I do not recommend such practices for novice programmers. On the other hand, students who want to expand their programming skills can find several interesting techniques in the codes.

I do try to emphasize **defensive programming** when I teach courses that involve scientific computing. By this term, I mean the use of programming practices that make it easier to prevent or identify programming errors. It is often difficult to catch the use of uninitialized variables, the access of memory out of bounds, or memory leaks. The mixed-language programs all use the following defensive steps. First, floating-point traps are enabled in unoptimized code. Second, floating-point array values are initialized to IEEE infinity. Third, a memory debugger handles all memory allocation by overloading `operator new` in C^{++}. When the program makes an allocation request, the memory debugger gets even more space from the heap, and puts special bit patterns into the space before and after the user memory. As a result, the programmer can ask the memory debugger to check individual pointers or all pointers for writes out of bounds. This memory debugger is very fast, and does not add significantly to the overal memory requirements. The memory debugger also informs the programmer about memory leaks, providing information about where the unfreed pointer was allocated.

Unfortunately, mixing Fortran and C^{++} allows the possibility of truly bizarre programming errors. For example, declaring a Fortran subroutine to have a return value in a C^{++} `extern "C"` block can lead to stack corruption. I don't have a good defensive programming technique for that error.

But this book is really about numerical methods, not programming. I became interested in hyperbolic conservation laws well after graduate school, and I am indebted to several people for helping me to develop that interest. John Bell and

Gregory Shubin were particularly helpful when we worked together at Exxon Production Research. At Lawrence Livermore National Laboratory, I learned much about Godunov methods from both John Bell and Phil Colella, and about object oriented programming from Bill Crutchfield and Mike Welcome. I want to thank all of them for their kind assistance during our years together.

Finally, emotional support throughout a project of this sort is essential. I want to thank my wife, Becky, for all her love and understanding throughout our years together. I could not have written this book without her.