NUMERICAL METHODS

DESIGN, ANALYSIS, AND COMPUTER IMPLEMENTATION OF ALGORITHMS

ANNE GREENBAUM & TIMOTHY P. CHARTIER

NUMERICAL METHODS

NUMERICAL METHODS

DESIGN, ANALYSIS, AND COMPUTER IMPLEMENTATION OF ALGORITHMS

ANNE GREENBAUM
TIMOTHY P. CHARTIER

PRINCETON UNIVERSITY PRESS • PRINCETON AND OXFORD

Published by Princeton University Press, 41 William Street, Princeton, New Jersey 08540

In the United Kingdom: Princeton University Press, 6 Oxford Street, Woodstock, Oxfordshire OX20 1TW

press.princeton.edu

Cover image courtesy of the University of Sheffield and Ansys UK.

Library of Congress Cataloging-in-Publication Data

Greenbaum, Anne.
Numerical methods: design, analysis, and computer implementation of algorithms / Anne Greenbaum, Timothy P. Chartier.
p. cm.
ISBN 978-0-691-15122-9 (hbk. : alk. paper)
1. Numerical analysis. I. Chartier, Timothy P., 1969–
II. Title.
QA297.G15 2012
518–dc23

2011045732

British Library Cataloging-in-Publication Data is available

This book has been composed in Sabon
Printed on acid-free paper. ∞

Typeset by S R Nova Pvt Ltd, Bangalore, India
Printed in the United States of America

10 9 8 7 6 5 4

To Tanya

for her constant love and support,

whether life is well conditioned or unstable.

—TC

CONTENTS

PREFACE

In this book we have attempted to integrate a reasonably rigorous mathematical treatment of elementary numerical analysis with motivating examples and applications as well as some historical background. It is designed for use as an upper division undergraduate textbook for a course in numerical analysis that could be in a mathematics department, a computer science department, or a related area. It is assumed that the students have had a calculus course, and have seen Taylor's theorem, although this is reviewed in the text. It is also assumed that they have had a linear algebra course. Parts of the material require multivariable calculus, although these parts could be omitted. Different aspects of the subject—design, analysis, and computer implementation of algorithms—can be stressed depending on the interests, background, and abilities of the students.

We begin with a chapter on mathematical modeling to make the reader aware of where numerical computing problems arise and the many uses of numerical methods. In a numerical analysis course, one might go through all or some of the applications in this chapter or one might just assign it to students to read. Next is a chapter on the basics of MATLAB [94], which is used throughout the book for sample programs and exercises. Another high-level language such as SAGE [93] could be substituted, as long as it is a language that allows easy implementation of high-level linear algebra procedures such as solving a system of linear equations or computing a QR decomposition. This frees the student to concentrate on the use and behavior of these procedures rather than the details of their programming, although the major aspects of their implementation are covered in the text in order to explain proper interpretation of the results.

The next chapter is a brief introduction to Monte Carlo methods. Monte Carlo methods usually are not covered in numerical analysis courses, but they should be. They are *very* widely used computing techniques and demonstrate the close connection between mathematical modeling and numerical methods. The basic statistics needed to understand the results will be useful to students in almost any field that they enter.

The next chapters contain more standard topics in numerical analysis—solution of a single nonlinear equation in one unknown, floating-point arithmetic, conditioning of problems and stability of algorithms, solution of linear systems and least squares problems, and polynomial and piecewise polynomial interpolation. Most of this material is standard, but we do include some recent results about the efficacy of polynomial interpolation when the interpolation points are *Chebyshev points*. We demonstrate the use of a MATLAB software package called `chebfun` that performs such interpolation, choosing the degree of the interpolating polynomial adaptively to attain a level of accuracy near the machine precision. In the next two chapters, we discuss the application of this approach to numerical differentiation and integration.

We have found that the material through polynomial and piecewise polynomial interpolation can typically be covered in a quarter, while a semester course would include numerical differentiation and integration as well and perhaps some material on the numerical solution of ordinary differential equations (ODEs). Appendix A covers background material on linear algebra that is often needed for review.

The remaining chapters of the book are geared towards the numerical solution of differential equations. There is a chapter on the numerical solution of the initial value problem for ordinary differential equations. This includes a short section on solving systems of nonlinear equations, which should be an easy generalization of the material on solving a single nonlinear equation, assuming that the students have had multivariable calculus. The basic Taylor's theorem in multidimensions is included in Appendix B. At this point in a year long sequence, we usually cover material from the chapter entitled "More Numerical Linear Algebra," including iterative methods for eigenvalue problems and for solving large linear systems. Next come two-point boundary value problems and the numerical solution of partial differential equations (PDEs). Here we include material on the fast Fourier transform (FFT), as it is used in fast solvers for Poisson's equation. The FFT is also an integral part of the `chebfun` package introduced earlier, so we are now able to tell the reader a little more about how the polynomial interpolation procedures used there can be implemented efficiently.

One can arrange a sequence in which each quarter (or semester) depends on the previous one, but it is also fairly easy to arrange independent courses for each topic. This requires a review of MATLAB at the start of each course and usually a review of Taylor's theorem with remainder plus a small amount of additional material from previous chapters, but the amount required from, for example, the linear algebra sections in order to cover, say, the ODE sections is small and can usually be fit into such a course.

We have attempted to draw on the popularity of mathematical modeling in a variety of new applications, such as movie animation and information retrieval, to demonstrate the importance of numerical methods, not just in engineering and scientific computing, but in many other areas as well. Through a variety of examples and exercises, we hope to demonstrate some of the many, many uses of numerical methods, while maintaining the emphasis on analysis and understanding of results. Exercises seldom consist of simply computing an answer; in most cases a computational problem is combined with a question about convergence, order of accuracy, or effects of roundoff. Always an underlying theme is, "How much confidence do you have in your computed result?" We hope that the blend of exciting new applications with old-fashioned analysis will prove a successful one. Software that is needed for some of the exercises can be downloaded from the book's web page, via http://press.princeton.edu/titles/9763.html. Also provided on that site are most of the MATLAB codes used to produce the examples throughout the book.

Acknowledgments. The authors thank Richard Neidinger for his contributions and insights after using drafts of the text in his teaching at Davidson College. We also thank the Davidson College students who contributed ideas for improving the text, with special thanks to Daniel Orr for his contributions to the exercises. Additional exercises were contributed by Peter Blossey and Randall LeVeque of the University of Washington. We also thank Danny Kaplan of Macalester College for using an early version of the text in his classes there, and we thank Dan Goldman for information about the use of numerical methods in special effects.

NUMERICAL METHODS

1

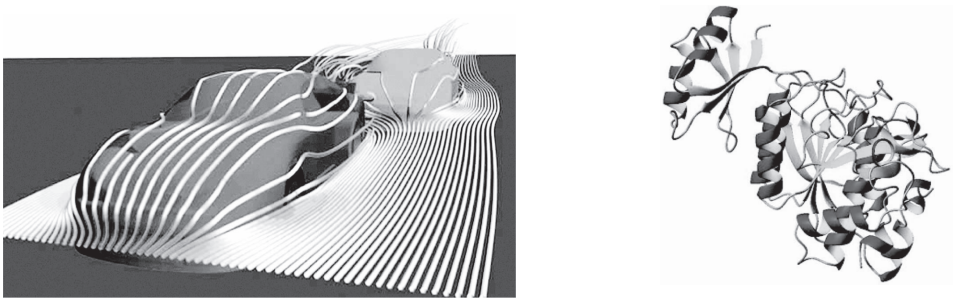
MATHEMATICAL MODELING

Numerical methods play an important role in modern science. Scientific exploration is often conducted on computers rather than laboratory equipment. While it is rarely meant to completely replace work in the scientific laboratory, computer simulation often complements this work.

For example, the aerodynamic simulation of two NASCAR autos pictured in figure 1.1(a) requires the numerical solution of *partial differential equations* (PDEs) that model the flow of air past the car. An auto body must be smooth and sleek, so it is often modeled using cubic (or higher order) *splines*. Similar computations are done in designing aircraft. We will study the numerical issues in using splines and solving PDEs in chapters 8 and 14, respectively.

Other examples occur in the field of mathematical biology, an active area of research in industry, government, and academia. Numerical algorithms play a crucial role in this work. For example, protein-folding models are often solved as large *optimization* problems. Protein arranges itself in such a way as to minimize energy—nature has no trouble finding the right arrangement, but it is not so easy for humans. The field of numerical optimization is an entire subject on its own, so it will not be covered in this book. The numerical methods described here, however, form the core of most optimization procedures.

Before studying issues in the analysis and implementation of efficient and accurate numerical methods, we first look briefly at the topic of mathematical modeling, which turns real-world problems into the sorts of mathematical equations that numerical analysts can tackle. The mathematical formulation usually represents only a *model* of the actual physical situation, and it is often important for the numerical analyst or computational scientist to know something about the origin of the model; in fact, numerical analysts sometimes work directly with scientists and engineers in devising the mathematical model. This interaction is important for a number of reasons. First, many algorithms do not produce the exact solution but only an approximate one. An understanding of the origin of the problem is necessary to determine what constitutes an acceptably good "approximate" solution: an error of a few centimeters might be acceptable in locating an enemy tank, but it would not be acceptable in locating a tumor for laser surgery! Second, even if the algorithm theoretically produces the exact solution, when implemented on a computer using finite-precision arithmetic, the results produced will most



(a) (b)

Figure 1.1. (a) A simulation of two NASCAR autos depicts the streamlines of air produced as a car drafts and is about to pass another. (Simulation performed with STAR-CCM+.) (b) Solving protein-folding models utilizes numerical optimization.

likely be inexact. Part of numerical analysis is the understanding of the impact of finite-precision computations on the accuracy of results. We will look more deeply at the issues in computing in finite precision in chapter 5.

In this chapter we present a variety of applications that involve numerical computation and come from the mathematical modeling of various processes.

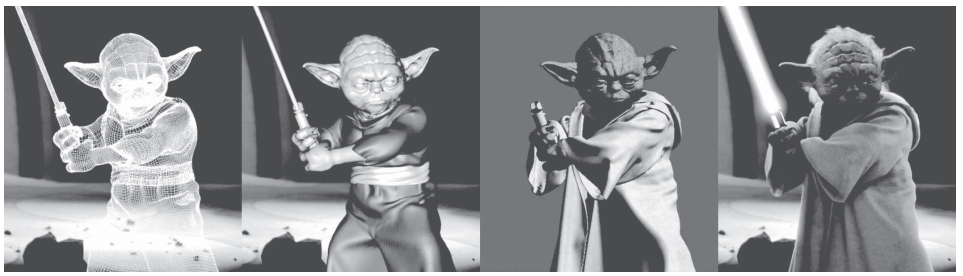
1.1 MODELING IN COMPUTER ANIMATION

Many of the computer generated graphics that dominate the silver screen are produced with **dynamic simulation**; that is, a model is created, often using the laws of physics, and numerical methods are then used to compute the results of that model. In this section, we will look at the role of numerics in animation that appeared in the 2002 film *Star Wars: Episode II Attack of the Clones*. In particular, we will take a careful look at some of the special effects used to digitally create the character of Yoda, a Jedi master who first appeared as a puppet in the Star Wars saga in the 1980 film, *The Empire Strikes Back*. In the 2002 film, Yoda was digitally created, which required heavy use of numerical algorithms.

A key aspect of creating a digital Yoda involves producing believable movement of the character. The movement of Yoda's body is described using **key-frame animation**, in which a pose is specified at particular points in time and the computer automatically determines the poses in the intervening frames through interpolation. (We will discuss several interpolation techniques in chapter 8.) Animators have many controls over such movement, with the ability to specify, for instance, velocities and tangents of motion. While animators indicate the movement of Yoda's body, the computer must determine the resulting flow of his robe.

1.1.1 A Model Robe

Referring to figure 1.2, we see that the robe is represented with triangles and the motion of each vertex of the robe must be determined. Each vertex is



(a) (b) (c) (d)

Figure 1.2. Stages of simulation in the animation of the digitally created Yoda in the fight scene with Count Dooku in *Star Wars Episode II*. Two layers of Yoda's clothing, seen in (b) and (c), were computed separately. A process known as collision detection ensured that the inner layer of clothing did not intersect the outer robe and become visible. A simplified model of cloth illumination created the appearance of a real garment, producing the final rendering of the image in (d) [22]. (Courtesy of Lucasfilm Ltd. *Star Wars: Episode II - Attack of the Clones* ™ & © 2002 Lucasfilm Ltd. All rights reserved. Used under authorization. Unauthorized duplication is a violation of applicable law. Digital Work by Industrial Light & Magic.)

modeled as a particle, which in this context is a pointlike object that has mass, position, and velocity, and responds to forces, but has no size.

The motion of a particle is governed by Newton's second law, which is expressed mathematically by the equation

$$\mathbf{F} = m\mathbf{a} = m(d^2\mathbf{y}/dt^2), \tag{1.1}$$

where $\mathbf{y}$ is a distance function of time t. Note that the equations involve vector-valued functions since our computations are performed in three dimensions. Since a particle has mass ($m \neq 0$), equation (1.1) can be rewritten as the second-order ordinary differential equation (ODE)

$$\frac{d^2\mathbf{y}}{dt^2} = \frac{\mathbf{F}}{m}. \tag{1.2}$$

This ODE is part of an initial value problem since the state of the particle at some initial time is given. In the case of a movie, this is where the scene (which may be several seconds or a fraction of a second in duration) begins.

To keep the shape of the robe, pairs of neighboring particles are attached to each other using a spring force. Hooke's law states that a spring exerts a force F_s that is proportional to its displacement from its rest length x_0. This is expressed mathematically as

$$F_s = -k(x - x_0),$$

where x denotes the current position of the spring and k is the spring constant. For simplicity, we have stated the one-dimensional formulation of Hooke's law, but to model the Jedi's robe a three-dimensional version is used.

Many other forces are computed to animate Yoda's robe, including gravity, wind forces, collision forces, friction, and even completely made-up forces that are invented solely to achieve the motion that the director requests. In the

simplest of cases, an *analytic solution* of the model may exist. In computer animation, however, the forces acting on the particle constantly change and finding an analytic solution for each frame—if even possible—would be impractical. Instead, numerical methods are used to find approximate solutions by simulating the motion of the particles over discrete time steps. There is a large body of literature on the numerical solution of initial value problems for ODEs, and some of the methods will be covered in chapter 11.

BENDING THE LAWS OF PHYSICS

When simulating the motion of Jedi robes in *Star Wars Episode II*, animators discovered that if the stunts were performed in reality, the clothing would be ripped apart by such accelerations of the motion. To solve the problem, custom "protection" effects were added to the simulations, which dampened this acceleration. In the end, the clothes on the digital actors were less distorted by their superhuman motion. To the left we see a digital double of Obi-Wan Kenobi (played by Ewan McGregor), performing a stunt too dangerous for a live actor, in *Star Wars Episode II*. Note that the hairs, like the clothing, were simulated as strips of particles connected by springs [22]. (Courtesy of Lucasfilm Ltd. *Star Wars: Episode II - Attack of the Clones* ™ & © 2002 Lucasfilm Ltd. All rights reserved. Used under authorization. Unauthorized duplication is a violation of applicable law. Digital Work by Industrial Light & Magic.)

A goal of movie animation is creating convincing simulations. Accuracy, which is a leading goal of scientific simulation, may conflict with this goal. As such, numerical simulations are used for different goals by scientists and by the entertainment industry, but many of the mathematical tools and strategies are common. Let us now turn our attention to a simulation in science.

1.2 MODELING IN PHYSICS: RADIATION TRANSPORT

The transport of radiation can be described stochastically and modeled using Monte Carlo simulation. Monte Carlo methods will be described in chapter 3. Radiation transport also can be modeled by an integro-differential equation, the Boltzmann transport equation. This is a PDE that also involves an integral

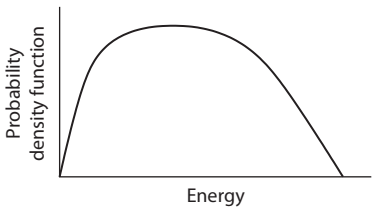


Figure 1.3. An example distribution of the energy of photons.

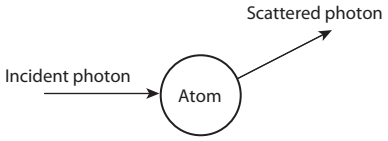


Figure 1.4. A collision between a photon and an atom.

term. In chapter 14 we discuss the numerical solution of PDEs, while chapter 10 describes methods of numerical integration. A combination of these ideas, together with iterative techniques for solving large linear systems (section 12.2), is used to approximate solutions to the Boltzmann transport equation.

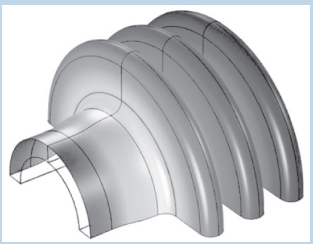
What radiation dose does your body receive from a dental X-ray examination? You probably recall that a heavy vest is usually placed over you during the exam and that the dental assistant leaves the room while the X-ray machine is on. The purpose of the covering is to absorb radiation. How can the transport of X-ray photons be modeled mathematically in order to aid in the design of such protective materials and to verify their effectiveness? The photons are produced by an electron beam that is turned on and off as X-rays are needed. The energy and direction of travel of any individual photon cannot be predicted, but the overall distribution of energy and direction of the X-rays can be approximated.

The independent variables in the system are energy, position, direction, and time of production of the photons, and each of these variables can be thought of as random but obeying a certain distribution. The energy of the photons might be distributed as shown in figure 1.3, for example.

It is assumed that particles move in straight lines until they enter matter, where, with a certain probability (depending on the characteristics of the material), they collide with an atom. The collision usually involves an exchange of energy with an electron in the atom, after which the photon emerges with reduced energy and an altered direction of travel. The photon may even give up all of its energy to the electron, in which case it is considered to be absorbed by the atom.

The probabilities of each of these events must be known in order to simulate the situation. These probabilities are deduced from theoretical and measured properties of X-rays and of various materials that might be used as shields. One can then run a computer simulation, often with millions of photons, each following a random path determined by these probability distributions. The history of each photon is followed until it is either absorbed (or loses so much energy that it effectively can be ignored) or travels outside the system to a point

from which it will not return. Average results are then taken to determine what dose of radiation is received at a particular location.



STUDYING SEMICONDUCTORS

Studying the behavior of electrons in semiconductor materials requires solving the Boltzmann transport equation, which involves complicated integrals. Both deterministic methods and Monte Carlo methods are sometimes used in this case. The picture to the left is a finite element discretization used in a deterministic model for radiation transport problems. (Image reproduced with the kind permission of the Applied Modelling and Computational Group at Imperial College London and EDF Energy. All other rights of the copyright owners are reserved.)

1.3 MODELING IN SPORTS

In FIFA World Cup soccer matches, soccer balls curve and swerve through the air, in the players' attempts to confuse goalkeepers and send the ball sailing to the back of the net. World class soccer players such as Brazil's Roberto Carlos, Germany's Michael Ballack and England's David Beckham have perfected "bending" the ball from a free kick.

According to Computational Fluid Dynamics (CFD) research by the University of Sheffield's Sports Engineering Research Group and Fluent Europe, the shape and surface of the soccer ball, as well as its initial orientation, play a fundamental role in the ball's trajectory through the air. In particular, such CFD research has increased the understanding of the "knuckleball" effect sometimes used to confuse an opposing goalkeeper who stands as the last line of defense. To obtain such results, a soccer ball was digitized down to its stitching as seen in figure 1.5. Note the refinement near the seams, which is required in order to properly model the boundary layer.

Some free kicks in soccer have an initial velocity of almost 70 mph. Wind tunnel experiments demonstrate that a soccer ball moves from laminar to turbulent flow at speeds between 20 and 30 mph, depending on the ball's surface structure and texture.

The techniques developed in Sheffield facilitated detailed analysis of the memorable goal by David Beckham of England versus Greece during the World Cup Qualifiers in 2001. In a sense, Beckham's kick applied sophisticated physics. While the CFD simulations at Sheffield can accurately model turbulent flow only when it is averaged over time, and so cannot yet give realistic trajectories in all cases, such research could affect soccer players from beginner

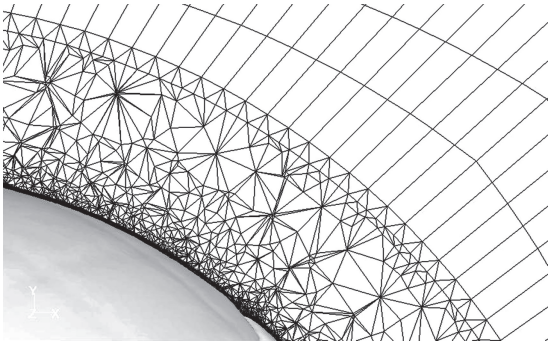
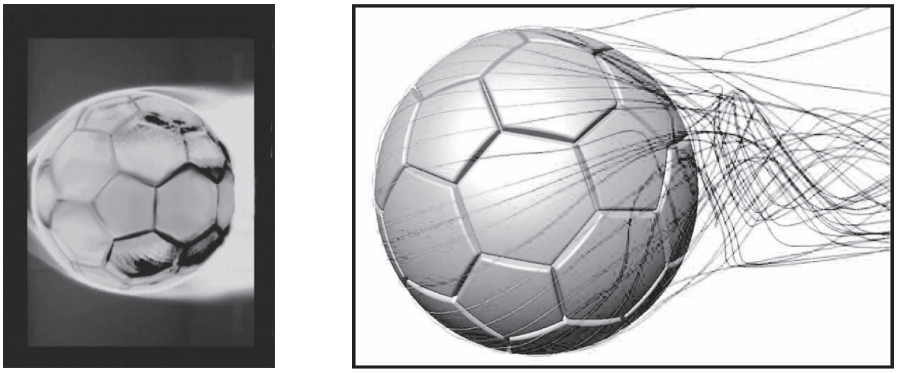


Figure 1.5. An important step in CFD simulations at the University of Sheffield is capturing the geometry of a soccer ball with a three-dimensional noncontact laser scanner. The figure shows part of a soccer ball mesh with approximately 9 million cells. (Courtesy of the University of Sheffield and Ansys UK.)



(a) (b)

Figure 1.6. (a) Wind tunnel smoke test of a nonspinning soccer ball. (b) CFD simulation showing wake-flow path lines of a nonspinning soccer ball, air speed of 27 mph. (Courtesy of the University of Sheffield and Ansys UK.)

to professional. For instance, ball manufacturers could exploit such work to produce a more consistent or interesting ball that could be tailored to the needs and levels of players. Such work could also impact the training of players. For more information, see, for instance [6] or [7].

To this end, there is a simulation program called Soccer Sim developed at the University of Sheffield. The program predicts the flight of a ball given input conditions, which can be acquired from CFD and wind tunnel tests, as well as from high-speed videoing of players' kicks. The software can then be used to compare the trajectory of a ball given varying initial orientations of the ball or different spins induced by the kick. Moreover, the trajectory can be compared for different soccer balls.

Note that this application, like that in section 1.1, involves the solution of differential equations. The next application that we discuss involves a discrete phenomenon.

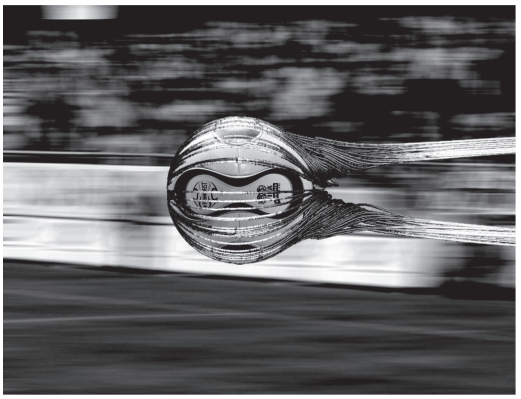


Figure 1.7. High-speed airflow path lines colored by local velocity over the 2006 Teamgeist soccer ball. (Courtesy of the University of Sheffield and Ansys UK.)

1.4 ECOLOGICAL MODELS

Computational biology is a growing field of application for numerical methods. In this section, we explore a simplified example from ecology.

Suppose we wish to study the population of a certain species of bird. These birds are born in the spring and live at most 3 years. We will keep track of the population just before breeding, when there will be three classes of birds, based on age: Age 0 (born the previous spring), Age 1, and Age 2. Let $v_0^{(n)}$, $v_1^{(n)}$ and $v_2^{(n)}$ represent the number of females in each age class in Year n. To model population changes we need to know:

- Survival rates. Suppose 20% of Age 0 birds survive to the next spring, and 50% of Age 1 birds survive to become Age 2.
- Fecundity rates. Suppose females that are 1 year old produce a clutch of a certain size, of which 3 females are expected to survive to the next breeding season. Females that are 2 years old lay more eggs and suppose that of these, 6 females are expected to survive to the next spring.

Then we have the following model of the number of females in each age class:

$$\begin{aligned} v_0^{(n+1)} &= 3v_1^{(n)} + 6v_2^{(n)}, \\ v_1^{(n+1)} &= 0.2v_0^{(n)}, \\ v_2^{(n+1)} &= 0.5v_1^{(n)}. \end{aligned}$$

In matrix–vector form,

$$\begin{aligned} \mathbf{v}^{(n+1)} &= A\mathbf{v}^{(n)} \\ &= \begin{pmatrix} 0 & 3 & 6 \\ 0.2 & 0 & 0 \\ 0 & 0.5 & 0 \end{pmatrix} \begin{pmatrix} v_0^{(n)} \\ v_1^{(n)} \\ v_2^{(n)} \end{pmatrix}. \end{aligned} \tag{1.3}$$

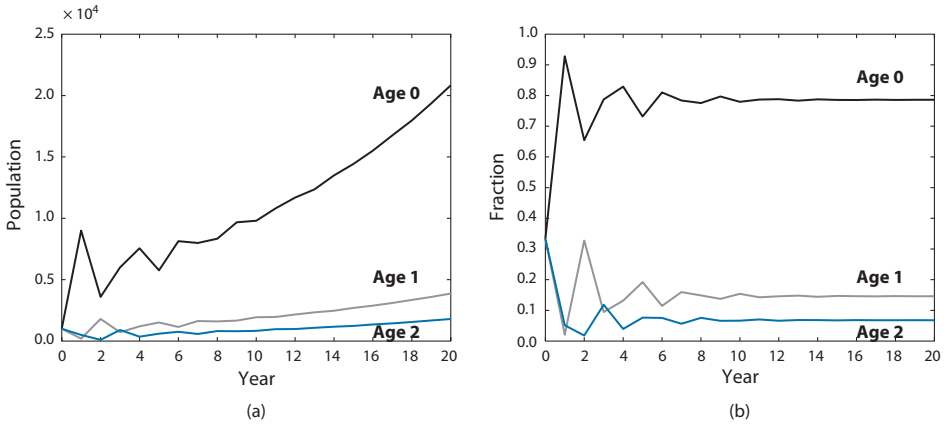


Figure 1.8. Bird population with $a_{32} = 0.5$ and initial data $\mathbf{v}^{(0)} = (1000, 1000, 1000)^T$.

The matrix A in (1.3) can be used to predict future populations from year to year and expected long-term behavior. This type of matrix, reflecting survival rates and fecundities, is called a **Leslie matrix**.

Suppose we start with a population of 3000 females, 1000 of each age. Using (1.3), we find

$$\mathbf{v}^{(0)} = \begin{pmatrix} 1000 \\ 1000 \\ 1000 \end{pmatrix}, \qquad \mathbf{v}^{(1)} = A\mathbf{v}^{(0)} = \begin{pmatrix} 0 & 3 & 6 \\ 0.2 & 0 & 0 \\ 0 & 0.5 & 0 \end{pmatrix} \mathbf{v}^{(0)} = \begin{pmatrix} 9000 \\ 200 \\ 500 \end{pmatrix},$$

$$\mathbf{v}^{(2)} = A\mathbf{v}^{(1)} = \begin{pmatrix} 3600 \\ 1800 \\ 100 \end{pmatrix}, \quad \text{and} \quad \mathbf{v}^{(3)} = A\mathbf{v}^{(2)} = \begin{pmatrix} 6000 \\ 720 \\ 900 \end{pmatrix}.$$

Plotting the population in each age group as a function of the year produces the graph in figure 1.8(a). Clearly the population grows exponentially. Figure 1.8(b) shows the proportion of the population in each age class as a function of the year. Note how the proportion of birds in each age class settles into a steady state.

To be more quantitative, we might look, for instance, at the population vectors $\mathbf{v}^{(20)}$ and $\mathbf{v}^{(19)}$. We find that $\mathbf{v}^{(20)} = \big(20833, 3873, 1797\big)^T$, which indicates that in year 20 the fraction of birds in each age class is $\big(0.7861, 0.1461, 0.068\big)^T$. Looking at the difference between $\mathbf{v}^{(20)}$ and $\mathbf{v}^{(19)}$, we find that the total population grows by a factor of 1.0760 between year 19 and year 20.

Note that $\mathbf{v}^{(n)} = A\mathbf{v}^{(n-1)} = A^2\mathbf{v}^{(n-2)} = \ldots = A^n\mathbf{v}^{(0)}$. In chapter 12, we will see how this observation indicates that the asymptotic behavior of this system (the behavior after a long time period) can be predicted from the dominant eigenvalue (the one of largest absolute value) and associated eigenvector of A. The eigenvalues of A can be computed to be 1.0759, $-0.5380 + 0.5179i$, and $-0.5380 - 0.5179i$ (where $i = \sqrt{-1}$). The largest magnitude of an eigenvalue is 1.0759, and the associated eigenvector, scaled so that its entries sum to 1, is $\hat{\mathbf{v}} = \big(0.7860, 0.1461, 0.0679\big)$. The largest magnitude of an eigenvalue

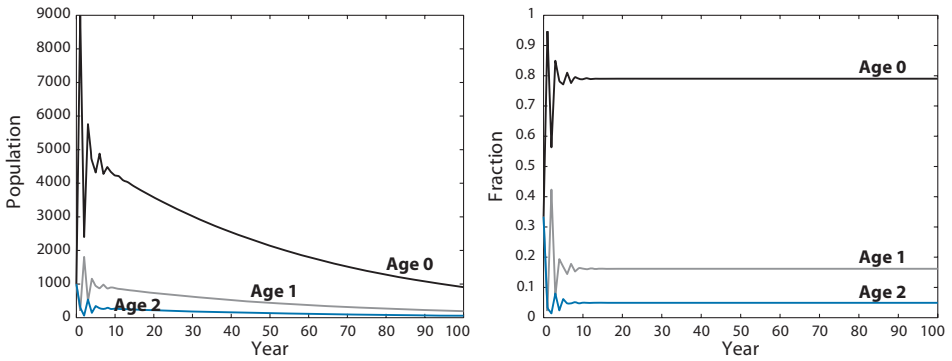


Figure 1.9. Bird population with $a_{32} = 0.3$ and initial data $\mathbf{v}^{(0)} = (1000, 1000, 1000)^T$.

determines the rate of growth of the population after a long period of time, and the entries in $\hat{\mathbf{v}}$ give the fractions of birds in each class, as these fractions approach a steady state. While the eigenvalues and eigenvectors of a 3 by 3 matrix such as A can be determined analytically, we will learn in chapter 12 about methods to numerically approximate eigenvalues and eigenvectors of much larger matrices.

Harvesting Strategies. We can use this model to investigate questions such as the following. Suppose we want to harvest a certain fraction of this population, either to control the exponential growth or to use it as a food source (or both). How much should we harvest in order to control the growth without causing extinction?

Harvesting will decrease the survival rates. We might wish to decrease these rates to a point where the dominant eigenvalue is very close to 1 in absolute value, since this would mean that the total population, after a period of time, would remain fixed. If we decrease the survival rates too much, then all eigenvalues will be less than 1 in magnitude and the population size will decrease exponentially to extinction.

For example, suppose we harvest 20% of the 1-year-olds, so that the survival rate falls from 0.5 to 0.3. In this case, entry a_{32} in matrix A changes from 0.5 to 0.3, and it turns out that the largest magnitude eigenvalue decreases to 0.9830. A simulation with these parameters yields the graphs in figure 1.9.

Note that we now have exponential decay of the population and eventual extinction. The decrease in population is gradual, however, and over the 100-year time span shown here it decreases only from 3000 to 1148. Asymptotically (i.e., after a long period of time), the total population will decrease each year to 0.9830 times that of the previous year. Hence if the population is 1148 after 100 years of this harvesting strategy, the number of additional years before the population drops below 1 (i.e., the species becomes extinct), might be estimated by the value of k that satisfies $1148(0.9830)^k = 1$, which is $k = -\log(1148)/\log(0.9830) \approx 411$ years.

One could try to find the matrix element a_{32} that results in the largest magnitude eigenvalue of A being exactly 1, and while this is an interesting

mathematical problem, from the point of view of population control, it would likely be an exercise in futility. Clearly, this model is highly simplified, and even if the assumptions about survival rates and fecundity rates held initially, they might well change over time. Populations would need to be monitored frequently to adjust a particular harvesting strategy to the changing environment.

1.5 MODELING A WEB SURFER AND GOOGLE

Submitting a query to a search engine is a common method of information retrieval. Companies compete to be listed high in the rankings returned by a search engine. In fact, some companies' business is to help raise the rankings of a paying customer's web page. This is done by exploiting knowledge of the algorithms used by search engines. While a certain amount of information about search engine algorithms is publicly available, there is a certain amount that is proprietary. This section discusses how one can rank web pages based on content and is introductory in nature. An interested reader is encouraged to research the literature on search engine analysis, which is an ever-growing field. Here we will consider a simple vector space model for performing a search. This method does not take into account the hyperlink structure of the World Wide Web, and so the rankings from such a model might be aggregated with the results of an algorithm that does consider the Web's hyperlink structure. Google's PageRank algorithm is such a method and will be looked at briefly in this section and in more detail in sections 3.4 and 12.1.5.

1.5.1 The Vector Space Model

The vector space model consists of two main parts: a list of documents and a dictionary. The list of documents consists of those documents on which searches are to be conducted, and the dictionary of terms is a database of keywords. While the dictionary of terms could be all the terms in every document, this may not be desirable or computationally practical. Words not in the dictionary return empty searches.

With a list of n documents and m keywords, the vector space model constructs an m by n *document matrix* A with

$$a_{ij} = \begin{cases} 1 & \text{if document } j \text{ is relevant to term } i, \\ 0 & \text{otherwise.} \end{cases} \tag{1.4}$$

When a query is issued, a query vector $\mathbf{q} = (q_1, \ldots, q_m)^T$ is then formed, with $q_i = 1$ if the query includes term i and $q_i = 0$ otherwise. The "closeness" of the query to each document is then measured by the cosine of the angle between the query vector $\mathbf{q}$ and the column of A representing that document. If $\mathbf{a}_j$ denotes the jth column of A (and if it is not entirely zero due to document j containing no keywords), then the angle θ_j between $\mathbf{q}$ and $\mathbf{a}_j$ satisfies

$$\cos(\theta_j) = \frac{\mathbf{a}_j^T \mathbf{q}}{\|\mathbf{a}_j\|_2 \|\mathbf{q}\|_2}. \tag{1.5}$$

TABLE 1.1
The dictionary of terms and documents used in a three-dimensional example.

Dictionary		Documents
electric	I	the art of war
fencing	II	the fencing master
foil	III	fencing techniques of foil, epee, and saber
	IV	hot tips on building electric fencing

Since a larger value of the cosine implies a smaller angle between the vectors, documents are ranked in order of relevance to the query by arranging the cosines in descending order.

A major difficulty in forming the document matrix is determining whether a document is relevant to a term. There are a variety of ways of doing this. For instance, every document can be searched for each keyword. A document is deemed relevant to those keywords that are contained within it. This requires considerable computational expense when one faces a large number of documents and keywords. An alternative that is employed by some search engines is to read and perform analysis on only a portion of a document's text. For example, Google and Yahoo! pull only around 100K and 500K bytes, respectively, of web page text [12]. Other choices that must be made include whether to require exact matching of terms or to allow synonyms, and whether or not to count word order. For example, do we treat queries of `boat show` and `show boat` as the same or different? All of these choices impact which documents are deemed most relevant [36].

To illustrate the method, we give a simple example, in which a document is deemed relevant if its title contains the keyword exactly.

Searching in a Tiny Space

Suppose that our dictionary contains three keywords—"electric", "fencing", and "foil"—and that there are four documents entitled "the art of war", "the fencing master", "fencing techniques of foil, epee, and saber", and "hot tips on building electric fencing". We have written the document titles and dictionary terms in lower case to avoid questions about matching in the presence of such a factor. The information is listed in table 1.1 for easy reference.

To form the document matrix A—where rows 1, 2, and 3 correspond to the terms "electric", "fencing", and "foil", respectively, while columns 1, 2, 3 and 4 correspond to documents I, II, III, and IV, respectively—we use (1.4) to obtain

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Figure 1.10 depicts the columns of A (each except the first normalized to have length 1) as vectors in three-space.

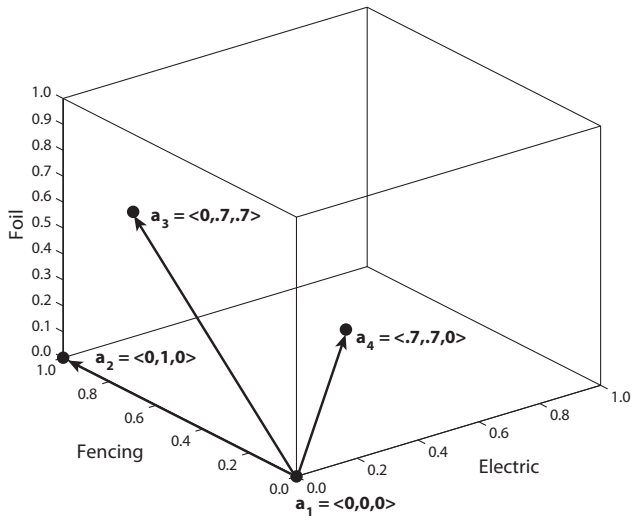


Figure 1.10. Visualization of vector space search in a three-dimensional example.

Suppose our query is `fencing`. Then the query vector is $\mathbf{q} = (0, 1, 0)^T$, and we can see by inspection that it is identical to column 2 of A. Thus, document II would be deemed most relevant to this query. To determine the rankings of the remaining documents, we compute the cosine of the angle between $\mathbf{q}$ and every other nonzero column of A using (1.5) to find

$$\cos(\theta_3) = \frac{\mathbf{a}_3^T\mathbf{q}}{\|\mathbf{a}_3\|_2\|\mathbf{q}\|_2} = \frac{1}{\sqrt{2}},$$

$$\cos(\theta_4) = \frac{\mathbf{a}_4^T\mathbf{q}}{\|\mathbf{a}_4\|_2\|\mathbf{q}\|_2} = \frac{1}{\sqrt{2}}.$$

Thus documents III and IV would be ranked equally, behind document II.

1.5.2 Google's PageRank

Real-world search engines deal with some of the largest mathematical and computer science problems in today's computational world. Interestingly, "Google" is a play on the word "googol", the number 10^{100}, reflecting the company's goal of organizing all information on the World Wide Web.

When you submit a query, such as `numerical analysis`, to Google, how does the search engine distinguish between the web page listed first and the one listed, say, 100th? There are various factors that play into this decision, one of which is the web page's relevance to your query, as discussed previously. Another important component, however, is the "quality" or "importance" of the page; "important" pages, or pages that are pointed to by many other pages, are more likely to be of interest to you. An algorithm called PageRank is used to measure the importance of a web page. This algorithm relies on a model of web-surfing behavior.

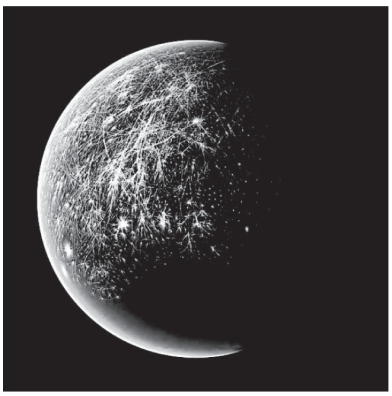


Figure 1.11. A representation of the graph of the Internet created by David F. Gleich, based on an image and data from the OPTE project.

As with any model of reality, Google's model of web-surfing behavior is an approximation. An important feature of PageRank is its assumption about the percentage of time that a surfer follows a link on the current web page. The exact assumptions that are made are proprietary, but it is believed that the PageRank algorithm used by Google assumes that a surfer follows a link on a web page about 85% of the time, with any of the links being equally likely. The other 15% of the time the surfer will enter the URL for another web page, possibly the same one that is currently being visited.

These assumptions about surfing behavior affect not only the accuracy of the model but also the efficiency of numerical techniques used to solve the model. Keep in mind that Google indexes billions of web pages, making this one of the largest computational problems ever solved! In chapter 12 we discuss more about the numerical methods used to tackle this problem.

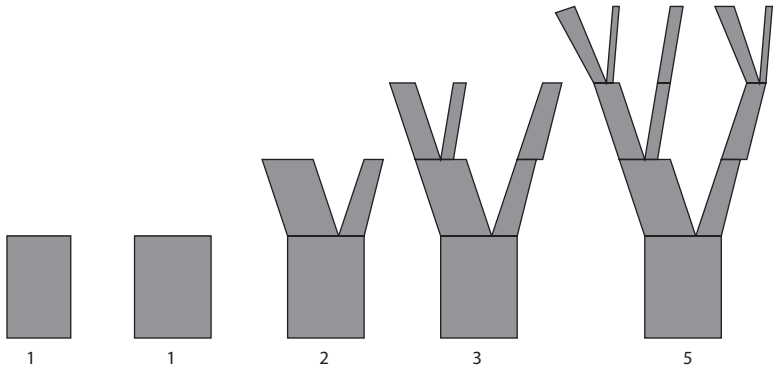
1.6 CHAPTER 1 EXERCISES

1. **The Fibonacci numbers and nature.** The Fibonacci numbers are defined by $F_1 = 1$, $F_2 = 1$, $F_3 = F_2 + F_1 = 2$, $F_4 = F_3 + F_2 = 3$, etc. In general, $F_{j+1} = F_j + F_{j-1}$, $j = 2, 3, \ldots$. Write down F_5, F_6, and F_7.

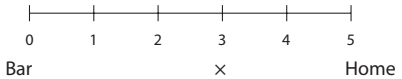
   It is often observed that the number of petals on a flower or the number of branches on a tree is a Fibonacci number. For example, most daisies have either 34, 55, or 89 petals, and these are the 9th, 10th, and 11th Fibonacci numbers. The reason four-leaf clovers are so rare is that 4 is not a Fibonacci number.

   To see the reason for this, consider the following model of branch growth in trees. We start with the main trunk ($F_1 = 1$), which spends one season growing ($F_2 = 1$), and then after another season, develops two branches ($F_3 = 2$)—a major one and a minor one. After the next season, the major branch develops two branches—a major one and a minor one—while the minor branch grows into a major one, ready to divide in the following season. At this point there are $F_4 = 3$ branches—two major ones and one

minor one. At the end of the next season, the two major branches divide, producing two major branches and two minor ones, while the minor one from the previous season grows into a major one. Thus at this point there are $F_5 = 5$ branches, with $F_4 = 3$ of these being major branches ready to divide during the next season. Explain why the Fibonacci numbers arise from this model of branch growth.



2. **Drunkard's walk.** A drunkard starts at position x in the diagram below and with each step moves right one space with probability .5 and left one space with probability .5. If he reaches the bar, he stays there, drinking himself into oblivion. If he reaches home, he goes to bed and stays there. You wish to know the probability $p(x)$ that he reaches home before reaching the bar.

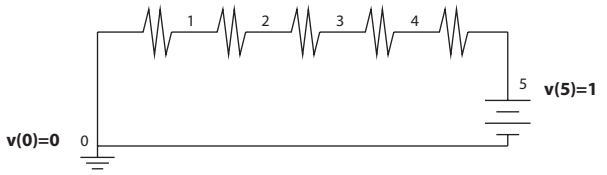


This is a typical Markov chain problem that will be discussed later in the text. Note that $p(0) = 0$ and $p(5) = 1$, since the drunk does not leave either the bar or his home. For $x = 1, 2, 3, 4$, $p(x) = .5\,p(x-1) + .5\,p(x+1)$, since he moves left with probability .5 and right with probability .5.

(a) Let the drunk's starting position be $x = 3$. What are the possible positions that he could be in after one step, and what are the probabilities of each? How about after two steps?
(b) For which initial positions x would you expect him to reach the bar first and for which would you expect him to reach home first, and why?

This is a typical *random walk* problem, and while it is posed as a silly story, it has real physical applications.

Consider an electrical network with equal resistors in series and a unit voltage across the ends.



Voltages $v(x)$ will be established at points $x = 0, 1, 2, 3, 4, 5$. We have grounded the point $x = 0$ so that $v(0) = 0$, and there is no resistor between the source and point $x = 5$, so that $v(5) = 1$. By Kirchoff's laws, the current flowing into x must be equal to the current flowing out. By Ohm's law, if points x and y are separated by a resistor of strength R, then the current i_{xy} that flows from x to y is

$$i_{xy} = \frac{v(x) - v(y)}{R}.$$

Thus for $x = 1, 2, 3, 4$, we have

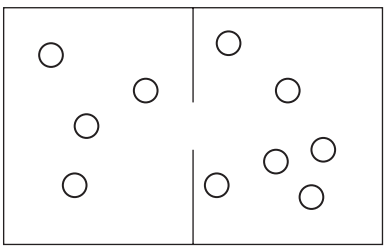
$$\frac{v(x-1) - v(x)}{R} = \frac{v(x) - v(x+1)}{R}.$$

Multiplying by R and combining like terms we see that $v(x) = .5v(x - 1) + .5v(x + 1)$. This is exactly the same formula (with the same boundary conditions) that we found for $p(x)$ in the drunkard's walk problem.

Can you think of other situations that might be modeled in this same way? The behavior of a stock price perhaps? Suppose you generalize by allowing different probabilities of the drunkard moving right or left: say, the probability of moving right is .6 while that of moving left is .4. What generalization of the resistor problem would this correspond to? [Hint: Consider resistors of different strengths.]

3. **Ehrenfests' urn.** Consider two urns, with N balls distributed between them. At each unit of time, you take a ball from one urn and move it to the other, with the probability of choosing each ball being equal, $1/N$. Thus, the probability of choosing a ball from a given urn is proportional to the number of balls in that urn. Let $X(t)$ denote the number of balls in the left urn at time t, and suppose that $X(0) = 0$. Then $X(1) = 1$, since a ball must be drawn from the right urn and moved to the left one.

   (a) Let $N = 100$. What are the possible values for $X(2)$, $X(3)$, and $X(4)$, and what is the probability of each?
   (a) If this process were carried out for a very long time, what do you think would be the most frequently occurring value of $X(t)$?



This model was introduced in the early 1900s by Paul Ehrenfest and Tatiana Ehrenfest-Afanassjewa to describe the diffusion of gas molecules through a permeable membrane. See, for example, http://en.wikipedia.org/wiki/Ehrenfest_model for a discussion of its relation to the second law

of thermodynamics. In chapter 3 we will discuss Monte Carlo simulation of physical processes using models such as this one. Can you think of other situations that might be modeled by such a process? In addition to physical or chemical processes, you might consider, for example, financial decisions or social interactions.

4. Each year undergraduates participate in the *Mathematical Contest in Modeling (MCM)*. See www.mcm.org. Following is an example of the sort of modeling problems that they tackle:

> An ornamental fountain in a large open plaza surrounded by buildings squirts water high into the air. On gusty days, the wind blows spray from the fountain onto passersby. The water-flow from the fountain is controlled by a mechanism linked to an anemometer (which measures wind speed and direction) located on top of an adjacent building. The objective of this control is to provide passersby with an acceptable balance between an attractive spectacle and a soaking: The harder the wind blows, the lower the water volume and height to which the water is squirted, hence the less spray falls outside the pool area.
>
> Your task is to devise an algorithm which uses data provided by the anemometer to adjust the water-flow from the fountain as the wind conditions change.

   Think about how you might create a mathematical model for this problem and compare your ideas with some of the students' solutions that can be found in: *UMAP: Journal of Undergraduate Mathematics and its Applications*. 2002. 23(3):187–271.

5. Consider the following dictionary of keywords:

   chocolate, ice cream, sprinkles,

   and the following list of documents:

   D1. I eat only the chocolate icing off the cake
   D2. I like chocolate and vanilla ice cream
   D3. Children like chocolate cake with sprinkles
   D4. May I have another scoop of ice cream if you hold both the sprinkles and chocolate sauce

   Form the document matrix A, and, using the vector space model, rank the documents D1, D2, D3, and D4 according to their relevance to the query: `chocolate, ice cream`.

6. When you submit a query to a search engine, an ordered list of web pages is returned. The pages are ranked by their relevance to your query and also by the quality of the page. Consider the small network of web pages in figure 1.12. We will assume this is the set of web pages indexed by our search engine. Each vertex in the graph is a web page. A directed link is drawn from web page i to web page j if web page i has a link to web page j. So, we can see, for example, that web page 1 links to web page 4. The PageRank algorithm, as proposed by Larry Page and Sergey Brin [18], assumes that a surfer follows a link on a web page 85% of the time,

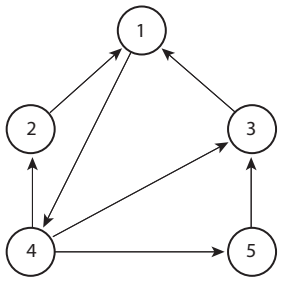


Figure 1.12. A small network of web pages.

with any of the links being equally likely. The other 15% of the time the surfer will enter the URL for another web page, possibly the same one that is currently being visited, again with all pages being equally likely. Let $X_i(t)$ denote the probability of being at web page i after the surfer takes t steps through the network. We will assume the surfer starts at web page 1, so that $X_1(0) = 1$ and $X_i(0) = 0$ for $i = 2, 3, 4, 5$.

(a) Find $X_i(1)$ for $1 \leq i \leq 5$.
(b) Find $X_i(2)$ for $1 \leq i \leq 5$.

As a measure of the quality of a page, PageRank approximates $\lim_{t\to\infty} X_i(t)$ for all i.

2

BASIC OPERATIONS WITH MATLAB

This book is concerned with the understanding of algorithms for problems of continuous mathematics. Part of this understanding is the ability to implement such algorithms. To avoid distracting implementation details, however, we would like to accomplish this implementation in the simplest way possible, even if it is not necessarily the most efficient. One system in which algorithm implementation is especially easy is called MATLAB [94] (short for MATrix LABoratory).

While a helpful academic and instructive tool, MATLAB is used in industry and government, as well. For instance, systems engineers at the NASA Jet Propulsion Laboratory used MATLAB to understand system behavior before launching the Mars Exploration Rover (MER) spacecraft into space.

This chapter contains a short description of basic MATLAB commands, and more commands are described in programs throughout the book. For further information about using MATLAB see, for instance, [52].

2.1 LAUNCHING MATLAB

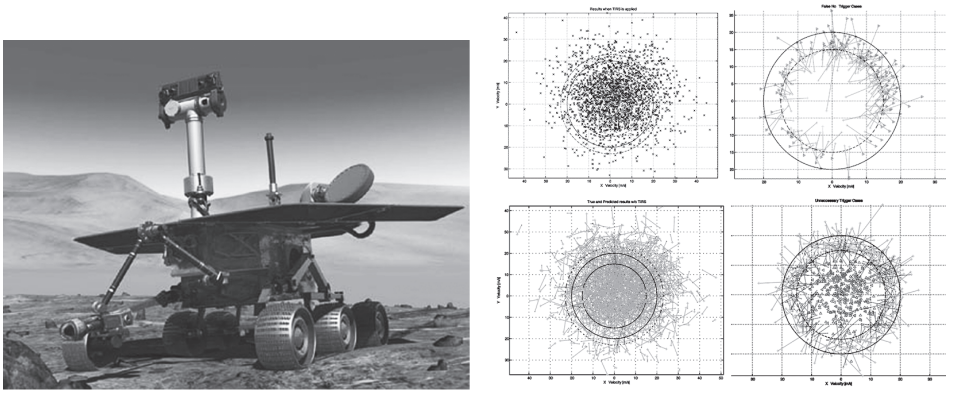
MATLAB is a high-level programming language that is especially well suited to linear algebra computations, but it can be used for almost any numerical problem. Following are some of the basic features of MATLAB that you will need to carry out the programming exercises in this book. Depending on what system you are using, you will start MATLAB either by double clicking on a MATLAB icon or by typing "matlab" or by some similar means. When MATLAB is ready for input from you it will give a prompt such as >>.

You can use MATLAB like a calculator. For instance, if you type at the prompt

```
>> 1+2*3
```

then MATLAB returns with the answer

```
ans =
     7
```



(a) (b)

Figure 2.1. (a) An artist's conception of the Mars rover. (b) Custom statistical MATLAB visualizations that were used to predict how the onboard systems would respond under various atmospheric conditions during descent to the Mars surface. (Images courtesy of NASA/JPL/Cornell University.)

Since you did not give a name to your result, MATLAB stores the result in a variable called `ans`. You can do further arithmetic using the result in `ans`:

```
>> ans/4
```

and MATLAB will return with the result

```
ans =
    1.7500
```

2.2 VECTORS

MATLAB can store row or column vectors. The commands

```
>>  v = [1; 2; 3; 4]
v =
     1
     2
     3
     4

>> w = [5, 6, 7, 8]
w =
     5     6     7     8
```

create a column vector `v` of length 4 and a row vector `w` of length 4. In general, when defining a matrix or vector, semicolons are used to separate rows, while

commas or spaces are used to separate the entries within a row. You can refer to an entry in a vector by giving its index:

```
>> v(2)
ans =
     2

>> w(3)
ans =
     7
```

MATLAB can add two vectors of the same dimension, but it cannot add `v` and `w` because `v` is 4 by 1 and `w` is 1 by 4. If you try to do this, MATLAB will give an error message:

```
>> v+w
??? Error using ==> +
Matrix dimensions must agree.
```

The transpose of `w` is denoted `w'`:

```
>> w'
ans =
     5
     6
     7
     8
```

You can add `v` and `w'` using ordinary vector addition:

```
>> v + w'
ans =
     6
     8
    10
    12
```

Suppose you wish to compute the sum of the entries in `v`. One way to do this is as follows:

```
>> v(1) + v(2) + v(3) + v(4)
ans =
    10
```

Another way is to use a `for` loop:

```
>> sumv = 0;
>> for i=1:4, sumv = sumv + v(i); end;
```

```
>> sumv
sumv =
    10
```

This code initializes the variable `sumv` to 0. It then loops through each value `i` = 1, 2, 3, 4 and replaces the current value of `sumv` with that value plus `v(i)`. The line with the `for` statement actually contains three separate MATLAB commands. It could have been written in the form

```
for i=1:4
  sumv = sumv + v(i);
end
```

MATLAB allows one line to contain multiple commands, provided they are separated by commas or semicolons. Hence in the one-line version of the `for` loop, we had to put a comma (or a semicolon) after the statement `for i=1:4`. This could have been included in the three-line version as well, but it is not necessary. Note also that in the three-line version, we have *indented* the statement(s) inside the `for` loop. This is not necessary, but it is good programming practice. It makes it easy to see which statements are inside and which are outside the `for` loop. Note that the statement `sumv = 0` is followed by a semicolon, as is the statement `sumv = sumv + v(i)` inside the `for` loop. Following a statement by a semicolon suppresses printing of the result. Had we not put the semicolon at the end of the first statement, MATLAB would have printed out the result `sumv = 0`. Had we not put a semicolon after the statement `sumv = sumv + v(i)`, then each time through the `for` loop, MATLAB would have printed out the current value of `sumv`. In a loop of length 4, this might be acceptable; in a loop of length 4 million, it probably would not be! To see the value of `sumv` at the end, we simply type `sumv` without a semicolon and MATLAB prints out its value. Of course, if the answer is not what we were expecting, then we might go back and omit the semicolon after the statement `sumv = sumv + v(i)`, since then we could see the result after each step. Extra output is often useful as a program debugging tool.

2.3 GETTING HELP

Actually, the entries in a vector are most easily summed using a built-in MATLAB function called `sum`. If you are unsure of how to use a MATLAB function or command, you can always type `help` followed by the command name, and MATLAB will provide an explanation of how the command works:

```
>> help sum
 SUM Sum of elements.
     For vectors, SUM(X) is the sum of the elements of X. For
     matrices, SUM(X) is a row vector with the sum over each
```

```
 column. For N-D arrays, SUM(X) operates along the first
 non-singleton dimension.

SUM(X,DIM) sums along the dimension DIM.

Example: If X = [0 1 2
                 3 4 5]

then sum(X,1) is [3 5 7] and sum(X,2) is [ 3
                                           12];

See also PROD, CUMSUM, DIFF.
```

In general, you can type `help` in MATLAB and receive a summary of the classes of commands for which help is available. If you are not sure of the command name for which you are looking, there are two other helpful commands. First, typing `helpdesk` displays the help browser, which is a very *helpful* tool. Additionally, you can type `doc` for the same help browser, or `doc sum` for the hypertext documentation on the MATLAB `sum` command. Second, if you are interested in commands related to summing, you can type `lookfor sum`. With this command, MATLAB searches for the specified keyword "sum" in all help entries. This command results in the following response:

```
>> lookfor sum
TRACE  Sum of diagonal elements.
CUMSUM Cumulative sum of elements.
SUM Sum of elements.
SUMMER Shades of green and yellow colormap.
UIRESUME Resume execution of blocked M-file.
UIWAIT Block execution and wait for resume.
RESUME Resumes paused playback.
RESUME Resumes paused recording.
```

It may take some searching to find precisely the topic and/or command that you are looking for.

2.4 MATRICES

MATLAB also works with matrices:

```
>> A = [1, 2, 3; 4, 5, 6; 7, 8, 0]
A =
     1     2     3
     4     5     6
     7     8     0

>> b = [0; 1; 2]
```

```
b =
     0
     1
     2
```

There are many built-in functions for solving matrix problems. For example, to solve the linear system $Ax = b$, type `A\b`:

```
>> x = A\b
x =
    0.6667
   -0.3333
    0.0000
```

Note that the solution is printed out to only four decimal places. It is actually stored to about sixteen decimal places (see chapter 5). To see more decimal places, you can type

```
>> format long
>> x
x =
    0.66666666666667
   -0.33333333333333
    0.00000000000000
```

Other options include `format short e` and `format long e` to display numbers using scientific notation.

You can check this answer by typing `b - A*x`. The notation `A*x` denotes standard matrix–vector multiplication, and standard vector subtraction is used when you subtract the result from `b`. This should give a vector of 0s, if `x` solves the system exactly. Since the machine carries only about 16 decimal digits, we do not expect it to be exactly zero, but, as we will see later, it should be just a moderate size multiple of 10^{-16}:

```
>> format short
>> b - A*x
ans =
   1.0e-15 *

   -0.0740
   -0.2220
         0
```

This is a good result!

2.5 CREATING AND RUNNING .M FILES

Typing MATLAB commands at the keyboard is fine if you are doing a computation once and will never need to make modifications and run it again.

Once you `exit` MATLAB, however, all of the commands that you typed may be lost. To save the MATLAB commands that you type so that they can be executed again, you must enter them into a file called *filename*`.m`. Then, in MATLAB, if you type *filename*, it will run the commands from that file. We will refer to such files as M-files. The M-file can be produced using any text editor, such as the one that comes up as part of the MATLAB window. Once you save this file, it will be available for future use. Before attempting to execute an M-file from MATLAB, you must remember to change the working directory of MATLAB to the directory in which that file resides.

2.6 COMMENTS

Adding documentation to your MATLAB code allows you and others to maintain your code for future use. Many a programmer has coded what appears to be a crystal clear implementation of an algorithm and later returned to be lost in the listing of commands. Comments can help to alleviate this problem. Adding comments to MATLAB code is easy. Simply adding a `%` makes the remaining portion of that line a comment. In this way, you can make an entire line into a comment:

```
% Solve Ax=b
```

or you can append a comment after a MATLAB command, as in

```
x = A\b;    % This solves Ax=b and stores the result in x.
```

The text following the `%` is simply a comment for the programmer and is ignored by MATLAB.

2.7 PLOTTING

Tables and figures are usually more helpful than long strings of numerical results. Suppose you are interested in viewing a plot of $\cos(50x)$ for $0 \leq x \leq 1$. You can create two vectors, one consisting of x values and the other consisting of the corresponding $y = \cos(50x)$ values, and use the MATLAB `plot` command. To plot the values of $\cos(50x)$ at $x = 0, 0.1, 0.2, \ldots, 1$, type

```
>> x = 0:0.1:1;   % Form the (row) vector of x values.
>> y = cos(50*x); % Evaluate cos(50*x) at each of the x values.
>> plot(x,y)      % Plot the result.
```

Note that the statement `x = 0:0.1:1;` behaves just like the `for` loop

```
>> for i=1:11, x(i) = 0.1*(i-1); end;
```

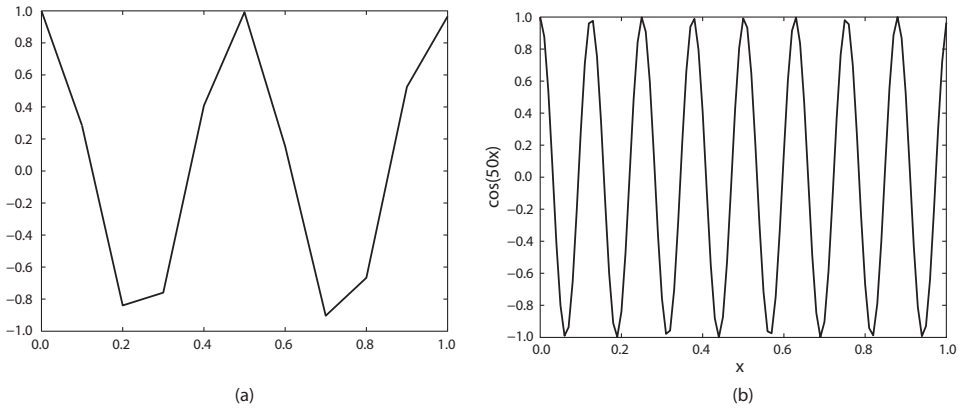


Figure 2.2. Basic MATLAB plots of cosine.

Note also that, unless otherwise specified, each of these statements produces a *row* vector. In order to produce a column vector, one could replace `x(i)` in the above `for` loop by `x(i,1)`, or one could replace the statement in the original code by `x = [0:0.1:1]';`. Clearly, this small number of evaluation points will not produce very high resolution, as is seen in figure 2.2(a).

In the next piece of code we change `x` to include more points and we also include a title and labels for the axes.

```
>> x = 0:0.01:1;      % Create a vector of 101 x values.
>> plot(x,cos(50*x)) % Plot x versus cos(50*x).
>> title('Plot of x versus cos(50x)')
>> ylabel('cos(50x)')
>> xlabel('x')
```

Figure 2.2(b) contains the plot resulting from these commands.

It is also possible to plot more than one function on the same graph. To plot the two functions $f(x) = \cos(50x)$ and $g(x) = x$ on the same graph, type

```
>> plot(x,cos(50*x),x,x)
```

The result is shown in figure 2.3. Note the small boxed legend on the plot; this was added with the command `legend('cos(50x)','x')`.

Another way to plot two functions on the same graph is to first plot one, then type `hold on`, and then plot the other. If you do not type `hold on`, then the second plot will replace the first, but this command tells MATLAB to keep plots on the screen after they are created. To go back to the default of removing old plots before new ones are added, type `hold off`. Again, type `help plot` for more information on plotting or type `doc plot` which posts the Helpdesk documentation for the `plot` command. Other useful commands are `axis` and `plot3`. For a bit of fun, type the commands

```
>> x = 0:0.001:10;
>> comet(x,cos(3*x))
```

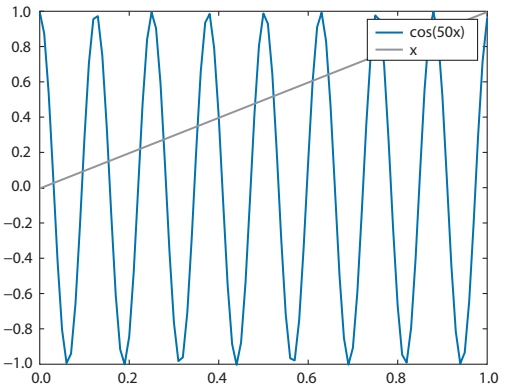


Figure 2.3. A basic MATLAB plot of cosine and a line along with a legend.

For more information on options available for plots, type the commands `hndlgraf`, `hndlaxis` and `ardemo`.

2.8 CREATING YOUR OWN FUNCTIONS

You can create your own functions to use in MATLAB. If your function is simple (e.g., $f(x) = x^2+2x$), then you may enter it using the command `inline`:

```
>> f = inline('x.^2 + 2*x')
f =
     Inline function:
     f(x) = x.^2 + 2*x
```

Note the `.^2` notation. The expression `x^2` produces the square of `x` if `x` is a scalar, but it gives an error message if `x` is a vector, since standard vector multiplication is defined only if the inner dimensions of the vectors are the same (i.e., the first vector is 1 by n and the second is n by 1 or the first is n by 1 and the second is 1 by m). The operation `.^` applied to a vector, however, squares each entry individually. Since we may wish to evaluate the function at each entry in a vector of `x` values, we must use the `.^` operation. To evaluate `f` at the integers between 0 and 5, type

```
>> f([0:5])
ans =
     0     3     8    15    24    35
```

Similarly, you can create an anonymous function by typing

```
>> f = @(x)(x.^2 + 2*x)
f =
    @(x)(x.^2+2*x)
```

If the function is more complicated, you may create a file whose name ends in `.m` which tells MATLAB how to compute the function. Type `help function` to see the format of the function file. Our function here could be computed using the following file (called `f.m`):

```
function output = f(x)
output = x.^2 + 2*x;
```

This function is called from MATLAB in the same way as above; that is, `f(x)`, where `x` can be a scalar or vector.

2.9 PRINTING

While graphical output can be an important visualization tool, numerical results are often presented in tables. There are several ways to print output to the screen in MATLAB. First, to simply display a variable's contents, use the command `display`.

```
>> x = 0:.5:2;
>> display(x)
x =
         0    0.5000    1.0000    1.5000    2.0000
```

In many cases, the extra carriage return imposed by the `display` command clutters printed results. Therefore, another helpful command is `disp`, which is similar to the `display` command.

```
>> disp(x)
         0    0.5000    1.0000    1.5000    2.0000
```

You may still wish to have the variable name printed. Concatenating an array of text for output accomplishes this purpose.

```
>> disp(['x = ',num2str(x)])
x = 0         0.5           1         1.5           2
```

For more information, type `help num2str`.

Tables such as the following can be created using `disp`.

```
>> disp('    Score 1   Score 2   Score 3'), disp(rand(5,3))
    Score 1   Score 2   Score 3
    0.4514    0.3840    0.6085
    0.0439    0.6831    0.0158
    0.0272    0.0928    0.0164
    0.3127    0.0353    0.1901
    0.0129    0.6124    0.5869
```

You may find the `fprintf` command easier to use for tables of results. For instance, consider the simple loop

```
>> fprintf('   x          sqrt(x)\n====================\n')
for i=1:5, fprintf('%f     %f\n',i,sqrt(i)), end
   x          sqrt(x)
====================
1.000000     1.000000
2.000000     1.414214
3.000000     1.732051
4.000000     2.000000
5.000000     2.236068
```

The `fprintf` command takes format specifiers and variables to be printed in those formats. The `%f` format indicates that a number will be printed in fixed point format in that location of the line, and the `\n` forces a carriage return after the two quantities `i` and `sqrt(i)` are printed. You can specify the total field width and the number of places to be printed after the decimal point by replacing `%f` by, say, `%8.4f` to indicate that the entire number is to be printed in 8 spaces, with 4 places printed after the decimal point. You can send your output to a file instead of the screen by typing

```
fid = fopen('sqrt.txt','w');
fprintf(fid,'   x      sqrt(x)\n================\n');
for i=1:5, fprintf(fid,'%4.0f     %8.4f\n',i,sqrt(i)); end
```

which prints the following table in a file called `sqrt.txt`.

```
   x      sqrt(x)
================
   1       1.0000
   2       1.4142
   3       1.7321
   4       2.0000
   5       2.2361
```

Again, for more information, refer to MATLAB documentation.

2.10 MORE LOOPS AND CONDITIONALS

We have already seen how `for` loops can be used in MATLAB to execute a set of commands a given number of times. Suppose, instead, that one wishes to execute the commands until some condition is satisfied. For example, one might approximate a root of a given function `f(x)` by first plotting the function on a coarse scale where one can see the approximate root, then plotting appropriate sections on finer and finer scales until one can identify the root to

the precision needed. This can be accomplished with the following MATLAB code.

```
xmin = input(' Enter initial xmin: ');
xmax = input(' Enter initial xmax: ');
tol = input(' Enter tolerance: ');
while xmax-xmin > tol,
  x = [xmin:(xmax-xmin)/100:xmax];
  y = f(x);
  plot(x,y)
  xmin = input(' Enter new value for xmin: ');
  xmax = input(' Enter new value for xmax: ');
end;
```

The user looks at each plot to determine a value `xmin` that is just left of the root and a value `xmax` that is just right of the root. The next plot then contains only this section, so that closer values `xmin` and `xmax` can be determined. In chapter 4 we discuss more efficient ways of finding a root of `f(x)`.

Another important statement is the conditional `if` statement. In the above code segment, one might wish to let the user know if the code happens to find a point at which the absolute value of `f` is less than some other tolerance, say, `delta`. This could be accomplished by inserting the following lines after the statement `y = f(x);`:

```
[ymin,index] = min(abs(y));
% This finds the minimum absolute value of y and its index.
if ymin < delta,
 fprintf(' f( %f ) = %f\n', x(index), y(index))
 % This prints the x and y values at this index.
end;
```

The `if` statement may also contain an `else` clause. For example, to additionally write a message when `ymin` is greater than or equal to `delta`, one could modify the above `if` statement to say:

```
if ymin < delta,
  fprintf(' f( %f ) = %f\n', x(index), y(index))
  % This prints the x and y values at this index.
else
  fprintf(' No points found where |f(x)| < %f\n', delta)
end;
```

2.11 CLEARING VARIABLES

You may clear a particular variable by typing

```
>> clear x
```

or all variables with

```
>> clear all
```

This is important when you want to be sure that all variable names have been erased from memory.

2.12 LOGGING YOUR SESSION

You can keep a record of your MATLAB session by typing

```
>> diary('hw1.txt')
... some other commands ...
>> diary off
```

This command records all subsequent commands that you type and all responses that MATLAB returns in a file named `hw1.txt`. You will want to name the file by replacing `hw1.txt` with a more descriptive name related to your work. Note, however, that you *cannot* then run the file from MATLAB; this is simply a device for recording what happened during your keyboard session.

Note also that if you execute an M-file in a session logged with the `diary` command, you may want to type `echo on` before executing the M-file. In this way, the commands in the M-file are echoed along with MATLAB's response. Otherwise, the diary file will contain only the responses, not the commands.

2.13 MORE ADVANCED COMMANDS

It is perhaps apparent from the reference at the beginning of this chapter to the work on the Mars Exploration Rover, that MATLAB has a large number of commands. To close this chapter, we demonstrate some of the graphical capabilities through an example.

The commands

```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
meshc(X,Y,Z);
axis([-3 3 -3 3 -10 5])
```

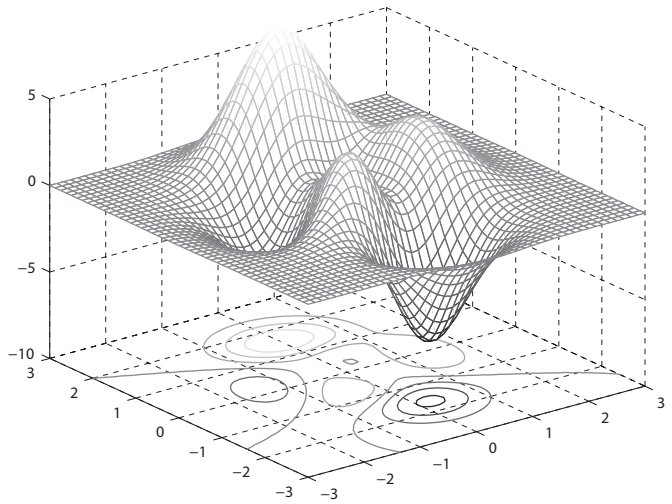


Figure 2.4. A three-dimensional MATLAB plot.

produce the plot in figure 2.4. In order to understand this plot, search MATLAB documentation for the commands `meshgrid`, `peaks`, `meschc`, and `axis`. While this chapter will get you started using MATLAB, effective use of the MATLAB documentation will be the key to proceeding to more complicated programs.

2.14 CHAPTER 2 EXERCISES

1. Run the examples in this chapter using MATLAB to be sure that you see the same results.
2. With the matrices and vectors

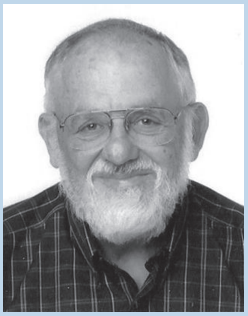
$$A = \begin{pmatrix} 10 & -3 \\ 4 & 2 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 \\ -1 & 2 \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

compute the following *both* by hand and in MATLAB. For the MATLAB computations, use the `diary` command to record your session.

(a) $\mathbf{v}^T\mathbf{w}$
(b) $\mathbf{v}\mathbf{w}^T$
(c) $A\mathbf{v}$
(d) $A^T\mathbf{v}$
(e) AB
(f) BA
(g) A^2 $(= AA)$
(h) the vector $\mathbf{y}$ for which $B\mathbf{y} = \mathbf{w}$
(i) the vector $\mathbf{x}$ for which $A\mathbf{x} = \mathbf{v}$

3. Use MATLAB to produce a single plot displaying the graphs of the functions $\sin(kx)$ across $[0, 2\pi]$, for $k = 1, \ldots, 5$.
4. Use MATLAB to print a table of values x, $\sin x$, and $\cos x$, for $x = 0, \frac{\pi}{6}, \frac{2\pi}{6}, \ldots, 2\pi$. Label the columns of your table.

MATLAB'S ORIGINS



Cleve Moler is chairman and chief scientist at MathWorks. In the mid to late 1970s, he was one of the authors of LINPACK and EISPACK, Fortran libraries for numerical computing. To give easy access to these libraries to his students at the University of New Mexico, Dr. Moler invented MATLAB. In 1984, he cofounded The MathWorks with Jack Little to commercialize the MATLAB program. Cleve Moler received his bachelor's degree from California Institute of Technology, and a Ph.D. from Stanford University. Moler was a professor of math and computer science for almost 20 years at the University of Michigan, Stanford University and the University of New Mexico. Before joining The MathWorks full-time in 1989, he also worked for the Intel Hypercube Company and Ardent Computer. Dr. Moler was elected to the National Academy of Engineering in 1997. He also served as president of the Society for Industrial and Applied Mathematics (SIAM) during 2007–2008 [68]. (Photo courtesy of Cleve Moler.)

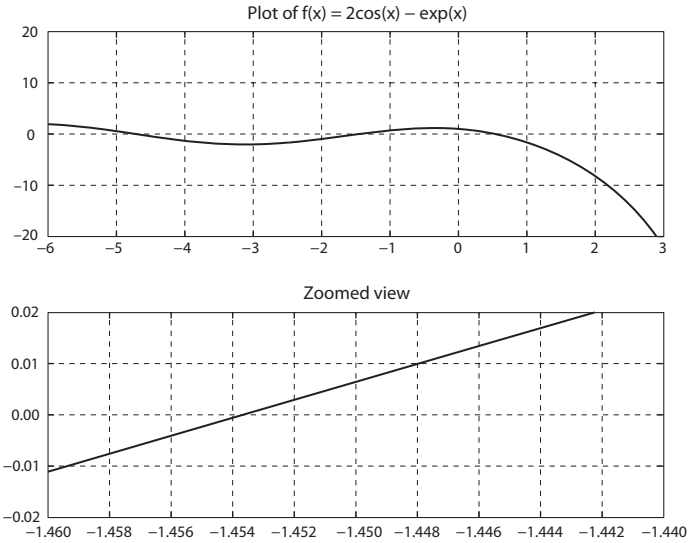
5. Download the file `plotfunction1.m` from the book's web page and execute it. This should produce the two plots on the next page. The top plot shows the function $f(x) = 2\cos(x) - e^x$ for $-6 \leq x \leq 3$, and from this plot it appears that $f(x)$ has three roots in this interval. The bottom plot is a zoomed view near one of these roots, showing that $f(x)$ has a root near $x = -1.454$. Note the different vertical scale as well as the different horizontal scale of this plot. Note also that when we zoom in on this function it looks nearly *linear* over this short interval. This will be important when we study numerical methods for approximating roots.

   (a) Modify this script so that the bottom plot shows a zoomed view near the leftmost root. Write an estimate of the value of this root to at least 3 decimal places. You may find it useful to first use the zoom feature in MATLAB to see approximately where the root is and then to choose your axis command for the second plot appropriately.

   (b) Edit the script from part (a) to plot the function

$$f(x) = \frac{4x\sin x - 3}{2 + x^2}$$

   over the range $0 \leq x \leq 4$ and also plot a zoomed view near the leftmost root. Write an estimate of the value of the root from the plots that is accurate to 3 decimal places. Note that once you have defined the vector `x` properly, you will need to use appropriate componentwise



multiplication and division to evaluate this expression:

```
y = (4*x.*sin(x) - 3) ./ (2 + x.^2);
```

6. Plot each of the functions below over the range specified. Produce four plots on the same page using the `subplot` command.

   (a) $f(x) = |x - 1|$ for $-3 \le x \le 3$. (Use `abs` in MATLAB.)
   (b) $f(x) = \sqrt{|x|}$ for $-4 \le x \le 4$. (Use `sqrt` in MATLAB.)
   (c) $f(x) = e^{-x^2} = \exp(-x^2)$ for $-4 \le x \le 4$. (Use `exp` in MATLAB.)
   (d) $f(x) = \dfrac{1}{10x^2 + 1}$ for $-2 \le x \le 2$.

7. Use MATLAB to plot the circles

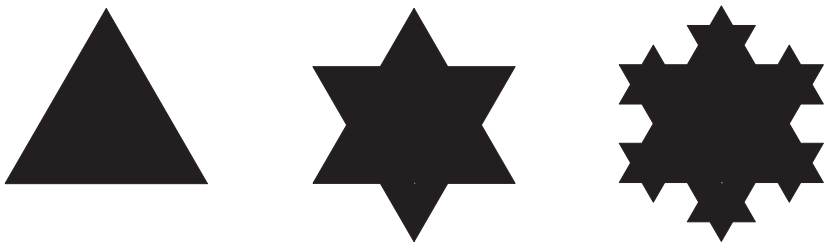
$$(x - 2)^2 + (y - 1)^2 = 2,$$
$$(x - 2.5)^2 + y^2 = 3.5$$

   and zoom in on the plot to determine approximately where the circles intersect.
   [Hint: One way to plot the first circle is:

```
theta = linspace(0, 2*pi, 1000);
r = sqrt(2);
x = 2 + r*cos(theta);
y = 1 + r*sin(theta);
plot(x,y)
axis equal     %  so the circles look circular!
```

   Use the command `hold on` after this to keep this circle on the screen while you plot the second circle in a similar manner.]

8. In this exercise, you will plot the initial stages of a process that creates a fractal known as *Koch's snowflake*, which is depicted below.



Stage 0 Stage 1 Stage 2

This exercise uses the MATLAB M-file `koch.m`, which you will find on the web page. The M-file contains all the necessary commands to create the fractal, except for the necessary plotting commands. Edit this M-file so that each stage of the fractal is plotted. [Hint: This can be accomplished by adding a plot command just before the completion of the outer `for` loop.] Add the following commands to keep consistency between plots in the animation.

```
axis([-0.75 0.75 -sqrt(3)/6 1]);
axis equal
```

Note that the `cla` command clears the axes. Finally, add the command `pause(0.5)` in appropriate places to slow the animation. (The `fill` command, as opposed to `plot`, produced the filled fractals depicted above.) We will create fractals using Newton's method in chapter 4.

9. A magic square is an arrangement of the numbers from 1 to n^2 in an n by n matrix, where each number occurs exactly once, and the sum of the entries in any row, any column, or any main diagonal is the same. The MATLAB command `magic(n)` creates an n by n (where $n > 2$) magic square. Create a 5 by 5 magic square and verify using the `sum` command in MATLAB that the sums of the columns, rows and diagonals are equal. Create a log of your session that records your work. [Hint: To find the sums of the diagonals, read the documentation for the `diag` and the `flipud` commands.]
10. More advanced plotting commands can be useful in MATLAB programming.

    (a) In the MATLAB command window type

    ```
    [X,Y,Z] = peaks(30);
    surf(X,Y,Z);
    ```

    (b) Give this plot the title "3-D shaded surface plot".
    (c) Type `colormap hot` and observe the change in the plot.
    (d) Print the resulting plot with the given title.

11. Computer graphics make extensive use of matrix operations. For example, rotating an object is a simple matrix–vector operation. In two dimensions, a curve can be rotated counterclockwise through an angle θ about the

origin by multiplying every point that lies on the curve by the rotation matrix

$$R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

As an example, let us rotate the rectangle with vertex coordinates $[1, 0]$, $[0, 1]$, $[-1, 0]$, and $[0, -1]$ through an angle $\theta = \pi/4$. In MATLAB, type the following code to generate the original and rotated squares plotted one on top of the other.

```
% create matrix whose columns contain the coordinates of
% each vertex.
U = [1, 0, -1, 0; 0, 1, 0, -1];

theta = pi/4;

% Create a red unit square
% Note U(1,:) denotes the first row of U
fill(U(1,:),U(2,:),'r')

% Retain current plot and axis properties so that
% subsequent graphing commands add to the existing graph
hold on

% Set the axis
axis([-2 2 -2 2]);

% Perform rotation.
R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
V = R*U;

fill(V(1,:), V(2,:),'b');

axis equal tight, grid on
```

Note that the `fill` command in MATLAB plots a filled polygon determined by two vectors containing the x- and y-coordinates of the vertices.

(a) Adapt this code to plot a triangle with vertices $(5, 0), (6, 2)$ and $(4, 1)$. Plot the triangles resulting from rotating the original triangle by $\pi/2, \pi$, and $3\pi/2$ radians about the origin. Plot all four triangles on the same set of axes.

(b) Rotating by θ radians and then rotating by $-\theta$ radians leaves the figure unchanged. This corresponds to multiplying first by $R(\theta)$ and then by $R(-\theta)$. Using MATLAB, verify that these matrices are inverses of each other (i.e., their product is the identity) for $\theta = \pi/3$ and for $\theta = \pi/4$.

(c) Using the trigonometric identities $\cos\theta = \cos(-\theta)$ and $-\sin\theta = \sin(-\theta)$, prove that $R(\theta)$ and $R(-\theta)$ are inverses of each other for any θ.

(d) Let R be the matrix that rotates counterclockwise through the angle $\pi/8$, and let $\hat{R} = 0.9*R$. Then the matrix $\hat{R}$ simultaneously rotates and shrinks an object. Edit the code above to repeatedly rotate and shrink (by the same amounts on each step) the square (again originally with coordinates $[1, 0], [0, 1], [-1, 0],$ and $[0, -1]$) for 50 iterations. The plot should show all of the 51 squares on the same set of axes.

(e) Apply $\hat{R}$ but now after each rotation translate the square by 1 unit in the x direction and 2 units in the y direction. Note, you must apply the rotation and translation to all the vertices of the square. The resulting squares should visually suggest the existence of a fixed point. Such a point satisfies the equation

$$\begin{pmatrix} x \\ y \end{pmatrix} = \hat{R} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

Solve this equation to find the numerical value of this fixed point.

12. The previous exercise used rotation matrices in two dimensions. Now we explore the speed of matrix operations in a computer graphics model in three dimensions. In figure 2.5, we see a model of Yoda. The tessellation contains 33,862 vertices. Let V be a matrix with 3 columns and 33,862 rows, where row i contains the x-, y-, and z-coordinates of the ith vertex in the model. The image can be translated by t units in the y direction by using a translation matrix T where

$$T = \begin{pmatrix} 0 & t & 0 \\ 0 & t & 0 \\ \vdots & \vdots & \vdots \\ 0 & t & 0 \end{pmatrix}.$$

If $V_t = V + T$, then V_t contains the vertex information for the model after a translation of t units in the y direction.

Download the files `yoda.m` and `yodapose_low.mat` from the web page. Run the file `yoda.m` in MATLAB. You will see an animation of the model being translated in space using matrix addition.

(a) The image can be rotated by θ radians about the y-axis by multiplying V on the right by R_y where

$$R_y = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}.$$

Edit the code to continuously rotate the image by $\pi/24$ radians until the image has made one full rotation about the y-axis.

(b) How many multiplications are performed when you use matrix multiplication (with R_y) to rotate the image once by $\pi/24$ radians? (Remember that V is a 33,862 by 3 matrix.) Keep this in mind as you watch how fast MATLAB performs the calculations and displays the results.

Figure 2.5. A model of Yoda created with 33,862 vertices. (Model created by Kecskemeti B. Zoltan. Courtesy of Lucasfilm Ltd. *Star Wars: Episode II - Attack of the Clones* ™ & © 2002 Lucasfilm Ltd. All rights reserved. Used under authorization. Unauthorized duplication is a violation of applicable law. Digital Work by Industrial Light & Magic.)

13. You can see a picture of a mandrill by typing the MATLAB commands

```
load mandrill, image(X), colormap(map), axis off equal
```

Each row of the 220 by 3 matrix `map` corresponds to a color with the first, second, and third elements specifying the intensity of red, green, and blue, respectively.

(a) Write down a 3 by 3 matrix T which, when applied to `map` on the right, will reverse the order of the columns. Use the MATLAB commands `map2 = map*T; colormap(map2)` to see the effect. Explain what happened and why.

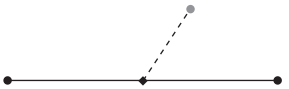
(b) Write down a 3 by 3 matrix S which, when applied to `map` on the right, leaves columns one and two unchanged but replaces column three by a column of 0s. Use the MATLAB commands `map3=map*S; colormap(map3)` to see the effect of this change. Explain what happened and why. You may need to type `help colormap` to find out exactly what this new color map does.

14. In this exercise, we will create a fractal coastline. Fractals have many uses, and here we see how they can be used to create qualitatively realistic-looking pictures.

We will use the following iterative algorithm to create two-dimensional fractal landscapes.

0. Begin with one straight line segment.
1. For each line segment in the current figure, find the midpoint, denoted by a solid diamond in the picture below.

2. Create a new point by moving a random amount in the x and y directions from that midpoint as seen below. The size of the random displacement will be adjusted at each iteration.



3. Connect the endpoints of the original line with the new point.

4. If the picture looks good then stop, else adjust the random displacement size and go to step 1.

You will need to determine a suitable range for the random displacement at each iteration to obtain a realistic-looking picture. One such choice resulted in the figures below.

Iterate 0

Iterate 1

Iterate 2

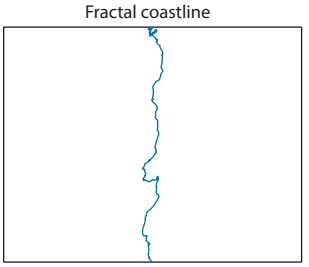
Iterate 8

For the exercise, you need not color the land and water masses (although you may do so by using the `fill` command), but simply generate a realistic-looking coastline. If your implementation stores the x values and y values for points on the fractal coastline in the vectors `xValues` and

`yValues`, respectively, then the MATLAB commands:

```
plot(xValues,yValues)
axis equal
```

will plot the fractal coastline with a 1:1 aspect ratio for the axes.



(a) Write a program to create fractal coastlines using the algorithm above.
(b) Describe how your implementation adjusts the range of the random numbers used for displacements in step 2 at each iteration.
(c) Create at least two fractal coastlines with your code.

15. Find documentation on the `movie` command by typing `helpdesk` or `doc movie`. At the bottom of the documentation on this command, you will find the code:

```
Z = peaks;
surf(Z);
axis tight
set(gca,'nextplot','replacechildren');
for j = 1:20
    surf(sin(2*pi*j/20)*Z,Z)
    F(j) = getframe;
end movie(F,5)
```

Cut and paste this code into the MATLAB command window and describe the results. Proficiency in MATLAB programming can increase dramatically through effective use of the available documentation.