

An Interactive Course in Numerical Methods for the Earth Sciences

Susan E. Allen

Department of Earth and Ocean Sciences, University of British Columbia, Vancouver,
British Columbia, Canada

John M. Stockie

Department of Mathematics, University of British Columbia, Vancouver, British
Columbia, Canada

Philip H. Austin

Atmospheric Sciences Programme, Department of Geography, University of British
Columbia, Vancouver, British Columbia, Canada

Short title:

Abstract.

Physical oceanography and atmospheric science are becoming more and more dependent on numerical simulation, modelling, and computationally intensive data analysis. Traditionally, our students received a strong background in mathematics, physics and the Earth sciences but little background in modelling. They arrive with a variety of backgrounds but require a hands-on acquaintance with a wide range of practical numerical methods fairly early in their graduate education. We have written a self-paced, computer-based course at the first year graduate/last year undergraduate level on numerical modelling for Earth Science students. The course is Web-based with explanations, interactive examples, and codes available on-line. Students are required to investigate numerical solutions and modify existing skeleton code. Upon completing the course they have an understanding of the properties of numerical schemes and a set of numerical tools directly relevant to their research.

1. Introduction

Many researchers in physical oceanography or atmospheric science will at some time employ computer simulations in their work. The computer codes used to model physical processes in the atmosphere and oceans are typically large and complex due to the corresponding complexity of the underlying mathematical models. It is a difficult and time-consuming process to write such code entirely from scratch; in fact, it is not necessary to do so, since many reliable and multipurpose numerical libraries are available that allow the scientist to avoid some of the tricky computational details and concentrate on the implementation of the physical model. Nevertheless, it is very important that researchers using these codes have a sound understanding of the properties and behaviour of the underlying numerical schemes. Critical thinking skills are required to make the appropriate choice of model and parameters and to judge the resulting model output.

Graduate students arriving at the University of British Columbia to begin research in the Earth sciences have a broad range of scientific backgrounds and computer experience. Undergraduate degrees are typically in physics, mathematics, chemistry, biology or geography, and exposure to numerical computing ranges from zero in many cases to some students (Ph.D. transfers from physics departments, for example) who may be relatively sophisticated computer users. At UBC, students can take third and fourth year courses in numerical methods using MatlabTM as the programming language, or a graduate-level mathematics course with an emphasis on the theory behind the algorithms.

Over the past five years, two of us (SEA and PHA) have spent many hours informally tutoring graduate students on the practical use of numerical methods to solve Earth sciences research problems. We saw the need to supplement the standard university courses by teaching basic numerical methods for ordinary and partial differential equations, while at the same time providing concrete experience with large,

multi-purpose numerical libraries to solve real problems. At the end of our supplemental course, we hoped that the students would not only be comfortable with fundamental issues like discretisation, accuracy, and stability, but also have a set of numerical tools that were directly relevant to their thesis research, along with proficiency in the use of the Unix operating system and some idea of how to write robust numerical programs.

In this article, we describe our implementation of a self-paced course designed to meet these objectives using the World Wide Web¹. In Section 2 we provide some additional background on the course objectives and discuss the design decisions made to implement the course. Section 3 gives details on the topics covered and software used during the first (1995-1996) year of the course. In Section 4 we discuss student response as well as additions and changes introduced for the 1996-1997 academic year. Section 5 outlines future plans.

2. Course Objectives

Before describing how the real course was implemented, it is useful to list the characteristics of our ideal course. We believe that such a course should:

1. Be self-paced to accommodate the wide range of backgrounds of incoming graduate students;
2. Provide an introduction to the most common numerical techniques, with emphasis on techniques we have found useful in our own work; and
3. Teach students how to organise projects, debug code, and visualise results.

Table 1.

In addition to these three main points, we also gave high priority to implementing the course using public domain software. A list of the software mentioned in this article

¹A brief glossary of Internet terms is included at the end of the article

is given in Table 1. Using public domain software has the advantage of acquainting students with useful software archives such as Netlib (home of Lapack and FFTpack), while letting them read cleanly written, well-organised source code (itself a useful teaching tool). It also permits motivated students to work on the course on their home computers, and frees funds so that almost all development money can be used to hire talented students on co-operative work terms to develop new lab modules. An additional objective of the project was to provide these student interns, who would in large part write both the software and the lab descriptions, with a challenging and rewarding experience learning and teaching numerical methods.

It was important that the tools used by the students and the environment they worked in closely matched our research computing environment. Half of the students' course grade would come from an independent project, and we hoped that this project (selected in consultation with the students' advisors) would be directly applicable to the students' research. Less obviously, two of us (SEA and PHA) had to steal time for course development from normal teaching and research. This is much easier if there is a close connection between the course software and our own research tools.

As a part of our third objective (organisation, debugging, and visualisation), we decided to provide a brief introduction to the idea of object-oriented programming, both because languages like *C++* and Java are gradually becoming more popular within the scientific community, and because the new perspective offered by object-oriented programming forces even experienced student programmers to ask useful questions about the organisation of computer programs. As part of the course, students would be required to modify or extend existing *C++* classes (for example to change the numerical solution technique from Euler to Runge-Kutta). Informal instructor demonstrations could include the use of a source code debugger and object browser, and mundane but practical tips on the use of makefiles and the revision control system. Two other significant practical advantages to the choice of *C++* were the availability of a free

compiler and debugger (GNU gcc and gdb), and access to engineering and computer science co-op students with several years of *C++* programming experience.

3. Course structure and implementation

Table 2.

We proposed the idea of a self-paced Web course to the University administration in the fall of 1994 and were given \$Can 28,475 through the UBC Teaching and Learning Enhancement Fund. The grant paid the salary of a graduate student supervisor (JMS) and three additional students for the four summer months of 1995. By the end of the summer, the group had produced eight, self-paced Web modules. Table 2 itemises the topics in each module. The first two labs can be done directly from the Web page using standard CGI (Common Gateway Interface) scripts written in the Perl5 scripting language. Lab 3 introduces the students to Octave, a free Matlab clone written in *C++* that uses Lapack linear algebra routines. Beginning in Lab 4 students are asked to modify and run *C++* programs that solve ordinary and partial differential equations. By the time the students reach Lab 8, they have had an introduction to explicit and implicit schemes and to choosing staggered grids. A subset of the objectives for the labs is given in Table 3.

Table 3.

Each lab module is a self-contained HTML document including all the requisite background material, examples, and written and programmed exercises. The material is highly cross-referenced, making full use of hypertext links in HTML to reference other labs, program code and other documents on the Internet.

Typically, the student is introduced to a particular numerical technique first by way of an interactive example, where they can play with the input parameters for a problem and observe the numerical results displayed in graphical format. Next, the student writes his or her own program, usually by modifying an existing skeleton code in order to investigate the numerical issues more deeply.

Figure 1.

Some of the features of the labs are demonstrated in Figure 1, which is a typical

Web page (as viewed from within the Netscape Navigator). This particular page is taken from Lab #3, and shows hyperlinked equations, a problem and the “navigation panel” which appears at the top and bottom of every page. It includes, in order from left to right, links to the “previous,” “up” and “next” sectional units; the lab table of contents; a glossary; a searchable index; and a mail button to send e-mail to the course administrator. The “eye” icon transfers the student to an interactive demonstration, Figure 2, while a “computer” icon would transfer the reader to technical details in an appendix, which enhances the self-documenting nature of the labs.

Figure 2.

Figure 2 shows the interactive demonstration referenced by the problem in Figure 1. The top of the page repeats the problem. The lower section allows the student to specify three parameters of the system. Figure 3 shows the result obtained after the demonstration has been run. Two graphs are shown and further down the page (only partially shown here) are the numerical values of the variables at the final time step. To answer the problem, students run the demonstration with various parameters. The graphs allow the students to verify that the system has reached a steady state.

Figure 3.

The labs are numbered consecutively, but need not be taught in the exact order given. A suggested ordering of the material is given in Figure 4, from which it is clear that the labs divide nicely into two streams, one concerned with ODE's and the other with PDE's. It is also straightforward to create additional labs, and to modify or extend existing ones, using a “lab template” document, making the course flexible and extensible.

Figure 4.

4. The Course Trial

As the course is self-paced and web-based, there are no formal lectures. The students taking the course work on their own: read the material either on a computer, or more often, on a paper print-out of the lab, work through the interactive examples on a computer and prepare their solutions to the assigned problems. Most students work

on their research supervisor's computer. However, for some students it is necessary to access remotely the instructor's machines to use software that is not locally available. Although most of the work is done on their own, students have the opportunity to interact in a number of ways with the instructors and the other students. There is a weekly informal meeting. Most of these meetings are question and answer periods but two to three times each term, one of the instructors presents a mini-lecture to supplement the course material. There is an electronic mailing list where students ask questions and post difficulties they are having. All students meet one on one with the instructors at some point. In addition, students help each other either through their own informal groups or through an instructor's suggestion. The last part of the course is a term project, selected in consultation with the instructors and the student's research supervisor.

The course was first offered during the academic year 1995–96 to a group of nine graduate students. The course was originally intended to be a single-semester (four-month) offering. However, the length of time required for some of the lab exercises (particularly from Labs 7 & 8) together with the project pushed the course into a full two semesters of work for most students. The student projects have been of uniformly high quality, written in C, Matlab, and *C++*. They include a finite difference model for ripple formation in a sandy stream bed, a model of ice intrusion into sand beds, and a finite difference solution of a seismic travel time problem.

A questionnaire was distributed to the participants at the end of the academic year to solicit opinions on the course. Overall, the response was very positive. Most students had a favourable opinion of the self-paced nature of the course, though this was tempered by comments that it was exactly this that caused them to take so much time to finish the assigned work. Students reacted favourably to the hypertext presentation of the course material and examples, but were also quite glad to have the option of producing an identical hardcopy that they could annotate. The major complaint was

that the labs (particularly the later ones) were too long.

We were encouraged enough by our first-year experience to apply for additional funding through the British Columbia Innovation Fund to support two students to extend and modify the labs during the summer of 1996. The *C++* code was streamlined to remove some of the class hierarchies, general documentation and examples were added, and preliminary versions of new labs dealing with Fourier transforms, flux-conserved advection, and non-linear optimisation were written. These optional labs will provide additional choices for students who don't need the detailed treatment of the quasi-geostrophic model provided in Lab 8.

This fall we are testing a new fourth-year undergraduate version of the course. It has shorter exercises, well-defined weekly assignments and fixed due dates. The project portion of the course will also be shorter, and specific project suggestions are available to help students get past the initial project planning stage. Undergraduates meet with the graduate students in the weekly discussion sessions, and participate in the graduate course mail-list discussions. The first-year computer language used in the UBC computer science department is *C++*, so in this sense the undergraduates may be better prepared than graduate students arriving with a background in Pascal or Fortran.

5. Discussion

Our first venture into Web-based instruction has met the objectives we set for ourselves in the winter of 1995. We were particularly impressed by the undergraduate student interns hired to develop the lab modules; their productivity allowed us to extend the scope of the course substantially beyond our initial objectives, and they proved to be patient teachers as we learned *C++* by watching them program. We are also pleased that project work by the first class of students has directly benefited student research (and in at least one case, will be submitted to a peer-reviewed publication).

The rapid evolution of the Internet has meant that we now have several more free

alternatives to *C++* and Octave, including languages such as Python, Java, F, and Elf90 that run on Windows95 as well as Unix. Further undergraduate expansion of the course will require this kind of portability, as we place our Web-materials in UBC's drop-in computer labs and on students' home PCs. Windows95 ports are also underway for the *C++* compiler and debugger, as well as all of the other Unix tools we use. We plan to continue to expand the topic coverage, as other faculty members use our template examples to develop Web-modules for topics that interest them. Our decision to use free software also means that the course, including code, all software tools, and a free version of the Unix operating system (Linux), can be distributed to students for the cost of a blank CDROM.

A more detailed discussion of the lab templates, the software and hardware requirements for our labs, and access to the labs themselves are available at <http://www.geog.ubc.ca/numeric>. Readers are invited to forward comments/questions to us at numerical_methods@geog.ubc.ca. We would be interested in sharing experiences with others who are beginning to explore instruction via the Internet.

Glossary

CGI: (Common Gateway Interface) A protocol for communicating between programs and the hypertext viewer.

hypertext: A collection of electronic documents (text, images, audio, video, etc.) which are joined together by "hyperlinks." The links in a hypertext document can be read in any order, as compared to normal text which is meant to be read sequentially, left to right, top to bottom.

hyperlink: A link to another document or file that appears in a hypertext document.

HTML: Hypertext Markup Language, used to display documents on the Web. It consists of normal text, along with embedded codes or "tags" which tell a Web

browser how to display the text and how to link to other documents.

Web: (World Wide Web) An Internet based information retrieval system based on hypertext.

Acknowledgments.

The course was developed through grants from the UBC Teaching and Learning Enhancement Fund and the Innovation fund of the British Columbia Ministry of Education, Skills, and Training. Programming and writing of the labs was done by Peter Gorniak, Carmen Guo, Ken Wong, Lin Yang and Grace Yung as well as by the authors. The authors thank Vincent Kujula, Jim Mintha and Joseph Tam for technical support. We are grateful for the reviewers' comments, which improved the clarity of the text.

Susan E. Allen, Department of Earth and Ocean Sciences, 6339 Stores Rd, University of British Columbia, Vancouver, B.C. V6T 1Z4 CANADA (email: allen@eos.ubc.ca)

John M. Stockie, Department of Mathematics, 1984 Mathematics Road, University of British Columbia, Vancouver, B.C. V6T 1Z2 CANADA (email: stockie@math.ubc.ca)

Philip H. Austin, Atmospheric Sciences Programme, #217 Geography, 1984 West Mall, University of British Columbia, Vancouver, B.C. V6T 1Z2 CANADA (email: phil@geog.ubc.ca)

Received October 20, 1996

Submitted to Bull. Amer. Met. Soc., 96/10/18

This manuscript was prepared with AGU's \LaTeX macros v4, with the extension package 'AGU++' by P. W. Daly, version 1.5a from 1996/10/09.

Figure Captions

Figure 1. A sample lab page taken from Laboratory #5.

Figure 2. The Lab #5 interactive demonstration on Daisywold steady states.

Figure 3. Results from a run of the steady state interactive example.

Figure 4. Suggested ordering of material. The arrows indicate which labs are required for following labs.

Tables

Software package	Trademark/copyright holder	Web site
Matlab	The MathWorks, Inc.	www.mathworks.com
LAPACK, FFTPACK	Public Domain	www.netlib.org
Unix	X/Open Company	xoweb.xopen.org
Java	Sun Microsystems Inc.	java.sun.com
Perl	Free Software Foundation, Inc. (FSF)	www.perl.com/perl
gcc, gdb	FSF	www.gnu.ai.mit.edu
Netscape Navigator	Netscape Communications Corp.	www.netscape.com
Octave	FSF	www.che.wisc.edu/octave
Windows 95	Microsoft Corp.	www.microsoft.com
Python	Stichting Mathematisch Centrum	www.python.org
Elf90	Lahey Computer Systems, Inc.	www.lahey.com
F	Imagine1, Inc.	www.imagine1.com/imagine1

Table 1. A list of the software mentioned in this article, its copyright owner and a source location on the Web.

Lab	Topic	Tool
1	Discretisation	CGI examples
2	Accuracy and Stability	CGI examples
3	Linear Algebra	Octave/Lapack
4	Adaptive Runge-Kutta I	<i>C++</i>
5	Adaptive Runge-Kutta II	<i>C++</i>
6	The Lorenz Equations	<i>C++</i>
7	Partial Differential Equations	Octave, <i>C++</i>
8	Quasi-Geostrophic Model	<i>C++</i>

Table 2. Eight initial instructional modules, with the main programming tools used for each lab

Objective	Lab
Discretise a continuous problem	1
Define accuracy and stability	2
Quantify error in terms of order	2
Use a linear algebra package to invert matrices, find eigenvalues <i>etc.</i>	3
Program in <i>C++</i> using the <i>mv++</i> class libraries	4
Use a Runge-Kutta method	4
Implement adaptive step-size control	5
Describe some of the effects of nonlinearity in ODE's including chaos	6
Explain the usefulness of staggered grids for PDE's	7
State the CFL condition on explicit scheme stability	7
Implement relaxation	8
Describe nonlinear instability and a method to remove it	8

Table 3. Examples of objectives for specific labs.

Figures

The screenshot shows a web browser window with a menu bar (File, Edit, View, Go, Bookmarks, Options, Directory, Window, Help) and a toolbar with icons for Back, Forward, Home, Reload, Images, Open, Print, Find, and Stop. The address bar shows the location: `file:/nfs/peacock/homes/numeric/labs/lab5/lab5/node10.html`. Below the address bar are buttons for "What's New?", "What's Cool?", "Destinations", "Net Search", "People", and "Software".

The main content area displays the text "temperature T_i " followed by the equation:

$$\beta_i = 1.0 - 0.003265(295.5K - T_i)^2$$

Below the equation, it states: "If the conductivity R is non-zero, the local temperature is a function of planetary albedo α_p (3.10)".

The next equation is:

$$T_i = \left[RL \frac{S_0}{4\sigma} (\alpha_p - \alpha_i) + T_r^4 \right]^{\frac{1}{4}}$$

which is determined by the daisy population (3.11).

A bulleted list follows:

- Physically, this provides the feedback from the daisy population back to the temperature, completing the loop between the daisies and temperature.
- Mathematically, this introduces a rather nasty non-linearity into the equations which, as pointed out in the lab 1, usually makes it difficult, if not impossible, to obtain exact analytic solutions.

A horizontal line separates this from the "Problem" section:

Problem: The feedback means a stable daisy population (a steady state) and the environmental conditions are in a delicate balance. For the steady state which arises from a given initial daisy population,

- Add a small initial fraction of black daisies (say, 0.01) and see what effect this has on the temperature and final daisy populations. Do you still have a final non-zero daisy population?
- Attempt to adjust the initial white daisy population to obtain a non-zero steady state. Do you have to increase or decrease the initial fraction? What is your explanation for this behavior?
- Experiment with other initial fractions of daisies and look for non-zero steady states.

Below the list is a small image of a daisy and a link: [Daisyworld steady states and initial conditions.](#)

At the bottom, there are navigation buttons: "Previous", "Up", "Next", "Contents", "A to Z", and a search icon. Below these are links: "Previous: [The Local Temperature -](#)", "Up: [Daisyworld](#)", and "Next: [Daisyworld Class](#)".

Figure 1. A sample lab page taken from Laboratory #5.

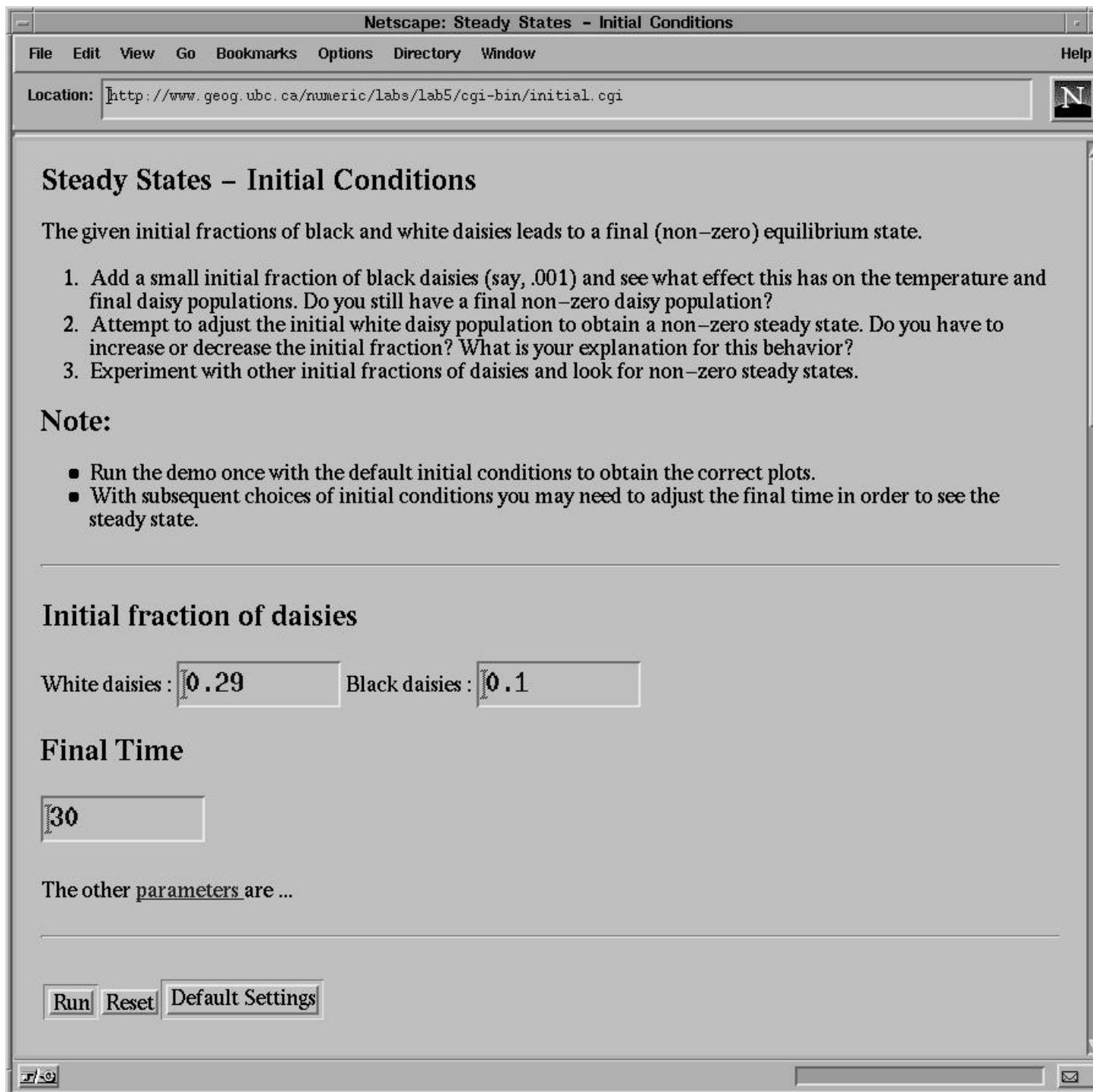


Figure 2. The Lab #5 interactive demonstration on Daisywold steady states.

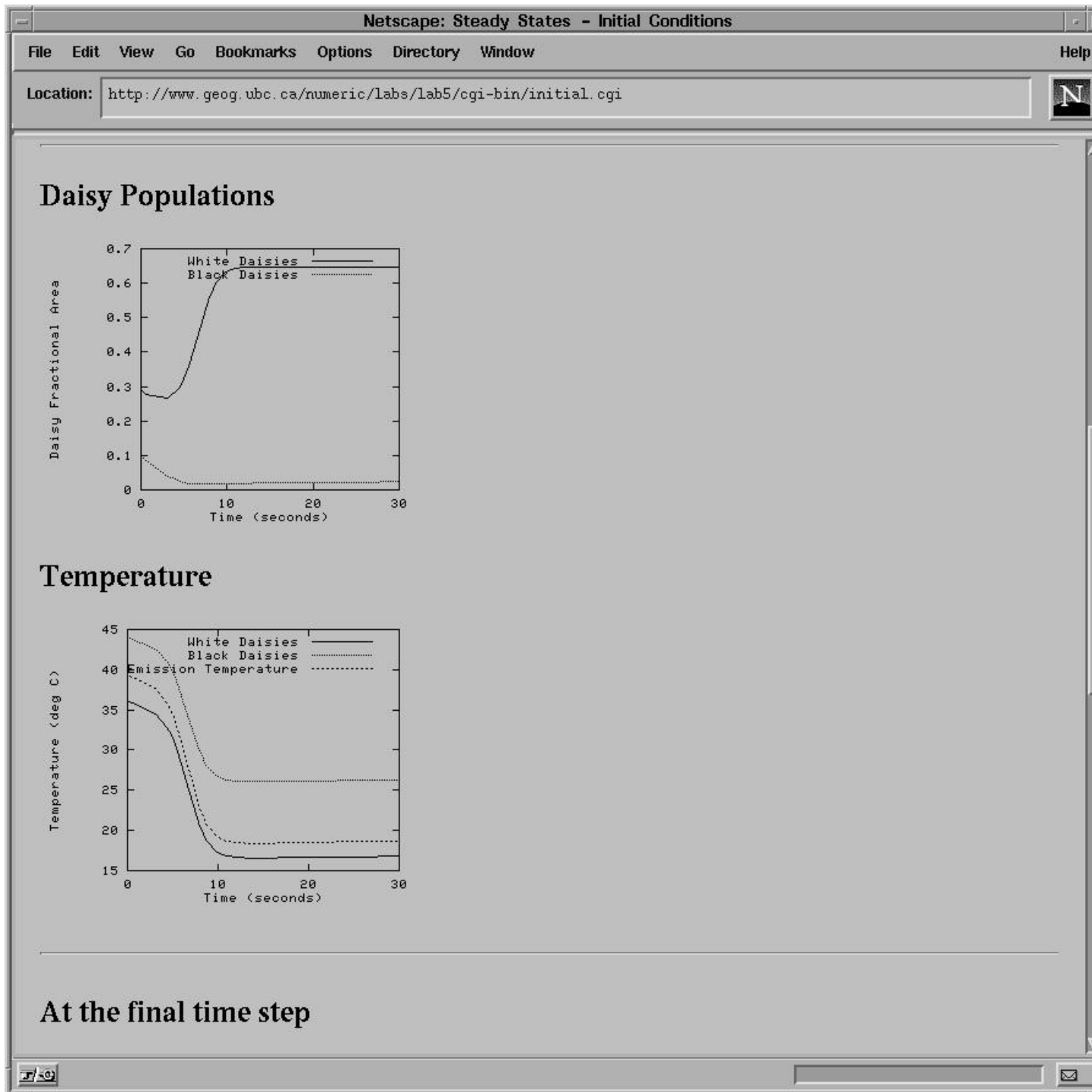


Figure 3. Results from a run of the steady state interactive example.

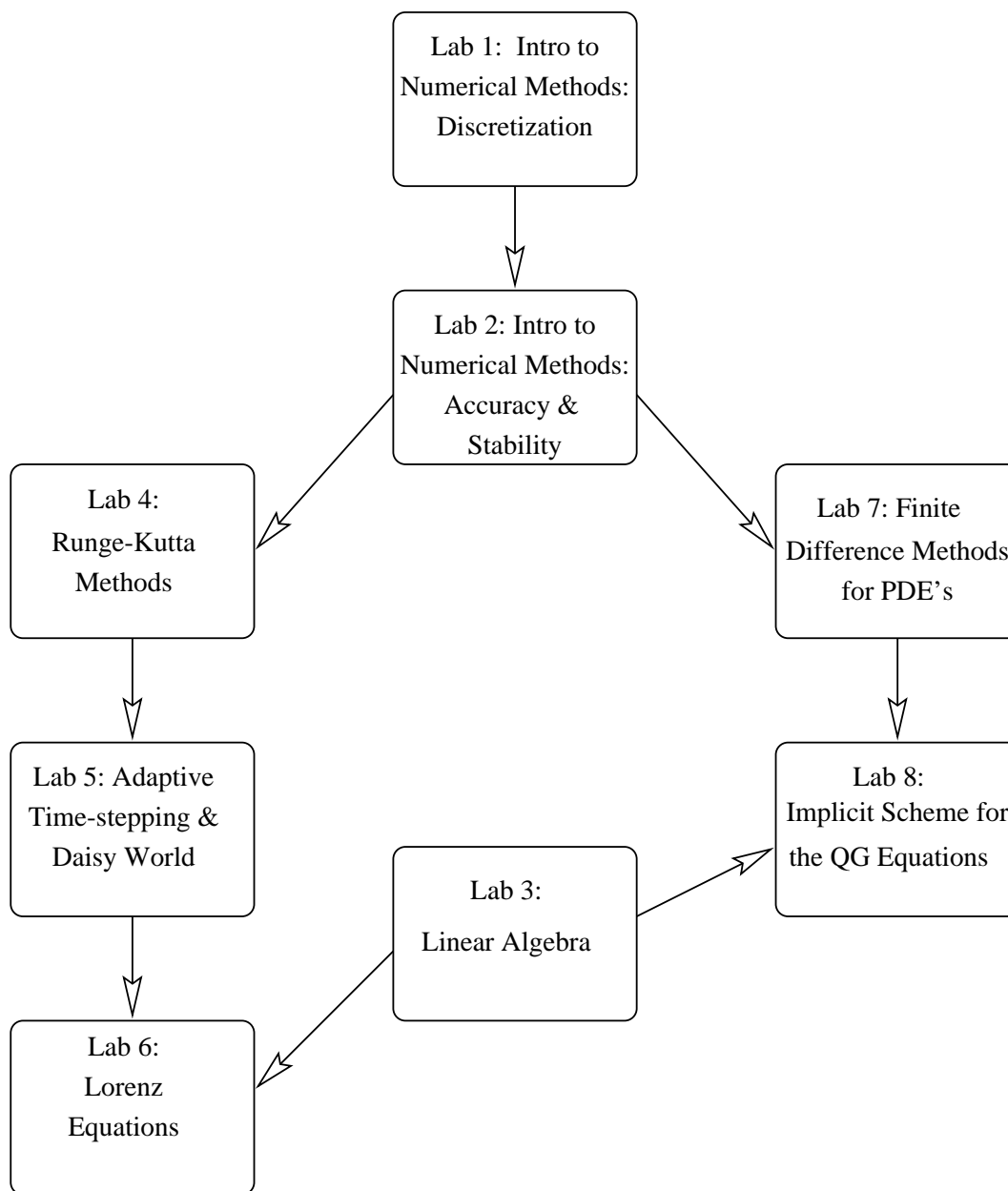


Figure 4. Suggested ordering of material. The arrows indicate which labs are required for following labs.