

Chapter 1

Introduction

©2002 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
14 January 2002

The importance of computers in physics and the nature of computer simulation is discussed. The nature of object-oriented programming and various computer languages is also considered.

1.1 Importance of Computers in Physics

Computers are becoming increasingly important in how our society is organized. Like any new technology they are affecting how we learn and how we think. Physicists are in the vanguard of the development of new computer hardware and software, and computation has become an integral part of contemporary science. For the purposes of discussion, we divide the use of computers in physics into the following categories:

- numerical analysis
- symbolic manipulation
- simulation
- collection and analysis of data
- visualization

In the *numerical analysis* mode, the simplifying physical principles are discovered prior to the computation. For example, we know that the solution of many problems in physics can be reduced to the solution of a set of simultaneous linear equations. Consider the equations

$$\begin{aligned} 2x + 3y &= 18 \\ x - y &= 4. \end{aligned}$$

It is easy to find the analytical solution $x = 6$, $y = 2$ using the method of substitution and pencil and paper. Suppose we have to solve a set of four simultaneous equations. We again can find

an analytical solution, perhaps using a more sophisticated method. However, if the number of simultaneous equations becomes much larger, we would have to use numerical methods and a computer to find a numerical solution. In this mode, the computer is a tool of numerical analysis, and the essential physical principles, for example, the reduction of the problem to the inversion of a matrix, are included in the computer program. Because it is often necessary to compute a multidimensional integral, manipulate large matrices, or solve a complex differential equation, this use of the computer is important in physics.

An increasingly important mode of computation is *symbolic manipulation*. As an example, suppose we want to know the solution to the quadratic equation, $ax^2 + bx + c = 0$. A symbolic manipulation program can give us the solution as $x = [-b \pm \sqrt{b^2 - 4ac}]/2a$. In addition, such a program can give us the usual numerical solutions for specific values of a , b , and c . Mathematical operations such as differentiation, integration, matrix inversion, and power series expansion can be performed using most symbolic manipulation programs. Maple, Matlab, and Mathematica are examples of symbolic languages.

In the *simulation* mode, the essential elements of a model are included with a minimum of analysis. As an example, suppose a teacher gives \$10 to each student in a class of 100. The teacher, who also begins with \$10 in her pocket, chooses a student at random and flips a coin. If the coin is “tails,” the teacher gives \$0.50 to the student; otherwise, the student gives \$0.50 to the teacher. If either the teacher or the student would go into debt by this transaction, the transaction is not allowed. After many exchanges, what is the probability that a student has s dollars and the probability that the teacher has t dollars? Are these two probabilities the same? One way to find the answers to these questions is to do a classroom experiment. However, such an experiment would be difficult to arrange, and it would be tedious to do a sufficient number of transactions. Although these particular questions can be answered exactly by analytical methods, many problems of this nature cannot be solved in this way.

Another way to proceed is to convert the rules of the model into an algorithm and a computer program, and simulate many exchanges, and compute the probabilities. After we compute the probabilities of interest, we might gain a better understanding of their relation to the exchanges of money, and more insight into the nature of an analytical solution if one exists. We also can modify the rules and ask “what if?” questions. For example, how would the probabilities change if the exchanges were \$1.00 rather than \$0.50? What would happen if the teacher were allowed to go into debt?

In all these approaches, the main goal of the computation is generally insight rather than simply numbers. Computation has had a profound effect on the way we do physics, on the nature of the important questions in physics, and on the physical systems we choose to study. All these approaches require the use of at least some simplifying approximations to make the problem computationally feasible. However, because the simulation mode requires a minimum of analysis and emphasizes an exploratory mode of learning, we stress this approach in this text. For example, if we change the names of the players in the above example, for example, let money → energy, then the above questions would be applicable to problems in magnetism and particle physics.

Computers often are involved in all phases of a laboratory experiment, from the design of the apparatus to the collection and analysis of data. This involvement of the computer has made possible experiments that would otherwise be impossible. Some of these tasks are similar to those encountered in theoretical computation such as the analysis of data. However, the tasks involved in

real-time control and interactive data analysis are qualitatively different and involve the interfacing of computer hardware to various types of instrumentation. For these reasons we will not discuss the application of computers in experimental physics.

1.2 The Nature of Computer Simulation

Why is computation becoming so important in physics? One reason is that most of our analytical tools such as differential calculus are best suited to the analysis of *linear* problems. For example, you probably have learned to analyze the motion of a particle attached to a spring by assuming a linear restoring force and solving Newton's second law of motion. In this case, a small change in the displacement of the particle leads to a small change in the force. However, many natural phenomena are *nonlinear*, and a small change in a variable might produce a large change in another. Because relatively few nonlinear problems can be solved by analytical methods, the computer gives us a new tool to explore nonlinear phenomena. Another reason for the importance of computation is the growing interest in complex systems with many variables or with many degrees of freedom. The money exchange model described in Section 1.1 is a simple example of a system with many variables.

Computer simulations are sometimes referred to as *computer experiments* because they share much in common with laboratory experiments. Some of the analogies are shown in Table 1.1. The starting point of a computer simulation is the development of an idealized model of a physical system of interest. We then need to specify a procedure or *algorithm* for implementing the model on a computer. The computer program simulates the physical system and defines the computer experiment. Such a computer experiment serves as a bridge between laboratory experiments and theoretical calculations. For example, we can obtain essentially exact results by simulating an idealized model that has no laboratory counterpart. The comparison of the simulation results with an approximate theoretical calculation serves as a stimulus to the development of methods of calculation. On the other hand, a simulation can be done on a realistic model in order to make a more direct comparison with laboratory experiments.

Computer simulations, like laboratory experiments, are not substitutes for thinking, but are tools that we can use to understand complex phenomena. But the goal of all our investigations of fundamental phenomena is to seek explanations of physical phenomena that fit on the back of an envelope or that can be made by the wave of a hand.

Developments in computer technology are leading to new ways of thinking about physical systems. Asking the question "How can I formulate the problem on a computer?" has led to new formulations of physical laws and to the realization that it is both practical and natural to express scientific laws as rules for a computer rather than only in terms of differential equations. This new way of thinking about physical processes is leading some physicists to consider the computer as a physical system and to develop novel computer architectures that can more efficiently model physical systems in nature.

Laboratory experiment	Computer simulation
sample	model
physical apparatus	computer program
calibration	testing of program
measurement	computation
data analysis	data analysis

Table 1.1: Analogies between a computer simulation and a laboratory experiment.

1.3 Importance of Graphics and Visualization

Because computers are changing the way we do physics, they will change the way we learn physics. For example, as the computer plays an increasing role in our understanding of physical phenomena, the visual representation of complex numerical results will become even more important. The human eye in conjunction with the visual processing capacity of the brain is a very sophisticated device for analyzing visual information. Most of us can draw a good approximation to the best straight line through a sequence of data points very quickly. Such a straight line is more meaningful to us than the “best fit” line drawn by a statistical package that we do not understand. Our eye can determine patterns and trends that might not be evident from tables of data and can observe changes with time that can lead to insight into the important mechanisms underlying a system’s behavior.

At the same time, the use of graphics can increase our understanding of the nature of analytical solutions. For example, what does a sine function mean to you? We suspect that your answer is not the series, $\sin x = x - x^3/3! + x^5/5! + \dots$, but rather a periodic, constant amplitude curve (see Figure 1.1). What is most important is the mental image gained from a visualization of the form of the function.

An increasing application of computers is the visualization of large amounts of data. Traditional modes of presenting data include two- and three-dimensional plots including contour and field line plots. Frequently, more than three variables are needed to understand the behavior of a system, and new methods of using color and texture are being developed to help researchers gain greater insight about their data.

1.4 Programming Languages

There is no single best programming language any more than there is a best natural language. Fortran is the oldest of the more popular languages and was developed by John Backus and his colleagues at IBM between 1954 and 1957. Fortran is still the most common language used in scientific applications, and Fortran 90/95 has many features that are similar to C. The Basic programming language was developed in 1965 by John Kemeny and Thomas Kurtz of Dartmouth College as a language for introductory courses in computer science. In 1983 Kemeny and Kurtz created True Basic to extend and standardize the language and included structured programming and platform independent graphics. The programs in the first two editions of our textbook were written in True Basic. Popular versions of Basic include Visual Basic and REALBasic.

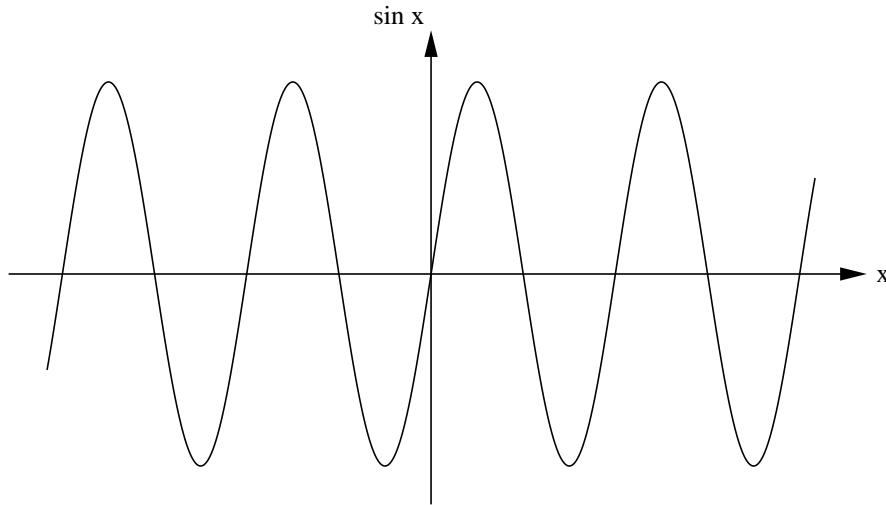


Figure 1.1: Plot of the sine function.

C was developed in 1972 by Dennis Ritchie at Bell Laboratories as a general purpose and a system programming language. It features an economy of expression, a rich set of operators, and modern data structures and control statements. C++ is an extension of C designed by Bjarne Stroustrup at AT&T Bell laboratories in the mid-eighties. C++ is object-oriented and also incorporates other features that improve the C language. C++ is currently the most popular language of choice for most commercial software developers.

Python, like Basic, originated in an educational environment. Guido van Rossum created Python in the late 80's and early 90's. It is an interpreted, interactive, object-oriented, general-purpose programming language that is also good for prototyping. Python enthusiasts like to say that C and C++ were written to make life easier for the computer, but Python was designed to be easier for the programmer.

Java is a relatively new object-oriented language that was created by James Gosling and others at Sun Microsystems. Java 1.0 was introduced in late 1995 and Java 1.1, which includes many new and important features was released in 1997. Java has become very popular and is evolving rapidly.

Programming languages are not static, but continue to change with developments in hardware and theories of computation. As with any language, a working knowledge of one makes it easier to learn another. C, Basic, and Fortran are examples of *procedural languages*. Procedural languages change the state or memory of a computer by a sequence of statements. In a *functional language* such as LISP, the focus of the language is not on changing the data in specific memory locations. Instead, a program consists of a function that takes an input and produces an output. For most of the simulations discussed in this text functional languages such as Mathematica, Matlab, and Maple and Lisp are slower than procedural languages.

Another important class of programming languages are *object-oriented* languages such as

C++, Java, Python, and Smalltalk. The idea is that functions and data are not treated separately, but are grouped together in an *object*. A program is a structured collection of objects that communicate with each other causing the data within a given object to change. One goal of the object-oriented approach is to produce libraries of objects that can be easily modified and extended for individual problems.

Our choice of Java for this text is motivated by its platform independence, intrinsic graphics statements, event-based programming capability, bit manipulation and parallel programming capability, and the fact that it is useful outside of physics so that the language will be maintained and improved and provide a marketable skill for students. It also has a growing library for doing numerical calculations, and is free or inexpensive depending on the programming environment that you choose. Java is also relatively simple to learn, especially the subset of Java that we will need to do physics.

Java has a built-in application programming interface (API) that can handle graphics and user interfaces such as buttons and text fields. Because of its rich set of API's, similar to the Macintosh and Windows, and its platform independence, Java can also be thought of as a platform in itself. Java programs are compiled to a platform neutral byte-code format so that they can run on any computer and operating system as long as the system implements the Java Virtual Machine. Although the use of a platform neutral byte-code format means that a Java program runs more slowly than platform-dependent compiled languages such as C and Fortran, this disadvantage is not important for most of the programs in this book. If a project requires more speed to obtain reasonable results, the computationally demanding parts of the program can be converted to C/C++ or Fortran.

Readers who wish to use another programming language should consider the Java program listings in the text to be pseudocode that can be converted into a language of their choice. To facilitate this conversion, sample code and a summary of the nature of Python, Fortran 90, C/C++, and True Basic is given in Appendices ??, ??, ??, and ??, respectively.

1.5 Programming Technique

If you already know how to program, try reading a program that you wrote several years, or even several weeks, ago. Many people would not be able to follow the logic of their program and consequently would have to rewrite it. And your program would probably be of little use to a friend who needs to solve a similar problem. If you are learning programming for the first time, it is important to learn good programming habits and to avoid this problem. One way is to employ object-oriented programming techniques such as inheritance, encapsulation, and polymorphism.

Inheritance allows a programmer to add capabilities to existing code without having to rewrite the code or even know the details of how the code works. For example, Chapter 2 defines an object called `Particle` that has properties of mass, position, and velocity. It also has a method called `move` that instructs the object to calculate new coordinates based upon its current position and velocity. Later in Chapter 6 we define a new object called `ChargedParticle` that adds a new property called charge and extends the move method to take into account electric and magnetic fields. The original code remains unchanged and only a small amount of code needs to be written to produce a more specialized object, a charged particle.

Encapsulation refers to the organizational principle whereby an object's essential information, such as its position, charge, and mass, is exposed through a well-documented interface, but unnecessary details of the code are hidden. Consider again the charged particle. Whenever a charged particle moves, it calculates its acceleration from the external electric and magnetic fields. Exactly how this calculation is done is important to a scientist and is the subject of this book, but inconsequential to someone who wishes to show the motion of a charged particle as part of an animation. The software engineer merely needs to know that a move method exists and how this method is invoked.

Polymorphism enables us to write code that can solve a wide variety of problems. It is easy to imagine many types of objects that depend on time and are able to move. For example, in Chapter xx we simulate an ensemble of particles in which the move method uses random numbers rather than forces to generate new configurations. But the animation procedure that is used does not need to be changed from the procedure that was used for forces because polymorphism allows us to write a program in a general way that leaves it up to the individual objects to determine how they should respond to a change in time.

Science students have special advantages in learning how to program. We know that our mistakes cannot harm the computer (except for spilling coffee or soda on the keyboard). More importantly, we have an existing context in which to learn programming. The past several decades of doing physics research with computers has given us numerous examples that we can use to learn physics, programming, and data analysis. Hence, we encourage you to learn programming in the context of the examples in each chapter.

Our experience is that the single most important criterion of program quality is readability. If a program is easy to read and follow, it is probably a good program. The analogies between a good program and a well-written paper are strong. Few programs come out perfectly on their first draft, regardless of the techniques and rules we use to write it. Rewriting is an important part of programming.

1.6 How to Use This Book

In general, each chapter begins with a short background summary of the nature of the system and the important physical questions. We then introduce the computer algorithms, new Java syntax as needed, and discuss a sample program. The programs are meant to be read as text on an equal basis with the discussions and the problems and exercises that are interspersed throughout the text. It is strongly recommended that all the problems be read, because many concepts are introduced after the simulation of a physical process. The emphasis of this book is on learning programming by example. We will not discuss all aspects of Java and this text is not a substitute for a text on Java.

It is a good idea to maintain a computer-based laboratory notebook to record your programs, results, graphical output, and analysis of the data. This practice will help you develop good habits for future research projects, prevent duplication, organize your thoughts, and save time. After a while, you will find that most of your new programs will use parts of your earlier programs. Ideally, you will use your notebook to write a laboratory report or mini-research paper on your work. Guidelines for writing such a paper are given in [Appendix 1A](#).

Many of the problems in the text are open ended and do not lend themselves to simple “back of the book” answers. So how will you know if your results are correct? How will you know when you have done enough? There are no simple answers to either question, but we can give some guidelines. First, you should compare the results of your program to known results whenever possible. The known results might come from an analytical solution that exists in certain limits or from published results. You also should look at your numbers and graphs, and determine if they make sense. Do the numbers have the right sign? Are they the right order of magnitude? Do the trends make sense as you change the parameters? What is the statistical error in the data? What is the systematic error? Some of the problems explicitly ask you to do these checks, but you should make it a habit to do as many as you can whenever possible.

How do you know when you are finished? The main guideline is whether you can tell a coherent story about your system of interest. If you have only a few numbers and do not know their significance, then you need to do more. Let your curiosity lead you to more explorations. Do not let the questions asked in the problems limit what you do. They are only starting points, and frequently you will be able to think of your own questions.

Appendix 1A: Laboratory Report

Laboratory reports should reflect clear writing style and obey proper rules of grammar and correct spelling. Write in a manner that can be understood by another person who has not done the assignment. In the following, we give a suggested format for your reports.

Introduction. Briefly summarize the nature of the physical system, the basic numerical method or algorithm, and the interesting or relevant questions.

Method. Describe the algorithm and how it is implemented in the program. In some cases this explanation can be given in the program itself. Give a typical listing of your program. Simple modifications of the program can be included in the appendix if necessary. The program should include your name and date, and be annotated in a way that is as self-explanatory as possible. Be sure to discuss any important features of your program.

Verification of program. Confirm that your program is not incorrect by considering special cases and by giving at least one comparison to a hand calculation or known result.

Data. Show the results of some typical runs in graphical or tabular form. Additional runs can be included in an appendix. All runs should be labeled, and all tables and figures must be referred to in the body of the text. Each figure and table should have a caption with complete information, for example, the value of the time step.

Analysis. In general, the analysis of your results will include a determination of qualitative and quantitative relationships between variables, and an estimation of numerical accuracy.

Interpretation. Summarize your results and explain them in simple physical terms whenever possible. Specific questions that were raised in the assignment should be addressed here. Also give suggestions for future work or possible extensions. It is not necessary to answer every part of each question in the text.

Critique. Summarize the important physical concepts for which you gained a better understanding and discuss the numerical or computer techniques you learned. Make specific comments on the assignment and your suggestions for improvements or alternatives.

Log. Keep a log of the time spent on each assignment and include it with your report.

References and Suggestions for Further Reading

Programming

We recommend that you learn programming the same way you learned English — with practice and with a little help from your friends and manuals. We list some of our favorite Java programming books here. The online Java documentation provided by Sun at <http://java.sun.com/docs/> is essential and the tutorial, java.sun.com/docs/books/tutorial/, is very helpful.

Stephen J. Chapman, *Java for Engineers and Scientists*, Prentice-Hall (2000).

Bruce Eckel, *Thinking in Java*, second edition, Prentice-Hall (2000). This text also discusses the finer points of object-oriented programming. See also <http://www.mindview.net/Books/>.

David Flanagan, *Java in a Nutshell*, third edition, O'Reilly (2000).

Brian R. Overland, *Java in Plain English*, third edition, M&T Books (2001).

Walter Savitch, *Java: An Introduction to Computer Science and Programming*, second edition Prentice-Hall (2001).

Patrick Winston and Sundar Narasimhan, *On to Java*, second edition, Addison-Wesley (1999).

General References on Physics and Computers

A more complete listing of books on computational physics is available at <http://sip.clarku.edu/books/>.

Richard E. Crandall, *Projects in Scientific Computation*, Springer-Verlag (1994).

Paul L. DeVries, *A First Course in Computational Physics*, John Wiley & Sons (1994).

Alejandro L. Garcia, *Numerical Methods for Physics*, second edition, Prentice Hall (2000). Matlab, C++, and Fortran are used.

Neil Gershenfeld, *The Nature of Mathematical Modeling*, Cambridge University Press (1998).

Nicholas Giordano, *Computational Physics*, Prentice Hall (1997).

Dieter W. Heermann, *Computer Simulation Methods in Theoretical Physics*, second edition, Springer-Verlag (1990). A discussion of molecular dynamics and Monte Carlo methods directed toward advanced undergraduate and beginning graduate students.

Rubin H. Landau and M. J. Paez, *Computational Physics, Problem Solving with Computers*, John Wiley (1997).

P. Kevin MacKeown, *Stochastic Simulation in Physics*, Springer (1997).

Tao Pang, *Computational Physics*, Cambridge (1997).

William J. Thompson, *Computing for Scientists and Engineers*, John Wiley & Sons (1992). This text contains many examples of C programs and a discussion of the nature of C.

Franz J. Vesely, *Computational Physics*, second edition, Kluwer Academic/Plenum Publishers (2001).

Michael M. Woolfson and Geoffrey J. Perl, *Introduction to Computer Simulation*, Oxford University Press (1999).

Chapter 2

Introduction to Java

©2002 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
11 February 2002

We introduce some of the basic structure and syntax of Java including classes and methods, primitive data types, loops, operator overloading, and method overriding. We also investigate the limits that finite precision arithmetic places on computation.

2.1 Introduction

Java, like any language, has a grammar or syntax that borrows from other languages including mathematics. If you are familiar with C or C++, you will find that the expressions, operators, and statements in Java are very similar. In this chapter we will introduce some of the syntax that we will need to write grammatically correct Java statements. As will be the norm, we first give a sample program and then explain the new syntax.

The following program is an example of a Java program that sums the series $\sum_{n=1}^{100} 1/n^2$:

```
public class MyFirstApp { // beginning of class definition

    public static void main(String[] args) {
        // statements within the braces {} form the body of the method
        double sum = 0;    // example of declaration and assignment statement
        for (int n = 1; n <= 100; n = n +1) { // beginning of for loop
            double term = 1.0/(n*n);
            sum = sum + term;
        } // end of for loop
        System.out.println("sum = " + sum); // print result
    } // end of method definition
} // end of class definition
```

As we will see, this program is not really object oriented, but it allows us to introduce much of the syntax of Java.

The first line declares a new class named `MyFirstApp`. The Java convention is to begin class names with an uppercase letter. If a name consists of more than one word, the words are joined together and each succeeding word begins with a uppercase letter (another Java convention). Each class must be in a separate file with the same name as the name of the class, for example, `MyFirstApp.java`. The left brace begins the body of the class definition and the corresponding right brace ends the class definition.

The second statement defines the `main` method. This method has a special status in Java. A Java application (a stand-alone program) consists of one or more class definitions, one of which must define the method `main`. (A Java applet is different insofar as that it runs inside a browser and does not usually contain a `main` method; instead it has other methods such as `init` and `start`.) The `main` method is automatically called first to start the program. We call the class that contains this method the *target* class. The syntax associated with the declaration of the `main` method is convoluted, but understanding this syntax is not really necessary. Note that methods are defined only inside a class definition. Because we will later write programs consisting of many classes, we will adopt our own convention that the filename of the target class ends with `App`. Familiarize yourself with your Java development environment by doing Exercise 2.1.

Exercise 2.1. First Application

- a. Enter the above listing of `MyFirstApp` into a file named `MyFirstApp.java`. Be sure to pay attention to capitalization because Java is case sensitive.
- b. Compile and run `MyFirstApp`. How do you know if the program is working correctly?
- c. What happens if you make a typing mistake? For example, try typing `double sun 0;` and see what happens.

Digital computers can store information only as binary numbers, that is, sequences of ones and zeros. Every one or zero is called a *bit* and a group of 8 bits is referred to as a *Byte*. For example, the integer 362 is stored as 0000000101101011 using two Bytes of memory. It would be difficult to write a program if we had to write numbers such as the speed of light in binary format. High-level computer languages allow us to reference stored numbers using *identifiers* or variable names. A valid identifiers is a series of characters consisting of letters, digits, underscores, and dollar signs that does not begin with a digit nor contain any spaces. Because Java distinguishes between upper and lowercase characters, `Sum` and `sum` are different identifiers. The Java convention is that variable names begin with a lowercase letter, except in special cases, and each succeeding word in a variable name begins with an uppercase letter.

A variable can refer to just about anything including objects such as graphs. In a purely object oriented language, all variables would be objects that we would introduce by their class definitions. However, there are certain constructs that are so basic and important that they have a special status and are especially easy to create and access. These variables are called *primitive data types* and represent integer, real, boolean, and character variables. An integer variable, a double variable, and a boolean variable are created and initialized by the following statements:

```
int numberOfParticles = 10;
double sum = 0;
boolean inert = false;
```

The primitive type `char` is used for single characters. A variable must be *declared* before it can be used, and the value of a variable can be initialized at the same time that its type is declared.

There are four types of integers, `byte`, `short`, `int`, and `long`, with the differences being the range that these types can store (see Table 2.3 in Appendix 2A). We will almost always use type `int` because it does not require as much memory as type `long`. There also are two types of floating point numbers, but we will always use type `double` to avoid excessive roundoff error.

Certain combinations of characters are part of the Java language and cannot be used as variable names. These reserved words are known as *keywords*. They are written in lowercase and include the following:

```
false  double  new  public  super  for  main  public  extends
```

A complete list is given in Appendix xx.

A statement causes Java to perform an action. It contains expressions and is terminated by a semicolon. A statement is like a complete sentence and an expression is similar to a phrase. The simplest expressions are identifiers or variables. More interesting expressions can be created by combining variables using *operators*, such as the following example of the plus operator:

```
x + 3.0
```

An expression can be evaluated to produce a value. The value of the above expression is determined by the current value of the variable `x` and the usual rules of mathematics.

The simplest statement is the *assignment statement*, for example

```
x = 1;  
c = 2.99792458e8;
```

The *assignment operator*, the equals sign, stores the value of the expression in the memory location that is associated with a variable, such as `c` in the above statement. Note how powers of 10 are written.

The following example illustrates an important difference between the equals sign in mathematics and in Java.

```
int x = 3;  
x = x + 1;
```

What is the value of `x`? The equals sign produces a replacement of a value in memory and is not a statement of equality. In fact, the left hand and right hand sides of an equals sign are usually not equal.

Java supports a variety of operators. Table 2.1 shows the operators that we will use most often. A complete listing is given in Appendix xx.

One way to sum a series of terms is to use a *loop* construction. Loops are blocks of code that are executed repeatedly. They typically require the initialization of one or more variables, a test to determine if the loop should continue or stop, and a counter variable that changes as the loop executes in order to bring about the fulfillment of the stopping condition. The `for` statement makes these three steps explicit. The loop in MyFirstApp illustrates how a `for` loop can be used to sum the first 100 terms in a series. What does the following example do?

operator	operand	description	sample expression	result
=	any	assignment	x = 7, y = 3;	7 in x and 3 in y
+, -	numbers	addition, subtraction	3.0 + x	10
*, /	numbers	multiplication, division	7/2	3
++, --	number	pre or post increment, decrement	x++;	8 stored in x
==	any	test for equality	x == y	false
!	boolean	logical complement	!(x == y)	true
+=	numbers	x+= 3; equivalent to x = x + 3;	x+ = 3;	11 stored in x
-=	numbers	x-= 2; equivalent to x = x - 2;	x- = 2;	9 stored in x
=	numbers	x= 4; equivalent to x = 4*x;	x* = 4;	36 stored in x
/=	numbers	x/= 2; equivalent to x = x/2;	x/ = 2;	18 stored in x
%=	numbers	x% = 4; equivalent to x = x % 4;	x% = 4;	2 stored in x

Table 2.1: Common operators. The operations are done in order. The operator % is the modulus operator. For example, 23 % 5 = 3.

```
for (int n = 0; n < 10; n++)
    System.out.println("n = " + n); // only one statement is performed
```

Because the test is performed before the body of the loop is executed and the increment is performed after the loop has executed, the number 0 is printed but the number 10 is not. The increment expression `n++` is equivalent to `n = n + 1` (see Table 2.1). The loop counter, `n`, is declared as part of the initialization of the loop. Declaring variables where they are used is good practice because it is easy to see what the variable does. In this case `n` is a counter, and it is not necessary to document it. Declaring `n` within the loop limits its use to within the body of the loop. The section of code where a variable can be used is referred to as its *scope*.

Note that we indented the statements within the class definition, the statements with the body of the `main` method, and the statements within the `for` loop. Indentation is done to clarify the structure of the program and is ignored by the Java compiler.

Another construction that is used to perform repetitive actions is the `while` loop. The `while` statement starts by checking a condition. If this condition is true, the following statement or group of statements is executed. The following example shows how the `while` loop is used to print the numbers 0 through 9.

```
int n = 0;
while (n < 10) { // beginning of loop
    System.out.println("n = " + n);
    n++;
} // end of loop
```

Note how braces are used to group statements to define the statements within a class, within a method, or within control statements such as the `while` or `for` loop. Also note how single line comment statements are written using `//`. Comments are very important for the user, but are ignored by the Java compiler and do not affect the speed of the program or compilation.

In `MyFirstApp`, the body of the `main` method contains a statement that prints a string that

is defined by the characters in quotes. (Strings are sequences of characters and are discussed in more detail in Chapter ??.) We note that `System.out.println` is used to display output. In brief, this syntax means that we are accessing the object `out`, which is contained in class `System`, and invoking the method `println` of the object `out` by using the *dot operator*.

Note that the text to be printed is created by adding a number to a string. Java converts the numbers into strings before *concatenating* (combining two string variables). This use of the plus operator to concatenate is an example of *operator overloading*. The plus operator acts differently when operating on two strings than when operating on two numbers. The type of variable is clearly important in determining if `4 + 5` should produce the integer number 9 or the string 45.

Exercise 2.2. While loop

- a. Replace the body of the main method in `MyFirstApp` class by the following code:

```
double sum = 0.0;
double maxChange = 1.0e-4; // maximum difference
double change = 1.0;
int n = 0;
while (change >= maxChange) {
    n++; // equivalent to n = n + 1
    double term = 1.0/(n*n);
    sum = sum + term;
    change = term/sum; // relative change
} // end of while loop
```

This code sums the terms in the series until the relative change in the sum is less than the desired amount.

- b. How many terms are required to achieve a relative change of less than 10^{-2} and less than 10^{-4} ? What do you think is the result of adding an infinite number of terms in the series $\sum_{n=1}^{\infty} 1/n^2$?

2.2 Static Methods

All Java programs are built from *classes*, which define a collection of related *data* and *methods*. Data is stored in variables which can refer to just about anything. However, there are two basic types of data, and these types are used very differently within a program. Primitive data types, such as `x` and `n`, are used to store numeric values and single characters. (There are eight primitive data types.) Composite data types, such `Complex` (see Section 2.4.3), group data in such a way that it can be referred to by a single variable.

Methods are operations on data. They are similar to functions or subroutines in other programming language with the important distinction that they have access to all data within a class. In most cases, classes cannot be used until they are instantiated using the `new` operator as explained in Section 2.3. However, some classes define *static* data and methods that can be used immediately. We discuss static methods first because they are used to define functions that can be used (invoked) anywhere in a Java program and because they are essential to starting a Java application.

Java provides many already defined mathematical methods in the `Math` class. These methods are static. For example, we can calculate the square root of 2.0 and set it equal to a double precision variable by the statement

```
double x = Math.sqrt(2.0); // example of the use of a static method
```

Note how we have used the dot operator to access the `sqrt` method of the `Math` class.

The method `Math.sqrt()` computes the square root of any value passed to it and returns the result. This method is independent of any data stored in the `Math` class and hence the method is static. Hence, we can use `Math.sqrt()` without first instantiating an object from the `Math` class. The distinction between static and non-static methods will become more apparent in Section 2.4, where we will see that we must first create an *object* or an *instance* of a class before we can use its non-static methods. In this section we will use only static methods.

The `Math` class provides many common mathematical methods, including trigonometric, logarithmic, exponential, and rounding operations, and predefined constants. Some examples that use the `Math` class include:

```
double theta = Math.PI/4; // constant pi defined in Math class
double u = Math.sin(theta); // sine of theta
double v = Math.log(0.1); // natural log of 0.1
double w = Math.pow(10,0.4); // 10 to the 0.4 power
double x = Math.atan(3.0); // arc tangent
```

The above statements illustrate that a class is described by its methods. These methods tell us what this class can do; we don't have to know about the details of how the methods are implemented. Note the Java convention that constants such as the value of π are written in uppercase letters, that is, `Math.PI`.

Exercise 2.3 asks you to read the `Math` class documentation and Exercise 2.4 asks you to use a trigonometric identity to test the accuracy of several of the methods in the `Math` class.

Exercise 2.3. The Math class

- Read the documentation for the `Math` class. The Java documentation is a part of most development environments and is available online at <http://java.sun.com/j2se/1.3/docs/api/>. It also can be downloaded to your computer.
- How many methods are defined in the `Math` class? How many constants?
- Describe the difference between the two versions of the arctangent method.
- Write a program to verify the output of various methods in the `Math` class.

We now list a program that tests the accuracy of the calculation of the trigonometric identity, $\sin x = \sqrt{1 - \cos^2 x}$. Create a file named `MathApp.java` and enter the following statements:

```
public class MathApp {
    public static void main(String[] args) {
        double theta = Math.PI/4.0; // angle in radians
        double a = Math.sin(theta);
```

```

    double b = Math.sqrt(1 - Math.cos(theta)*Math.cos(theta));
    System.out.println("sin(theta) = " + a);
    System.out.println("value of trigonometric identity = " + b);
    double difference = a - b;
    System.out.println("difference = " + difference);
}
}

```

Exercise 2.4. Trigonometric identity

Use the `MathApp` program to do the following:

- Determine if the trigonometric identity, $\sin x = \sqrt{1 - \cos^2 x}$, is satisfied exactly. Is the difference between the computed values of $\sin x$ and $\sqrt{1 - \cos^2 x}$ zero? If not, why not?
- Change the angle at which the trigonometric identity is being tested to $\pi/4000.0$. Does the difference change? Explain.
- Which variable do you think is most accurate, a or b ?

The `main` method is a static method, but has some important differences from most static methods. Because `main` does not calculate any values, its return value is `void`. In contrast, the return value for `Math.sin` is declared to be `double`. The arguments for the `main` method are also different than the `Math.sin` method. The `sin` method is passed a double variable, in contrast to the `main` method which is passed an array of strings, `String[] args`. (Arrays are defined in Section 3.4.) Although the argument to the `main` method is not used, it must be included.

The arguments of a method are called the *parameter list*. The parameters in the list are separated by commas and the type must be declared for each parameter in the list. If the method does not receive any parameters, the parentheses are still required. The type of data that is passed to a method must match the type of data in the method's *signature*, the combination of the method number, type, and order of arguments in the parameter list. For example, we cannot call the method `Math.sin("two")`, because the `Math.sin` method expects a number not a string.

We next define a new class that defines some hyperbolic functions that are not included in the `Math` class. The methods of `MyMath` can be invoked by statements similar to the following:

```

// test the hyperbolic tangent
double x = 0.5;
double y = MyMath.tanh(x);
System.out.println("x = " + x + " tanh(y) = " + y);

```

These statements should be inserted into the `main` method of the target class `MyMathApp`.

The hyperbolic sine (`sinh`), cosine (`cosh`), and tangent (`tanh`) functions can be defined in terms of the exponential function.

$$\sinh u = \frac{e^u - e^{-u}}{2} \quad (2.1a)$$

$$\cosh u = \frac{e^u + e^{-u}}{2} \quad (2.1b)$$

$$\tanh u = \frac{e^u - e^{-u}}{e^u + e^{-u}}. \quad (2.1c)$$

We implement these definitions in class MyMath as follows:

```
public class MyMath {

    private MyMath(){}           // prohibit instantiation

    public static final double MAX_EXPONENT = Math.log(Double.MAX_VALUE);
    public static final double MIN_EXPONENT = Math.log(Double.MIN_VALUE);

    public static double sinh(double u) {
        return (Math.exp(u) - Math.exp(-u))/2;
    }

    public static double cosh(double u) {
        return (Math.exp(u) + Math.exp(-u))/2;
    }

    public static double tanh(double u) {
        if (u > MAX_EXPONENT)
            return 1.0;
        else if (u < MIN_EXPONENT)
            return -1.0;
        else
            return (Math.exp(u) - Math.exp(-u))/(Math.exp(u) + Math.exp(-u));
    }
}
```

The term **public** in the first line means that the class can be used by other classes. The first two statements in the body of the class definition are examples of definitions of *named constants*. For example, we could define the speed of light c by

```
public static final double SPEED_OF_LIGHT = 3.0e8;
```

The keyword **static** means that this variable can be accessed using the class name. The keyword **final** means that it is not possible to change the value of **SPEED_OF_LIGHT** after it is given an initial value. By convention, named constants are written in uppercase letters with underscores used to separate the words.

The **Double** class defines the static constants, **Double.MAXIMUM** and **Double.MINIMUM**, as the maximum and minimum allowable value of the double primitive type. The **Double** class is distinct from the **double** primitive type. How would you find the largest and smallest values of type **int**? How would you find the largest and smallest values of type **int**?

We evaluate the constants **MAX_EXPONENT** and **MIN_EXPONENT** using the inverse exponential, the logarithm, to define the largest argument of the **tanh** function that can be evaluated accurately using exponentials. It is important to avoid using large arguments even though the **tanh** function is well defined.

The above implementation of the hyperbolic tangent introduces the **if** statement and the operator corresponding to less than. The logical operators evaluate expressions to produce boolean values of **true** or **false** (see Table 2.2). The **if** statement evaluates an expression and decides

which statement to execute. The following example shows how the `if` statement can be used to test for illegal negative numbers before the square root method is invoked.

```
if (x >= 0)
    y = Math.sqrt(x);
```

If the boolean expression produces a value of `true` then the statement is executed. Otherwise, the statement is skipped and the program continues. If there is only one statement to evaluate as in the above example, we will write it as two lines and indent the second line for clarity.

It is possible to evaluate more than one statement by enclosing a block of code in braces. A block of code is known as a *compound statement* and can be used anywhere a single statement is required. For example, we can write

```
if (x >= 0) {
    y = Math.sqrt(x);
    System.out.println("y = " + y);
}
```

The `if/else` statement is similar to the `if` statement except that it chooses between two statements. The first statement is executed if the expression is `true`; the second statement is executed if the expression is `false`.

```
if (x >= 0)
    y = Math.sqrt(x);
else
    y = Math.sqrt(-x);
```

`If` statements can be combined as shown in the definition of the `tanh` method. If the first expression is true, then the method exits and the value returned is 1.0. If the first expression is false, then a second `if` statement is executed. There are again two possibilities each of which exits with a different return value.

operator	description	sample expression	result
<code>></code>	greater than	<code>7 > 7</code>	<code>false</code>
<code>>=</code>	greater than or equal	<code>7 >= 7</code>	<code>true</code>
<code><</code>	less than	<code>3 < Math.PI</code>	<code>true</code>
<code><=</code>	less than or equal	<code>3 <= 7</code>	<code>true</code>
<code>==</code>	equal	<code>7 == 3 + 4</code>	<code>true</code>
<code>!=</code>	not equal	<code>7 != 15/2</code>	<code>true</code>
<code>&&</code>	logical AND	<code>7 > 2 && 5 < 3</code>	<code>false</code>
<code> </code>	logical OR	<code>7 > 2 3 < 5</code>	<code>true</code>

Table 2.2: Common Java relational and logical operators.

So far our programs have consisted of no more than two files, for example, `MyMath.java` and `MyMathApp.java`. The Java compiler will be able to find the two files as long as they are in the same directory. In Section 2.4.2 we will learn how to make *packages* so that the files can be placed anywhere.

Exercise 2.5. MyMath class

- a. Write a program by replacing the body of the `main` method in the `MyMathApp` class with code that tests the three hyperbolic functions. For example, write code such as the following:

```
double x = 0.2;
double y = MyMath.sinh(x);
System.out.println("x = " + x + " sinh(x) = " + y);
```

Check your values using a calculator. Find the maximum and minimum values for the arguments that produce correct results for all three functions.

- b. Remove the exponent range check from the `tanh` method. How does this change effect the domain of the function?
- c. Test the numerical accuracy of the identities $\cosh^2 x - \sinh^2 x = 1$ and $\cosh 2x = \cosh^2 x + \sinh^2 x$.

2.3 Non-Static Methods

There are two general types of methods in Java, *static* and *non-static*. We have already used static methods to define mathematical functions and the static method `main` to start a Java application. The syntax that is used to write static methods is common to most modern programming languages that discuss non-static methods.

Assume that we have two non-interacting particles constrained to move in one dimension. Each mass is modeled in a class named `ParticleFree`. Our goal is to write a small Java application that tests this class. Although the program will not run because we have not yet defined the `ParticleFree` class, `ParticleFreeApp` might be written as follows:

```
public class ParticleFreeApp { // ParticleFreeApp class begins

    public static void main(String args[]) { // main method begins
        ParticleFree p1 = new ParticleFree(2.0, 1.0);
        // create particle with y = 2, vy = 1
        ParticleFree p2 = new ParticleFree(5.0, 3.0); // create another particle
        System.out.println("particle 1:" + p1); // print state of particle 1
        System.out.println("particle 2:" + p2); // print state of particle 2
        double dt = 0.1; // time step for particle motion
        p1.move(dt); // move particle 1
        p2.move(dt); // move particle 2
        System.out.println("particle 1:" + p1); // print state of particle 1
        System.out.println("particle 2:" + p2); // print state of particle 2
    } // main method ends
} // ParticleFreeApp class ends
```

Note the differences in syntax from the `MathApp` program. In particular, note that we did not write

```
ParticleFree.move(dt);
```

but instead wrote

```
ParticleFree p1 = new ParticleFree(2.0, 1.0);
```

followed by

```
p1.move(dt);
```

This syntax change is a manifestation of the difference between static and non-static methods.

One of the most important object oriented programming concepts is the distinction between an object's definition, which is given by a class, and the actual object itself. The creation of an object requires that the class contain a special method known as a *constructor*. Constructors are easy to spot because they have the same name as the class. Unlike other methods, constructors do not have a return type. Constructors are called automatically by Java when an object is created using the `new` operator and allocate computer memory for the object. The following statement invokes the `ParticleFree` constructor and assigns the object to the variable `p1`.

```
p1 = new ParticleFree(2.0, 1.0);
```

Like other methods, constructors can have arguments, which are often used to initialize variables as is the case here. (In this case, the first parameter is the position of the particle and the second parameter is its velocity.)

The `ParticleFreeApp` class *instantiates* two `ParticleFree` objects and allows us to move these objects by invoking the `move` method of `p1` and `p2`. Objects are data as are the primitive data types, such as `int` and `double` which we have already encountered. However, an identifier that refers to an object does not contain the entire object, whereas a variable that refers to a primitive data type contains the actual data. An object refers to an address in memory (called a *pointer* in some languages) to where the object is stored. Java (unlike C) does not allow you to access this address directly, so you will sometimes read that Java does not support pointer arithmetic. The difference between primitive data types and objects is explored in Exercise 2.8.

2.4 Examples of Classes

We now introduce several examples of classes with non-static methods. As before, we introduce this concept by considering several examples.

2.4.1 One-Dimensional Motion

Consider the motion of a particle freely falling near the Earth's surface due to the gravitational attraction of the Earth on the particle. If we neglect air resistance, we know that the acceleration of the particle is constant with magnitude $g \approx 9.8 \text{ m/s}^2$. Let us adopt a coordinate system such that y increases upward. With this choice the acceleration $a = -g$. For this case of constant acceleration, the solutions of Newton's second law of motion can be expressed as

$$v(t) = v_0 - gt \tag{2.2a}$$

and

$$y(t) = y_0 + v_0 t - \frac{1}{2} g t^2, \tag{2.2b}$$

where y_0 and v_0 are the initial position and velocity of the particle, respectively.

Clearly, the analytical solution in (2.2) is so simple that we do not need to write a program to compute the position and velocity of the particle. But we have to start somewhere, so let us discuss how we might write a program to compute the position and velocity of a freely falling particle at a particular time.

Our first step is to define a `ParticleFree` class. In this case it is straightforward to think of a particular particle as an object of the class `ParticleFree`. We know that a particle is characterized by its position, velocity, acceleration, and mass. For simplicity, we consider motion in the vertical direction only. Our class looks like the following:

```
public class ParticleFree extends Object {
    double mass = 1.0;
    double y,v;      // position , velocity
    static final double g = 9.8; // acceleration near earth's surface
    double a = -g; // acceleration

    public ParticleFree(double _y, double _v) {
        y = _y;
        v = _v;
    }

    public void move(double dt) {
        v = v + a*dt; // dt is the time step
        y = y + v*dt + 0.5*a*dt*dt;
    }

    public String toString() {
        return "y = " + y + "," + "v = " + v + "," + "a = " + a;
    }
}
```

Because the acceleration due to the earth's gravity is independent of the mass of a particle, we have set the particle's mass equal to unity. Note that we have chosen a coordinate system such that the height y and the velocity v of the particle decrease with time. We have defined `ParticleFree` such that it can "print" itself using the `toString()` method, that is, display its position, velocity, and acceleration. In addition, we have included the method `move` that uses the analytical solution of a freely falling body in one dimension to compute the position and velocity.

The first line of the `ParticleFree` class states that `ParticleFree` is a *subclass* of the `Object` class. The keyword `extends` declares that `ParticleFree` has all the functionality of `Object`. The `extends` declaration is optional in this case because all classes implicitly extend `Object`, but we have included this declaration here to show explicitly that all classes *inherit* from `Object`. We say that `ParticleFree` is a *subclass* of `Object` and the latter is a *superclass*. We will not explicitly extend `Object` in future classes that we write.

Because `ParticleFree` is a subclass of `Object`, it automatically *inherits* the methods of `Object`. One of the methods of `Object` is the `toString()` method, which converts the data stored in an object into a format suitable for displaying. To display the data in `ParticleFree`, we need to *override* the `toString()` method and customize it. Several methods can be defined with the

same name if the methods have a different set of parameters or signature.

The variables `y`, `v`, `a`, and `mass` are part of the class definition and are called *member* or *instance* variables. Their values can be different for every instance of the class. That is, every instance of `ParticleFree` is allocated memory to store its own copy of these variables.

The next group of statements define the *constructor* for the class `ParticleFree`. As explained earlier, a constructor is a special static method that is called to create an instance of a class. A constructor always has the same name as the class that it initializes and does not have a return data type. A constructor always exists. If we do not define at least one constructor, the Java compiler supplies a default constructor with no parameters.

The `ParticleFree` constructor requires two initialization parameters, the position and the velocity. We could define this constructor in different ways so as to unambiguously show the difference between these passed variables and member variables. Two possible ways are as follows:

```
public ParticleFree(double y, double v) {
    this.y = y;
    this.v = v;
}
```

or

```
public ParticleFree(double _y, double _v) {
    y = -y;
    v = -v;
}
```

Instance variables have *class scope* so they are accessible in any non-static method (including constructors) in the class. In this case, the variables `y`, `v`, `a`, and `mass` are defined such that they have class scope. Class scope begins at the opening left brace of a class definition and ends at the closing right brace of the class definition. Class scope means that any (non-static method) in the class can access `y`, `v`, `a`, and `mass` directly. However, local variables have precedence if there is a name conflict. In this case the local variables `y` and `v` in the parameter list have the same name as the class instance variables `y` and `v`. Hence, the constructor `ParticleFree` cannot use the names `y` and `v` to access the instance variables. Instead, the instance variables are referred to as `this.y` and `this.v`. The reference `this` always refers to the current object, which is the object within which the reference appears. Hence, `this.y` refers to the instance variable `y` defined in the current object. To avoid the confusion between instance and local variables with the same names, the second example adds an underscore to the variables in the parameter list. We will adopt the convention that a passed variable that has a name conflict with an instance variable will be renamed with an underscore.

We now write `ParticleFreeApp` so that it will compute the position, velocity, and acceleration of a freely falling particle at different times and display this information on the screen.

```
public class ParticleFreeApp {
    public static void main(String[] args) {
        ParticleFree p;
        p = new ParticleFree(20,0); // create a particle with x = 20 and v = 0
        double t = 0.0;           // elapsed time from instantiation
        double dt = 0.1;
```

```

while (p.y >= 0) {
    p.move(dt);
    t = t + dt;
    // equivalent to System.out.println(t + "\t" + p.toString());
    System.out.println(t + "\t" + p); // \t is tab
}
}
}

```

The first statement of the `main` method declares `p` to be an object of type `ParticleFree`. This statement does not create the object, but says only that `p` is a suitable name for an object of type `ParticleFree`. That is, `p` can store a reference to an object of type `ParticleFree` if and when we create one.

The second line with the keyword `new` actually creates an object of type `ParticleFree` and associates it with the name `p`. We can think of `new` as creating the instance variables of the object `p`. We can combine these two statements into the single statement:

```
ParticleFree p = new ParticleFree(20,0); // new particle with x = 20 and v = 0
```

In the definition of class `ParticleFree`, we defined a constructor with two parameters so that the instance variables `y` and `v` are initialized using the values of the appropriate parameters.

Method `move` takes one parameter and uses the analytical solution (2.2) of a freely falling body to compute the position and velocity of the body. The variable `dt` can be thought of as a place holder for the numerical values that will be passed to the method when it is executed. Note that we have defined the return type of method `move` to be `void`.

In our example of class `ParticleFreeApp`, the `main` method creates the object `p` of type `ParticleFree` with initial height 20 (meters) and zero velocity. Because we want to access the variable `y` associated with this instantiation of a `ParticleFree`, we use the dot notation and write `p.y`. The variable `t` is an example of a *local variable*, a variable that is defined and accessible only within the same method. Note the use of the `while` loop and how the coordinates of the particle are displayed.

Exercise 2.6. Freely falling body

- a. Compile and run `ParticleFreeApp` and determine the time it takes for a particle to reach the Earth's surface. Check that the computed result is consistent with the analytical solution.
- b. Write a class that instantiates two particles, `p1` and `p2` with different initial positions and velocities. Suppose that `p1.y = 20`, `p1.v = 0`, and `p2.y = 30`, `p2.v = 5`. Which particle reaches the ground first?

Exercise 2.7. Testing the `ParticleFree` class

As mentioned in the text, it is possible to put a `main` method in any class. Because `main` is a static method, we have to treat it just like it was in another file. That is, within the `main` method we must create an object of type `ParticleFree` before we can call any of its methods. Modify class `ParticleFree` by putting a `main` method in it, and show that it is necessary to create an object of the class before you use any of the methods or access any of the instance variables of `ParticleFree`.

Exercise 2.8. Difference between types of variables

- a. Consider the primitive data types in the following code:

```
double x = 2.0;
double y = 3.0;
x = y;
System.out.println("x = " + x);
System.out.println("y = " + y);
```

What are the values of x and y? What if we then write

```
y = -4;
```

What are the values of x and y now? Write a short target application that displays the values. In this case, the identifiers x and y are primitive data types.

- b. Now consider the class data types in the following:

```
p1 = new ParticleFree(-2.0, -1.0);
p2 = new ParticleFree(2.0, 1.0);
p1 = p2;
System.out.println(p1);
System.out.println(p2);
```

In this case the identifiers p1 and p2 refer to the same object. If we move either p1 or p2, we would move the same particle. In fact, it is impossible to access the original particle 2 unless we saved a reference to this particle in another identifier.

As you use objects in various contexts, you will become aware that defining an object name does not create the object. For example, we could write the following:

```
ParticleFree p3;
```

Particle 3 does not yet exist because Java has created only the variable name. We must first create the object using the **new** operator before particle 3 can be used.

```
p3 = new ParticleFree(-2.0, 1.0);
```

We will also appreciate the fact that *instantiated objects can remember their state*. It is possible to define a static hyperbolic tangent function because each evaluation of the function produces exactly the same result. But if we want multiple particles to have different states, then the particle class must contain non-static variables and methods.

2.4.2 Packages and Access Modifiers

Anyone who has ever used a computer knows how difficult it can be to find a misplaced file on a hard drive. Directories, or folders, can make the job of organizing your work easier, but only if you follow consistent naming conventions. *Packages* are the Java equivalent of directories. Imagine the mess, not to mention naming conflicts, if thousands of Java class files that you and others wrote

were located in a single directory on your hard disk. Packages allow programmers to organize their code into logical units and provide a syntactic structure that enables programmers to use code in other directories. Creating a package is equivalent to creating a subdirectory that stores files containing Java source code.

Packages can have subpackages just as directories can have subdirectories. Various operating systems use slash, backslash, or a colon to designate the path from a directory to a subdirectory to a file. Java uses the dot notation which we have already encountered. For example, the `Math` class is located in the package `java.lang`, which means that the designers of Java can find the code for the `Math.java` class by starting in a directory called `java` and going to a subdirectory called `lang`. Whenever packages are used, Java requires that the first statement of the source code contain a *package declaration*. For example, the first statement inside the `Math.java` file is `package java.lang;`. The package declaration must exactly match the directory names.

The programs for this book is organized by chapter and is located in packages that end with the chapter number. However, because you will use the `MyMath` class in various places throughout the text, we suggest that the code for the `MyMath` class be placed in a package called `org.opensourcephysics.sip.extras`. To shorten the code listings, we will usually omit the package declaration in the narrative.

So far we have declared the methods of the several classes that we have written to be `public`, so that the methods can be used by any other class. However, we have not declared the instance variables of the various classes to be either `private` or `public`. If we do not specify the visibility level of a variable, the default is “package visibility,” that is, the instance variable can be accessed only within the class that defines it and within classes that are part of the same package. In contrast, `private` instance variables and methods can only be accessed from within their defining class. Finally, there is a fourth access modifier that provides an intermediate level of protection which is slightly less restrictive than the default. The `protected` access modifier states that a variable or a method is accessible to all classes in the same package and all subclasses even if the subclasses are not in the same package.

Many Java textbooks stress that it is not good programming practice to make the instance variables of a class `public` and all instance variables should be declared `private`. If an instance variable is `public`, then *any* other class can directly access the instance variable and change its value. Such a change might lead to unforeseen circumstances. For class `ParticleFree`, the mass does not affect the acceleration of a freely falling particle even though the mass of a particle is essential information in general. But for example, if we were to take into account drag resistance (as we do in Chapter 5), we would have to incorporate the dependence of the drag resistance on the mass (and shape) of the falling body. Because it is necessary to recalculate the drag coefficient when the mass changes, we should modify the `ParticleFree` class so that the mass variable is `private`. In that case we should also provide a method to access the mass of the particle and a method to set its mass:

```
public class ParticleFree {  
    private double mass;  
    double y,v,a;  
  
    public void setMass(double _mass) {  
        mass = _mass;  
        // code to calculate drag goes here (see Chapter 3)  
    }  
}
```

```

    }

    public double getMass() {
        return mass;
    }
}

```

For brevity, we have omitted the other methods. If we want to get/set the value of the mass of the particle, we would write something like

```

ParticleFree p = new ParticleFree();
p.setMass(3.0);
double mass = p.getMass();

```

Providing accessor methods such as the `getMass` and `setMass` methods in the above produces much extra code that may obscure the physics. Our approach will be to allow package visibility if there are only a few interrelationships. We assume that the package author knows what is going on inside his or her classes. However, we will in general not use `public` instance variables and will provide accessor methods when necessary. In Chapter 3, we will begin to use *protected* variables, which are accessible to all classes in the same package and all subclasses even if the subclasses are not in the same package.

2.4.3 Complex Numbers

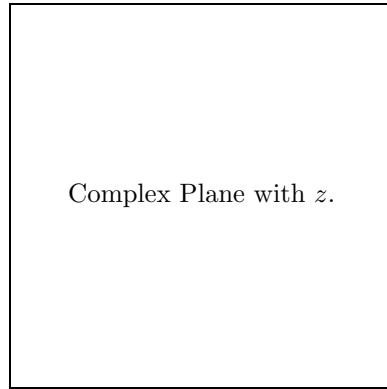


Figure 2.1: A complex number, z , can be defined by its real and imaginary parts, x and y , respectively, or by its magnitude, $|z|$, and phase angle, θ .

Complex numbers are used in physics to represent quantities such as alternating currents and quantum wavefunctions that have an amplitude and phase (see Figure 2.1). Java does not provide a complex number as a primitive data type, so we will build a class that implements some common complex arithmetic operations. This class is an explicit example of the statement that classes are effectively new programmer-defined types. If our new class is called `Complex`, we could test it using code such as the following:

```

public class ComplexApp {

    public static void main(String[] args) {
        Complex a = new Complex(3.0, 1.0); // complex number 3 + i1
        Complex b = new Complex(1.0,-1.0); // complex number 1 - i1
        System.out.println(a); // print a using a.toString()
        System.out.println(b); // print b using b.toString()
        a.conjugate(); // complex conjugate of a
        System.out.println(a); // print a
        Complex sum = b.add(a); // add a to b
        System.out.println(sum); // print sum
        Complex product = b.multiply(a); // multiply b by a
        System.out.println(product); // print product
    }
}

```

Note that `ComplexApp` is the target class. Because the methods of class `Complex` are not static, we must first *instantiate* a `Complex` object with a statement such as

```
Complex a = new Complex();
```

The variable `a` is an object of class `Complex`. As before, we can think of `new` in the above as creating the instance variables and memory of the object. Compare the form of this statement to the declaration,

```
double x = 3.0;
```

A variable of class type `Complex` is literally more complex than a primitive variable because its definition also involves associated methods and instance variables.

Note that we have first written a class that uses the `Complex` class before we have actually written the latter. Although programming is an iterative process, it is a good idea to think about how the objects of a class are to be used first. Exercise 2.9 encourages you to do so.

Exercise 2.9. Complex Number Test

What will be the output when `ComplexApp` is run? Make reasonable assumptions about how the methods of `Complex` will perform using your knowledge of Java and complex numbers.

Clearly, we need to define methods that add, multiply, and take the conjugate of complex numbers and define a method that prints their value. We next list the code for the `Complex` class.

```

public class Complex {
    double real = 0;
    double imag = 0; // real, imag are instance variables

    public Complex() { } // use default value 0 + i0

    public Complex(double _real, double _imag) {
        real = _real;
        imag = _imag;
    }
}

```

```

public void conjugate() {
    imag = -imag;
}

public Complex add(Complex c) {
    /* result is also complex so need to introduce another
     variable that is of type Complex */
    Complex sum = new Complex();
    sum.real = real + c.real;
    sum.imag = imag + c.imag;
    return sum;
}

public Complex multiply(Complex c) {
    Complex product = new Complex();
    product.real = real*c.real - imag*c.imag;
    product.imag = real*c.imag + imag*c.real;
    return product;
}

public String toString() {
    // note method overriding
    if (imag >= 0)
        return real + " + i" + Math.abs(imag);
    else
        return real + " - i" + Math.abs(imag);
}
}

```

Notice how class `Complex` *encapsulates* (hides) both the data and the methods that characterize a complex number. That is, we can use the `Complex` class without any knowledge of how its methods are implemented or how its data is stored.

The general features of this class definition are as before. The variables `real` and `imag` are the instance variables of class `Complex`. In contrast, the variable `sum` in method `add` is a local variable because it can be accessed only within the method in which it is defined.

In this case there are two constructors that are distinguished by their parameter list. The constructor with two arguments allows us to initialize the values of the instance variables. As mentioned on page 25, if a constructor is not defined, Java defines a default constructor with no parameters. However, once we add at least one constructor to a class, then Java does not create a default constructor.

The most important new feature of the `Complex` class is that we have used the class definition itself to define the return type of methods `add` and `multiply`. One reason we need to return a variable of type `Complex` is that a method returns (at most) a *single* value. For this reason we cannot return both `sum.real` and `sum.imag`. More importantly, we want the sum of two complex numbers to also be of type `Complex` so that we can add a third complex number to the result. Note also that we have defined methods `add` and `multiply` so that they do not change the values of the instance variables of the numbers to be added, but create a new complex number that stores the

sum. Finally, the class definition includes an example of how to write comment statements that span multiple lines.

Another way to represent complex numbers is by their magnitude and phase, $|z|e^{i\theta}$. If $z = a + ib$, then

$$|z| = \sqrt{a^2 + b^2}, \quad (2.3a)$$

and

$$\theta = \arctan \frac{b}{a}. \quad (2.3b)$$

We investigate this alternative representation in Exercise 2.10.

Exercise 2.10. Complex Numbers

- a. Add methods to get the magnitude and phase of a complex number, `getMagnitude` and `getPhase`, respectively. Add test code to invoke these methods. Be sure to check the phase in all four quadrants.
- b. Create a new class named `ComplexPolar` that stores a complex number as a magnitude and phase. Define methods for this class so that it behaves the same as the `Complex` class. Test this class using the code for `ComplexApp`.

Exercise 2.11. Final

The structure of the `Math` class defined by Java looks something like

```
public final class Math {

    public static final double PI = 3.14159265358979323846;
    public static final double E = 2.7182818284590452354;

    public static double cos(double a) {
        // code to calculate cos goes here
    }
}
```

Look at the online Java documentation or a Java textbook (see the references for suggestions) and read about the significance of `final` in the `Math` class definition. What is the significance of the keywords `public static final` in the definition of π ? How would you modify the `MyMath` class so that its method definitions cannot be changed?

Exercise 2.12. Static variables

Consider what happens when you make the following changes in the `Complex` class:

```
public class Complex {
    static int counter = 0; // # of complex numbers that have been created
    int index = 0;

    public Complex (double _real, double _imag) {
        index = ++counter;
    }
}
```

Because counter is a static variable, there is only *one* value for all objects that are instantiated from the `Complex` class. When a new `Complex` object is created, `counter` is increased by unity and stored in the variable `index`. Change the `toString` method to print the index in addition to the real and imaginary components. Write a program that illustrates these properties.

2.5 Summary

So far we have seen that classes and methods form the basis of *object oriented programming*. A note of encouragement is in order here; it will take time and practice to become a good object oriented programmer. There is much about Java that is slow to reveal itself, and we have only scratched the surface of object oriented programming in this introductory chapter. Object oriented programming is a design philosophy that will shape the structure of the programs that we write. But as with many seemingly straightforward ideas, for example, Galilean and Lorentz invariance, the full implication of object oriented programming may not be immediately obvious.

Appendix 2A

type	contains	default	size	range
<code>boolean</code>	<code>true or false</code>	<code>false</code>	1 bit	NA
<code>char</code>	unicode character	<code>\u0000</code>	16 bits	<code>\u0000</code> to <code>\uFFFF</code>
<code>byte</code>	signed integer	0	8 bits	-128 to 127
<code>short</code>	signed integer	0	16 bits	-32768 to 32767
<code>int</code>	signed integer	0	32 bits	-2147483648 to 2147483647
<code>long</code>	signed integer	0	64 bits	-2^{63} to $2^{63} - 1$
<code>float</code>	floating point	0.0	32 bits	$\pm 1.4E-45$ to $\pm 3.4028235E+38$
<code>double</code>	floating point	0.0	64 bits	$\pm 4.9E-324$ to $\pm 1.7976931348623157E+308$

Table 2.3: Primitive data types, their memory requirements, and their range.

References and Suggestions for Further Reading

Stephen J. Chapman, *Java for Engineers and Scientists*, Prentice-Hall (2000).

Bruce Eckel, *Thinking in Java*, second edition, Prentice-Hall (2000). This text also discusses the finer points of object-oriented programming. See also <http://www.mindview.net/Books/>.

David Flanagan, *Java in a Nutshell*, third edition, O'Reilly (2000).

Walter Savitch, *Java: An Introduction to Computer Science and Programming*, second edition, Prentice-Hall (2001).

Chapter 3

Inheritance and Interfaces

©2002 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
16 February 2002

We discuss some further examples of inheritance and then introduce the idea of an interface.

3.1 Inheritance

In Section 2.4 we introduced the idea of *inheritance*, one of the cornerstones of object oriented programming. Inheritance allows one or more subclasses to perform methods and access data in the parent class or *superclass*. If class B and class C inherit from class A, both B and C automatically inherit A's methods and (public and protected) instance variables, but not A's constructors.

The general rule is that subclasses are more specialized than superclasses. For example, in introductory physics we often define a particle to be a structureless mass with specified coordinates. A particle falling under the action of gravity without drag and a particle falling under the action of gravity with drag are specialized examples of particles. We can define a *class hierarchy* using a `Particle` superclass and two subclasses, `ParticleFree` and `ParticleDrag`, to make the relationships explicit.

At this early stage of learning Java, our examples of inheritance will be relatively simple and the advantages of inheritance will not be much better than “copying and pasting.” Many of our future uses of inheritance will not be obvious and will involve implicit inheritance from `Object` and the use of inheritance in the graphical input and output classes that we will develop (see Chapter 4).

Let's return to the particle example considered in Section 2.4.1 and consider the fall of a feather and rock. For simplicity, we will ignore the effect of drag resistance on the fall of the rock, but obviously we have to take into account the effect of drag resistance due to the earth's atmosphere on the fall of a feather. Both the rock and the feather are examples of particles (if we ignore their internal motion), but the motion of their center of mass can be quite different. For this reason we define the `Particle` class as follows:

```

package org.opensourcephysics.sip.ch3; // note package name

public abstract class Particle {
    protected double mass = 1.0;
    protected double y,v,a; // position, velocity , acceleration
    // static constants are usually upper case; break convention here
    public static final double g = 9.8;

    public Particle(double _y, double _v) {
        y = _y;
        v = _v;
    }

    public abstract void move(double dt);

    public String toString() {
        return "y = " + y + "," + "v = " + v + "," + "a = " + a;
    }
}

```

Note that `Particle` is an *abstract* class, which means that it cannot be used directly. Also note that we have also declared the `move` method to be abstract. In the latter context, the *abstract* keyword is used whenever we wish to declare a method, but do not yet know how it will be implemented. In this case we know that all particles will move, but we do not yet know how they will move. Abstract methods, such as `move`, contain only a header with no method body. The advantage of including the `move` method in `Particle` is that all classes that inherit from `Particle` can differ in the way this method is implemented. By creating abstract classes, we require common methods that all subclasses must have.

You might wonder why we didn't define the `move` method as `public void move(double dt)`, in which case we would not have to make `Particle` abstract. The reason is that `Particle` does not express a particular implementation of a particle, and hence creating a `Particle` object makes no sense. A class that declares at least one abstract method must declare itself to be abstract in the class header and cannot be instantiated.

We defined the instance variables `mass`, `y`, `v`, and `a` as *protected*, because all classes in the package as well as all their subclasses might need to have access to these variables.

Exercise 3.1. Directory hierarchy

Given that there is a directory named `org` that contains a directory named `opensourcephysics`, what is the directory hierarchy implied by the package statement in the file `Particle.java`? The acronym `sip` stands for Simulations in Physics.

We now redefine the `ParticleFree` class first introduced on page 24 by inheriting from `Particle` and assuming that the acceleration of the particle is due only to gravity.

```

package org.opensourcephysics.sip.ch3;

public class ParticleFree extends Particle {

    public ParticleFree(double _y, double _v) {

```

```

super(-y,-v);    // use constructor from super (Particle) class
a = -g;          // note choice of reference system
}

public void move(double dt) {
    y = y + v*dt + 0.5*a*dt*dt;
    v = v + a*dt;
}
}

```

Subclasses that define bodies for all abstract methods are said to be *concrete* manifestations of the abstract superclass.

The keyword `super` refers to a member of the superclass, which in this case is `Particle`. To ensure that the subclass is initialized correctly, we need to initialize the superclass first. By invoking the constructor from the superclass, we guarantee that we instantiate the member variables of the superclass. (If we do not explicitly make a call to the superclass constructor in the subclass constructor, Java would do so by default if the constructor has no arguments.)

To discuss the fall of the feather, we have to take into account the effects of drag resistance. Such effects are discussed in more detail in Chapter 5. For particles near the earth's surface, the direction of the retarding drag force due to air resistance is opposite to the motion of the particle. For a falling body the direction of the drag force F_d is upward, which we take to be positive, and the total force F on the particle can be written as

$$F = -mg + F_d, \quad (3.1)$$

where we have adopted a coordinate system where y increases upward, and thus $v < 0$ for a falling object. We know that the greater the speed of an object, the greater the magnitude of F_d . For simplicity, we will assume that $F_d/m = C|\mathbf{v}|$, that is, the drag force is proportional to the speed of the falling particle. For our choice of coordinate system and for a falling body, we have $|\mathbf{v}| = -v$, and the rate of change of the velocity becomes

$$\frac{dv}{dt} = -g - Cv. \quad (3.2)$$

Note that the falling object will eventually reach a terminal velocity v_t when $dv/dt = 0$. From (3.2) we see that $v_t = -g/C$ and we can express (3.2) in the convenient form (see (5.23a))

$$\frac{dv}{dt} = -g\left(1 - \frac{v}{v_t}\right), \quad (3.3)$$

with the initial condition $v(t = 0) = v_0$.

In Section 5.7 we will learn how to solve (3.2) on a computer using various numerical methods. However, (3.2) is sufficiently easy that we can guess its analytical solution. If you do not understand the following derivation, just look at the analytical solutions (3.7) and (3.8) which we will use to define the `move` method for a particle falling with drag resistance.

If the first term in (3.3) were absent, the derivative of the velocity would be proportional to the velocity, and the solution would be of the form $v(t) = Ae^{-\alpha t}$. So we guess that the general

form of the solution to (3.2) is

$$v(t) = Ae^{-\alpha t} + B. \quad (3.4)$$

It is easy to check that the general form (3.4) satisfies (3.3) and the initial condition if we choose A , B , and α appropriately. First we take the derivative of (3.4) and substitute it into (3.3):

$$\begin{aligned} \frac{dv(t)}{dt} &= -A\alpha e^{-\alpha t} = -g + \frac{gv(t)}{v_t} \\ &= -g + \frac{g}{v_t}(Ae^{-\alpha t} + B). \end{aligned} \quad (3.5)$$

Equation (3.5) can be satisfied for all values of the time t if

$$-A\alpha = \frac{g}{v_t}A \quad (3.6a)$$

$$0 = -g + \frac{g}{v_t}B. \quad (3.6b)$$

These two equations tell us that $\alpha = -g/v_t$ and $B = v_t$. The constant A is determined by the initial condition, $v(t = 0) = v_0$. Hence, we find that the general solution for $v(t)$ is

$$v(t) = v_t + (v_0 - v_t)e^{gt/v_t} = v_t + (v_0 - v_t)e^{-gt/|v_t|}. \quad (3.7)$$

If you are not very familiar with differential equations and their solutions, don't worry. We will learn in Section 5.2 how to solve differential equations such as (3.3) numerically using simple methods.

Exercise 3.2. Analytical solution

Fill in the missing steps in the derivation of (3.7) and show that $v(t)$ has the expected behavior at $t = 0$ and at $t \rightarrow \infty$. Also show that the solution to the differential equation $v(t) = dy/dt$ is

$$y(t) = y_0 + v_t t - \frac{v_t(v_0 - v_t)}{g}(1 - e^{gt/v_t}). \quad (3.8)$$

We now use the analytical solution to define a `ParticleDrag` class (a particle with drag resistance):

```
package org.opensourcephysics.sip.ch3;

public class ParticleDrag extends Particle {
    double vt;           // terminal velocity
    final double g = 9.8;

    public ParticleDrag(double _y, double _v, double _vt) {
        super(_y,_v);   // use constructor from super (Particle) class
        vt = _vt;
    }

    public void move(double dt) {
```

```

        y = y + vt*dt - vt*((v - vt)/g)*(1.0 - Math.exp(g*dt/vt));
        v = vt + (v - vt)*Math.exp(g*dt/vt);
    }
}

```

Note that the `move` method computes the velocity and the position at the end of the time step given their values at the beginning of the time step. For that reason, $v_0 \rightarrow v$, $y_0 \rightarrow y$, and $t \rightarrow \Delta t$ in going from (3.7) and (3.8) to the corresponding statements in the `move` method.

It is not possible to instantiate an object of type `Particle`, because `Particle` is an abstract class. Instead, we write a target class that instantiates particular members (objects) of `ParticleFree` and `ParticleDrag`.

```

package org.opensourcephysics.sip.ch3;

public class ParticleMoverApp {
    double minimumHeight, dt;

    public ParticleMoverApp(double _minimumHeight, double _dt) {
        minimumHeight = _minimumHeight;
        dt = _dt;
    }

    // return total time for particle to reach minimum height
    public double calculateTotalTime(Particle p) {
        double time = 0;
        while (p.y >= minimumHeight) {
            p.move(dt);
            time += dt;
        }
        return time;
    }

    public static void main(String[] args) {
        ParticleMoverApp particleMover = new ParticleMoverApp(0,0.01);
        Particle rock = new ParticleFree(20,0);
        double totalTime = particleMover.calculateTotalTime(rock);
        System.out.println("Time for rock to hit ground =" + totalTime);
        Particle feather = new ParticleDrag(20,0,-10);
        totalTime = particleMover.calculateTotalTime(feather);
        System.out.println("Time for feather to hit ground =" + totalTime);
    }
}

```

Note that the correct `move` method is automatically applied to each particle based on the subclass to which it belongs. This ability to choose the appropriate method depending on its subclass is known as *polymorphism*. (The mechanism that makes this choice automatic is called late or dynamical binding, which means that the appropriate connection or binding of the method to the correct object is not done until run time.) The advantage of this feature of object oriented programming is that it makes it very easy to modify our programs. For example, if we wanted to

introduce a particle whose drag resistance varies quadratically with the speed, we could define a new subclass and modify the `move` method (see Section 5.7).

The relation of `ParticleFree` and `ParticleDrag` is shown in the inheritance diagram of Figure 3.1. Note that the upcast can occur in a statement such as

```
Particle p = new ParticleFree(20,0);
```

The result is that a `ParticleFree` object is created and the resulting reference is assigned to a `Particle`. This assignment would seem to be an error because we have assigned one type to another, but such an assignment is correct because a `ParticleFree` is a `Particle` by inheritance. Suppose we later write

```
p.move(0.01);
```

Because of polymorphism and dynamic binding, the correct move method is used.

Note also that we did not instantiate a particle as a particular member of the `Particle` class, but instead instantiated particular kinds of particles. Analogously, we might define an `Animal` class and then subclass a `Dog`, `Cat`, and `Cow` from `Animal`.

3.2 Interfaces

An *interface* is a declaration of the type of functionality that an object is able to provide. Although an interface has a superficial similarity to a class containing all abstract methods, an interface is fundamentally different. Abstract classes work in conjunction with inheritance to define relationships. For example, a free particle is a type of particle. Why are interfaces useful? Interfaces add functionality to a class without having to worry about class inheritance. In other words, functionality can be added anywhere within the class hierarchy. In particular, a class can only inherit from one superclass, but it can *implement* many different interfaces.

Any object can implement any interface so long as the correct methods are defined. Suppose, that we define a Java class for dogs with subclasses for German Shepherds, Scotch Terriers, and other breeds. But many animals can be trained to fetch, and we want to be able to let the world know that some other object has this ability. So we are led to define the `Fetch` interface that includes the following method:

```
public interface Fetch {
    public void getBall();
}
```

However, we might have a very intelligent dog that can speak, and we could also define the `CanSpeak` interface:

```
public interface CanSit {
    public void speak();
}
```

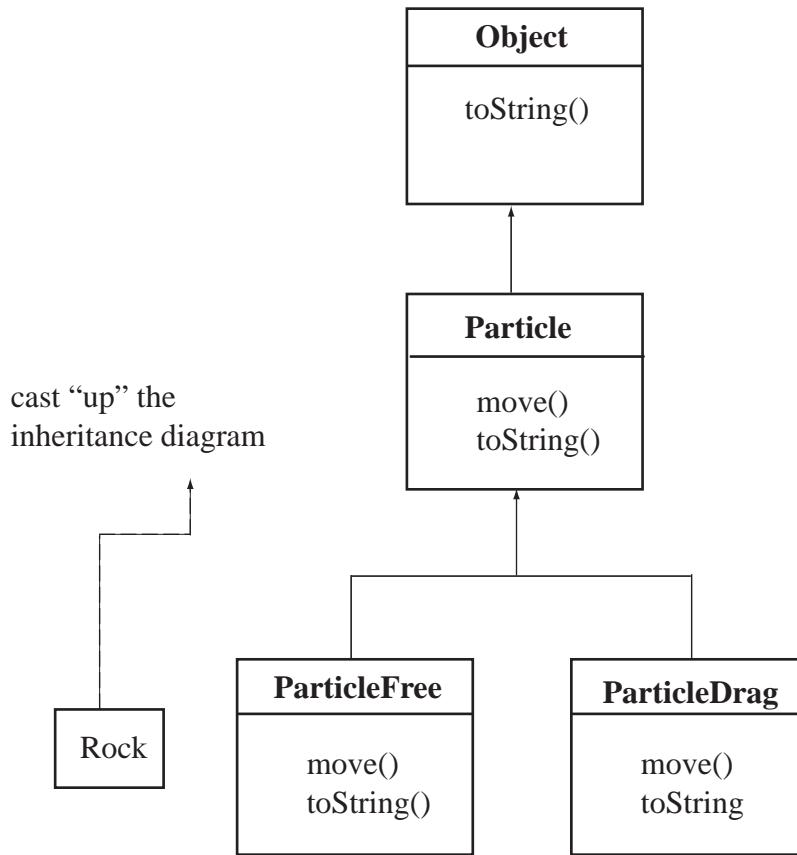


Figure 3.1: Inheritance diagram showing the relationships between **Particle**, **ParticleFree**, and **ParticleDrag**. The dotted line shows a particular instantiation of **ParticleFree**.

Any object, maybe even a **Horse**, can have one or both of these interfaces. All that is required is that the **Horse** know how to perform the required methods and that this ability be declared in the **Horse** definition using the `implements` keyword.

As a more relevant example, suppose that we wish to compute the numerical derivative of a function. Of course, we cannot compute numerical derivatives exactly on a digital computer because that would mean taking the limit of a difference Δx going to zero. So instead we will use what are called *finite differences*. If we look up derivatives in a calculus textbook, we would find the approximate relation

$$y' = f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}, \quad (3.9)$$

where y' is the approximate value of the derivative of $f(x)$ at x . A more accurate finite difference

relation for the first derivative is

$$y' = f'(x) \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2 \Delta x}. \quad (3.10)$$

Equation (3.10) is more accurate because the errors due to the nonzero value of Δx cancel each other when the function is evaluated on a symmetrical interval about x .

Problem 3.3. Numerical derivatives

Do a Taylor series expansion of $f(x)$ about x to show that (3.10) is correct to second-order in Δx , while (3.9) is correct only to first-order in Δx .

We next use the relation (3.10) and write a program without using interfaces that computes the derivative of the sin function.

Listing 3.1: The derivative of the sine function.

```
package org.opensourcephysics.sip.ch3;

public class Derivative1App {

    public static void main(String[] args) {
        int numberofPoints = 10;
        double xmin = 0;
        // compute derivative of sin(x) between x = 0 and x = 1
        double xmax = 0.5*Math.PI;
        double dx = (xmax - xmin)/numberofPoints;
        double x = xmin;
        for (int i = 0; i < numberofPoints; i++) {
            double derivative = (Math.sin(x+dx) - Math.sin(x-dx))/(2.0*dx);
            System.out.println("x = " + x + " derivative = " + derivative);
            x += dx;
        }
    }
}
```

Exercise 3.4. Numerical derivatives

- Use class `Derivative1App` to compute the derivative of $\sin(x)$ at various values of x . Vary the value of Δx and compare your numerical results to the exact value of the derivative at the same values of x .
- Modify the program so that it computes the numerical derivatives of the function $f(x) = (x^3 + \sin x)e^{-x^2}$ for x between 0 and 1.
- Modify the program so that it computes the the derivative of two different functions.

As you probably found in Exercise 3.4b, it is tedious to change Lisitng 3.1 so that it computes the derivative of a different function. So in the following we will define a function interface. Defining an interface is similar to defining a class. An interface definition begins with the keyword `interface`, rather than the keyword `class`, and the body of the interface lists the methods that

must be present for the interface to be implemented. However, the bodies of the methods are undefined. As for a class, an interface must be saved in file with the same name as the interface.

We can think of a *function* as a rule, f , that gives a well-defined output, y , corresponding to a well-defined input, x .

$$y = f(x), \quad (3.11)$$

for some range of x . The variable x is called the independent variable and variable y is called the dependent variable. If we restrict the input and output of $f(x)$ to real numbers, it is easy to define an interface that implements this idea:

```
public interface Function {
    public double evaluate(double x);
}
```

That's all to it in this case. Note that we have defined the signature of the method `evaluate`, but have not written the body of the method that would implement the proposed functionality.

Next we add a method to the `MyMath` class that computes the numerical derivative of an object of type `Function` using the relation (3.10):

```
public static double derivative(Function f, double x, double dx) {
    return (f.evaluate(x+dx) - f.evaluate(x-dx))/(2.0*dx);
}
```

Then we define class `MyFunction` that implements the `Function` interface and defines the desired function $(x^3 - \sin x)e^{-x^2}$.

```
package org.opensourcephysics.sip.ch3;
import org.opensourcephysics.numerics.Function;

public class MyFunction implements Function {
    public double evaluate(double x) {
        double f = (Math.pow(x,3) + Math.sin(x))*Math.exp(-x*x);
        return f;
    }
}
```

Finally we write `Derivative2App` to take advantage of the interface `Function`, method `derivative`, and class `MyFunction`:

Listing 3.2: Testing the derivative method.

```
package org.opensourcephysics.sip.ch3;
import org.opensourcephysics.numerics.Function;
import org.opensourcephysics.sip.extras.MyMath;

public class Derivative2App {

    public static void main(String[] args) {
        int numberofPoints = 10;
        double xmin = 0;
```

```

double xmax = 0.5*Math.PI;
double dx = (xmax - xmin)/numberOfPoints;
double x = xmin;
// note that an identifier can be defined by its interface
Function function = new MyFunction();
for (int i = 0; i < numberOfPoints; i++) {
    double derivative = MyMath.derivative(function,x,dx);
    System.out.println("x = " + x + " derivative = " + derivative);
    x += dx;
}
}
}

```

Exercise 3.5. Numerical derivatives

- Use class `Derivative2App` to compute the derivative of different functions at various values of x . What do you have to do? Vary the value of Δx and compare your numerical results to the exact value of the derivative at the same values of x .
- Define another class that implements `Function` and include this function in `Derivative2App`.

Exercise 3.6. Implementing the Function interface

Write a class that implements the `Function` interface and returns the value of the function $\sin x/x$ in the interval $-\pi \leq x \leq 1$.

We next look at another example of how the `Function` interface is useful. Suppose we want to compute the area under a function between two limits, a and b (the integral of the function). At this point, we do not know how to perform this operation, but suppose somebody else does. This person can write a method that computes this area, without knowing the functions for which we will want to consider. For example, the static method `computeArea(Function f, double a, double b)` in the `MyMath` class takes a `Function` and the two limits as parameters and returns the area. Note that we can use this method without knowing how it is implemented.

```

public class IntegralApp {
    public static void main(String[] args) {
        Function f = new SinFunction();
        double a = 0;
        double b = 1;
        double area = MyMath.computeArea(f, a, b);
        System.out.println("area = " + area);
    }
}

```

3.3 Special Functions

Special functions are a mainstay of computational physics, and we can now define classes that encapsulate these functions. A toy example that defines a first-order linear equation $f(x) = mx + b$ can be written as follows:

```
public class LinearFunction implements Function {
    double m,b;
    public LinearFunction(double _m, double _b) {
        m = _m;
        b = _b;
    }
    public double evaluate(double x) {
        return m*x + b;
    }
}
```

A constructor is often used to initialize variables and we have done so here. Linear functions can now be created in the usual way by using the **new** operator.

```
LinearFunction f1 = new LinearFunction(2,3);
LinearFunction f2 = new LinearFunction(-1,2);
```

The linear function works (that is, we can evaluate **f1** or **f2** or both), but we can make it better from an object oriented point of view. The coefficients *m* and *b* are encapsulated, but not really hidden. Any class within the package in which the function is defined can modify these values by statements such as

```
f1.m = 4;
f1.b = -1;
```

We can prohibit this type of (unintended) mischief by making the **LinearFunction** class *immutable*. This change can be accomplished by making defining the class to be **final** so that it cannot be subclassed and by defining its member variables to be **private**:

```
public final class LinearFunction implements Function {
    private final double m,b;
    public LinearFunction(double _m, double _b) {
        m = _m;
        b = _b;
    }
    public double evaluate(double x) {
        return m*x + b;
    }
}
```

3.4 Effective Java

The main reason for using interfaces is that a class can implement many different ones. When should we use an interface or an abstract class? In general, we will start from an abstract class

only if we have a well defined organizational hierarchy. However, the flexibility of being able to implement multiple interfaces usually means that interfaces are preferable. Only if you are forced to have concrete method definitions or member variables should you use a class hierarchy.

Exercise 3.7. Implementing a moveable interface

- a. Define a `Movable` interface that contains the methods `move` and `getY`.
- b. Change the `Particle` class so that it is not abstract and does not contain a `move` method. Change `ParticleFree` and `ParticleDrag` classes so that they implement the `Movable` interface. Test your new classes using the `ParticleMoverApp` program.

The major disadvantage to interfaces is that it is easier to change an abstract class than it is to change an interface. For example, if we add the non-abstract method `throw` to the `Particle` class, all existing subclasses will be able to perform this method. But if we add the `throw` method to the `Movable` interface, we must add this method to all classes that implement this interface. If other programmers use our `Particle` class, they will probably not notice if the new `throw` method exists. But if add `throw` to the `Movable` interface and don't tell anybody, any code that uses this interface will cease to work.

Inheritance is a powerful way to achieve code reuse, but it breaks encapsulation. A subclass all too often depends on the implementation details of its superclass if for no other reason than it invokes the superclass constructor. It is generally considered to be safe to use inheritance within a package where all classes are under the control of the same programmer. Carefully designed and documented class hierarchies, such as the user interface library distributed by Sun, can easily be subclassed.

3.5 Arrays

Ordered lists of data are most easily stored in arrays. For example, if we have an array variable named `x`, then we can access its first element as `x[0]`, its second element as `x[1]`, etc. All elements must be of the same data type, but they can be just about anything: primitive data types such as doubles or integers, objects, or even other arrays. The following statements show how array variables are defined and created:

```
double[] x;           // x defined to be an array of doubles
double x[];           // same meaning
x = new double[32];   // x array created of 32 elements
double[] y = new double[32]; // y array defined and created in one step
int [] num = new int[100]; // array of 100 integers
double[][] sigma = new double[3][3]; // 3 x 3 array of doubles
Particle [] p = new Particle[2];    // array of 2 Particle objects
```

We will adopt the style `double[] x` instead of `double x[]`, although both forms are acceptable.

Just as defining a variable of type `Particle` does not create that object, creating an array of objects does not create the objects. We must create each object using the `new` operator. For the `Particle[]` array example, we still have to create the elements of the array `p`:

```
p[0] = new ParticleFree(20.0,0);
p[1] = new ParticleDrag(10.0,0,5.0);
```

Note that the array index starts at zero and is always one less than the number of elements.

Arrays are treated as objects insofar as they are created with the `new` operator and have a `length` property that can be accessed using dot notation. For example, we can create an array named `square` and assign each element to the square of its index using the following code:

```
int numberOfElements = 32;
double square = new double[numberOfElements]; // create array
for (int i = 0, n = square.length; i < n; i++)
    square[i] = i*i;
```

If computational speed is important, it is usually more efficient not to determine the size or length of an array dynamically using the `length` method.

Exercise 3.8. Freely falling body

Modify `ParticleFreeApp` so that you can consider any number of particles with different initial conditions. Treat the freely falling particles as an array.

3.6 Summary

As in Chapter 2, we have covered much more Java syntax and object oriented programming concepts. However, the good news is that the core syntax and concepts that we have discussed in Chapters 2 and 3 are sufficient to write programs that are roughly equivalent to what you might write using procedural languages like Fortran or C. Of course, there is more to learn about programming, but not much compared to what we have already covered.

In Chapter 4, we will introduce Java syntax associated with graphical input and output. Unless you are already familiar with Java, it will probably be difficult for you to understand all the syntax that we will introduce there in one reading. However, it doesn't really matter because we have written a package of classes that make graphical input and output much easier. We have already used the `Math` class written by programmers at Sun and didn't think much about it. One of the advantages of object oriented programming is that we only have to understand how a class works, not how it is written. So if you still feel a bit shaky about writing your own Java programs, relax. It will be downhill from now on.

References and Suggestions for Further Reading

Joshua Bloch, *Effective Java*, Addison Wesley (2001).

Stephen J. Chapman, *Java for Engineers and Scientists*, Prentice-Hall (2000).

Bruce Eckel, *Thinking in Java*, second edition, Prentice-Hall (2000).

David Flanagan, *Java in a Nutshell*, third edition, O'Reilly (2000).

Brian R. Overland, *Java in Plain English*, third edition, M&T Books (2001).

Walter Savitch, *Java: An Introduction to Computer Science and Programming*, second edition, Prentice-Hall (2001).

Chapter 4

Input and Output: the Open Source Physics Library

©2002 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
16 February 2002

We introduce the Open Source Physics library to make graphical input and output easier and give some simple examples of the `Calculation` interface.

4.1 Graphical Input

So far we have “hard coded” the values of various parameters in our programs. For example, if we wish to change the values of `xmin` and `xmax` in Listing 3.2, we have to edit the `Derivative2App` program, recompile it, and then run it again. It would be much easier if we could input the data from the console (command line) and even better if we could use a graphical interface. The details of input and output are the most demanding part of the syntax in any programming language. Although languages such as Fortran and C have only console input and output, our experience is that once we have written a program that has the desired format for input and output, we simply copy and modify this format when we write a new program. The details of the syntax are not easy to remember.

Input and output in Java is both easier and more difficult than it is in traditional languages. The difficulty arises from the greater power of Java which allows us to input data graphically. Greater power means more syntax. In the following, we will introduce new vocabulary and syntax associated with `Container`, `JFrame`, `JButton`, and the `ActionListener` interface among many other terms. The new classes and interfaces will be confusing at first. But Java is also easier because the object oriented nature of Java allows us to “hide” much of this syntax from the user.

We have hidden most of the syntax associated with graphical input and output in the *Open Source Physics* (`OSP`) library or package of classes and interfaces. Our approach will be to first learn how to use this package to do graphical input (Section 4.4), plots (Section 4.6) and then

animations and visualizations (Chapter 6) and gradually learn how this package was constructed. This package is open source and you are encouraged to look at it and even change it. We suggest that you read the following to get an idea of what is involved in building a graphical interface, and then skip to pages 57 and 60 where we give examples of how to use the OSP package.

4.2 An Example

One of the great attractions of Java is that device and platform independent graphics is built directly into the language. Lines, circles, rectangles, images, and text can be drawn with just a few statements. However, creating even a simple graph can require a fair amount of programming. A scale needs to be established, axes need to be drawn, and data needs to be transformed. An additional complication arises due to the fact that a graph must be able to redraw itself whenever a window is covered or resized. But plotting is a fairly routine task. First we show a simple example that uses a `Drawable` interface and the `DrawingPanel` class that are part of the OSP library.

The `Drawable` interface is defined in the `org.opensourcephysics.display` class. It contains a single method, `draw`, that is invoked automatically from a method of the `DrawingPanel` class.

```
public interface Drawable {
    public void draw(DrawingPanel drawingPanel, Graphics g);
}
```

Drawable objects, that is, objects that implement the `Drawable` interface, are instantiated and then added to a drawing panel where the objects will draw themselves in the order that they are added. Listing 4.1 displays a program that creates a drawing panel containing a circle and an arrow.

Listing 4.1: A circle and an arrow in a drawing panel.

```
package org.opensourcephysics.sip.ch4;
import org.opensourcephysics.display.*;
import javax.swing.JFrame;

public class DrawablesApp {

    public static void main(String[] args) {
        // create a drawing frame and a drawing panel
        DrawingPanel panel = new DrawingPanel();
        DrawingFrame frame = new DrawingFrame(panel);
        panel.setSquareAspect(false);
        // create a circle and an arrow
        Circle circle = new Circle(0, 0);
        panel.addDrawable(circle);
        Arrow arrow = new Arrow(0, 0, 4, 3);
        panel.addDrawable(arrow);
        frame.show();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

The frame used in the example, `DrawingFrame`, is a subclass of `JFrame`, which we will discuss in Section 4.3.

The `OSP` package also defines `PlottingPanel` as a subclass of `DrawingPanel`. This class contains the necessary axes and titles to produce linear, log-log, and semilog graphs. For example, Listing 4.2 plots the period T versus the semimajor axis a of the planets in the Sun's solar system (see Chapter 8). The data arrays, `a` and `T`, contain the semimajor axis of the planets in astronomical units (AU) and the period in years, respectively. Note that the plot automatically adjusts itself to fit the data because the `autoscale` parameter is set to true for both axes.

Listing 4.2: Program to plot the semimajor axis versus the period.

```
package org.opensourcephysics.sip.ch4;
import org.opensourcephysics.display.*;
import java.awt.event.*;
import java.awt.*;
public class PlotDataApp {

    public static void main(String[] args) {

        PlottingPanel plotPanel = new PlottingPanel("a", "T", "Kepler's Second Law");
        DrawingFrame drawingFrame = new DrawingFrame(plotPanel);
        Dataset dataset = new Dataset();
        double[] a = {0.241, 0.615, 1.0, 1.88, 11.86, 29.50, 84.0, 165, 248};
        double[] period = {0.387, 0.723, 1.0, 1.523, 5.202, 9.539, 19.18, 30.06, 39.44};
        dataset.setConnected(false);
        dataset.append(a, period);
        plotPanel.addDrawable(dataset);
        plotPanel.setAutoscaleX(true);
        plotPanel.setAutoscaleY(true);
        plotPanel.repaint();
        drawingFrame.show();
        drawingFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Exercise 4.1. Log-log plots

- Modify `PlotDataApp` so that it plots $\log T$ versus $\log a$. What can you conclude from the plot (see Section 8.5)? Another way of making a log-log plot is to replace the statement `Dataset dataset = new Dataset()` by `LogDataset dataset = new LogDataset(Axis.LOG10, Axis.LOG10)`?
- What happens if you replace `dataset.setConnected(false)` by `dataset.setConnected(true)`?
- What happens if you add the statement `plotPanel.setSquareAspect(false)`? What happens when the aspect ratio is true?

4.3 Graphical Input

We now discuss in brief some of the considerations that go into developing a graphical interface. Our goal is not to give a thorough introduction to graphics in Java, but to give you a general idea of what is involved.

4.3.1 Components

Our first task is to create a window. An object of the `JFrame` class appears on the screen as a window with a border, a title, and icons for closing and resizing the window. The following example creates a window and displays it on the screen.

Listing 4.3: An example of a window using `JFrame`.

```
package org.opensourcephysics.sip.ch4;
import javax.swing.JFrame;

public class FrameApp {
    public static void main(String[] args) {
        // create a JFrame with title 'Frame Example'
        JFrame f = new JFrame("Frame Example");
        f.setSize(400,400); // set size of frame in pixels
        f.setVisible(true); // display frame on screen
        // exit when frame is closed
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Note that `JFrame` has a number of methods associated with it.

Java has an extensive library of standard packages that do input and output, image handling, sound, among many other essential functions. These packages are part of the Java Application Programming Interface (API). The statement,

```
import javax.swing.*;
```

means that we are using a package of graphical user interface (GUI) classes known as “Swing.” (We have implicitly been accessing the classes of `java.lang` which is imported by default.)

Exercise 4.2. Opening a window

Run `FrameApp` and notice that a `JFrame` can be resized. What happens when you close the `JFrame`? Remove the last statement of the program. Now what happens when you close the `JFrame`?

A `JFrame` is an example of a *component*, which is a graphical object that can be displayed on the screen. We will soon introduce other examples of components including buttons, text labels, and text boxes. A component is any object of a class that is a subclass of the abstract class `Component`. A `Container`, which inherits from `Component`, can contain other Components. Figure 4.1 shows the hierarchy of some of the important graphics classes. The important thing to remember is that all graphical objects are derived from `Component`.

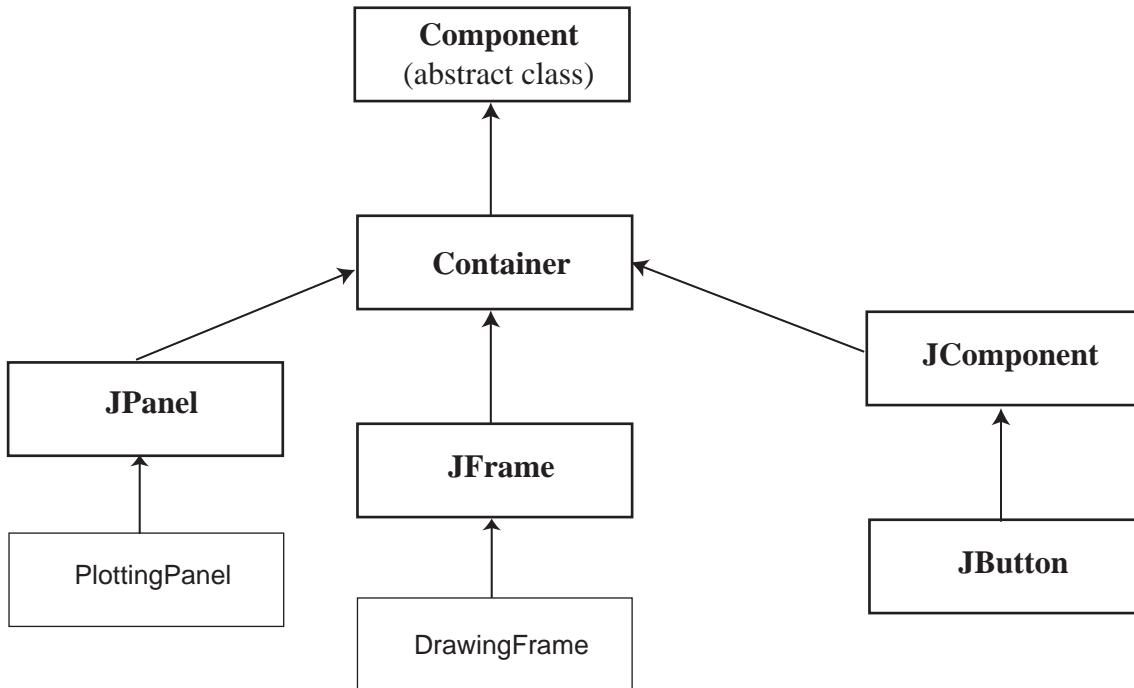


Figure 4.1: A part of the graphics class inheritance hierarchy. Classes that begin with `J` are part of Swing. `DrawingFrame` and `PlottingPanel` are part of the OSP library.

Now that we have created a main window using `JFrame`, how do we add other components such as buttons and text fields? Although a `JFrame` appears on the screen as a simple window, it is composed of multiple layers, each responsible for a different function. The layer of the `JFrame` that holds other components that are visible to the user is known as the `content pane`. The following code creates a frame and a button, retrieves the frame's content pane, and adds the button to the content pane:

```

JFrame frame = new JFrame();
Container c = frame.getContentPane();           // get content pane layer of JFrame
JButton button = new JButton("Calculate");
c.add(button,BorderLayout.CENTER);
  
```

Add this code (and any necessary import statements) to class `FrameApp` and see what happens.

The method `getContentPane()` returns the content pane of `frame`, which is an object of type `Container`. When we add a button or text field to the content pane, what determines where the text field will be placed within the container and the size of the text field? These properties are determined by the content pane's layout manager. By default the content pane uses a `BorderLayout` that places the objects it contains in one of five positions: `NORTH`, `SOUTH`, `CENTER`, `EAST`, or `WEST`. The `add` method adds a component such as a button to a container. The form of

the `add` method depends on the layout manager used by the container. For example, if we use the `FlowLayout` manager, no position information is needed in the `add` method.

In Listing 4.4 we construct a label, a text field, and a button for inputting `xmin`, `xmax`, and `dx` for the calculation of the derivative considered in Section 3.2.

Listing 4.4: First example of graphical input.

```
package org.opensourcephysics.sip.ch4;
import java.awt.Container; // use abstract window (awt) toolkit
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JLabel;

public class GUITest1App extends JFrame {
    protected JButton calculateButton;
    protected JTextField xminTextField;
    protected JTextField xmaxTextField;
    protected JTextField numberPointsTextField;

    public GUITest1App() {
        super("First example of graphical input");
        JLabel xminLabel = new JLabel("xmin = ");
        xminTextField = new JTextField(20); // 20 is number of columns
        JLabel xmaxLabel = new JLabel("xmax = ");
        xmaxTextField = new JTextField(20);
        JLabel numberPointsLabel = new JLabel("number of points = ");
        numberPointsTextField = new JTextField(20);
        calculateButton = new JButton("Calculate");
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        c.add(xminLabel);
        c.add(xminTextField);
        c.add(xmaxLabel);
        c.add(xmaxTextField);
        c.add(numberPointsLabel);
        c.add(numberPointsTextField);
        c.add(calculateButton);
        setSize(300,200);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        GUITest1App app = new GUITest1App();
    }
}
```

In this example we imported only the classes of `java.awt` and `javax.swing` that we needed. However, if we use many classes from the same package, it is more convenient to import the whole package by using an asterisk (*) in place of a specific class name. For example, to import all the classes in the `awt` package, we write

```
import java.awt.*;
```

Swing is built on top of the Abstract Windowing Toolkit (awt) package, which is the reason that we usually have to import classes from `java.awt` as well as `javax.swing`.

Exercise 4.3. Textfields and buttons

Run `GUITest1App` and explain the function of the button, textfields, and textlabels. What happens when you click on the button?

4.3.2 Events

When the user clicks the button displayed in `GUITest1App`, it triggers an *event*. Events are typically triggered by mouse clicks and keyboard input. A GUI-based program must be able to respond to user-initiated events at any time and is said to be *event driven*.

An *event* is initiated by the user and invokes methods in objects that have registered their interest. To handle events an object must have methods that know what to do once an event is received. In `GUITest1App`, nothing happened when the button is clicked because no object registered an interest in receiving events for the button. Objects that register their interest in receiving events are known as *event listeners*, and must implement the appropriate listener interface for the event. An object is notified when a button is clicked by implementing the `ActionListener` interface and registering this object as an action listener for the button by using the `addActionListener` method. When a user clicks on the button, the `actionPerformed` method is invoked for the ActionListeners that registered themselves for the particular button. In Listing 4.5, we extend `GUITest1App` to create a graphical user interface that performs a calculation when the button is clicked.

Listing 4.5: Example of the implementation of the `ActionListener` interface.

```
package org.opensourcephysics.sip.ch4;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import org.opensourcephysics.numerics.Function;
import org.opensourcephysics.sip.extras.MyMath;
import org.opensourcephysics.sip.ch3.MyFunction;

public class GUITest2App extends GUITest1App implements ActionListener {

    public GUITest2App() {
        super();                                // use constructor of GUITest1App
        calculateButton.addActionListener(this);   // note additional statement
    }

    public void actionPerformed(ActionEvent e) { // need to add method
        String s = xmaxTextField.getText();
        double xmax = Double.parseDouble(s); // convert String to double
    }
}
```

```

s = xminTextField.getText();
double xmin = Double.parseDouble(s);
s = numberofPointsTextField.getText();
int numberofPoints = Integer.parseInt(s);
double dx = (xmax - xmin)/numberofPoints;
double x = xmin;
Function function = new MyFunction();
for (int i = 0; i < numberofPoints; i++) {
    double derivative = MyMath.derivative(function,x,dx);
    System.out.println("x = " + x + " derivative = " + derivative);
    x += dx;
}
}

public static void main(String[] args) {
    GUITest2App app = new GUITest2App();
}
}
}

```

To emphasize the necessary changes, we have used inheritance to define the class `GUITest2App`. In this case it would have been just as easy to copy and paste. Note the inclusion of the `actionPerformed` method.

The method `calculateButton.addActionListener` needs to know the specified action listener that is to receive events from `calculateButton`. Recall that the keyword `this` always refers to the current object, which is the object within which the reference appears. Hence, the argument `this` in the `calculateButton.addActionListener(this)` statement notifies `GUITest2App`, which implements `ActionListener`.

4.4 Using the OSP Package for Graphical Input

We have now learned how to construct a program that allows us to input parameters graphically without having to recompile our program. We could continue to build a user interface that is appropriate for each program. For example, if we want to have graphical input for `ParticleMoverApp`, we could build a user interface with text fields corresponding to `dt` and `minimumHeight` and a Calculate button. However, this process is time consuming and not very object oriented because we would be doing more or less the same thing over and over again. For these reasons we have developed the `CalculationControl` class and the `Calculation` interface as part of the `OSP` package to allow us to easily define the input parameters. To see how to use the `Calculation` interface, look at the following example:

Listing 4.6: Example of the use of the `Calculation` interface.

```

package org.opensourcephysics.sip.ch4;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.sip.ch3.MyFunction;
import org.opensourcephysics.numerics.Function;
import org.opensourcephysics.sip.extras.MyMath;

```

```

public class Derivative3App implements Calculation {
    protected Control myControl;
    protected double xmin, xmax, dx;
    protected int numberofPoints;
    protected Function function = new MyFunction();

    public void calculate() {
        xmax = myControl.getDouble("xmax");
        xmin = myControl.getDouble("xmin");
        numberofPoints = myControl.getInt("numberofPoints");
        calculateDerivative ();
    }

    public void resetCalculation() {
        myControl.setValue("xmin",0); // default values
        myControl.setValue("xmax",1);
        myControl.setValue("numberofPoints",10);
    }

    public void calculateDerivative() {
        double dx = (xmax - xmin)/numberofPoints;
        double x = xmin;
        for (int i = 0; i < numberofPoints; i++) {
            double derivative = MyMath.derivative(function,x,dx);
            System.out.println("x = " + x + " derivative = " + derivative);
            x += dx;
        }
    }

    // method used to give Calculation a reference to Control object
    public void setControl(Control c) {
        myControl = c;
        resetCalculation ();
    }

    public static void main(String[] args) {
        Derivative3App app = new Derivative3App();
        CalculationControl c = new CalculationControl(app);
        app.setControl(c);
    }
}

```

Run `Derivative3App` to see how it works.

`CalculationControl` implements the `Control` interface and is designed to control any calculation. `CalculationControl` has a text area where the user can enter parameters, a message area for output, and two buttons: calculate and reset. In our first example, the calculation simply prints the results. In Section 4.6 we will learn how to add graphical output.

Any class that implements `Calculation` must have a common set of methods that the control is guaranteed to find. These methods are listed in the code for the `Calculation` interface.

```
package org.opensourcephysics.templates;

public interface Calculation {
    public void calculate();
    public void resetCalculation();
    public void setControl(Control control);
}
```

When the calculate or reset buttons are clicked, the `CalculationControl` invokes a method with the corresponding name in a `Calculation` object. For example, when a user clicks the calculate button, the `CalculationControl` invokes the `calculate` method in the `Calculation`. The `calculate` method reads new values from the control, does the calculation, and displays the result. The `resetCalculation` method sets the default values of the input parameters for the calculation.

The important methods in the `Control` interface are listed in the following. The complete listing is given in the `OSP controls` package.

Listing 4.7: Important methods in the `Control` interface.

```
package org.opensourcephysics.templates;

public interface Control {
    // store name and value in control object
    public void setValue(String name, int val);
    public void setValue(String name, double val);
    public void setValue(String name, boolean val);
    public void setValue(String name, Object val);

    public int getInt(String name);      // get value from control object
    public double getDouble(String name);
    public String getString(String name);
    public boolean getBoolean(String name);

    public void println(String s);      // print string in control's message area
}
```

The `main` method in Listing 4.6 creates an instance of `Derivative3App` named `app`, which is the object that performs the calculation. In our example, the calculation merely print messages. The second statement creates the graphical user interface, an `Control` object named `c`. The `Control` constructor registers the `app` with the control so that the control can invoke the calculation's methods when the buttons are clicked. Statement registers the control with the simulation. This action enables `app` to store and retrieve parameters from the control. The `main` method has done its job by creating two objects—the control object and the calculation object—and registering each object with the other. In this way we have allowed the possibility of changing the implementation of the `Control` interface without requiring any changes in the above code.

After the `main` method finishes its work, the program is running and the control is visible on the screen. The program will be terminated when the control's `JFrame` is closed.

4.5 Using the OSP Package to Plot a Function

In Appendix 4A we discuss additional graphics classes and their methods that allow us to make simple plots. There is a lot to digest and unless you have prior experience with Java, it is unlikely that you will be able to follow all the details. However, it is not necessary to understand these details because the `OSP` package “hides” them. We encourage you to look at the source code for this package as you gain more experience with Java.

We first distinguish between two levels of plotting. In the “quick and dirty” mode, we want to plot the results produced by our program to help check the program and to gain a preliminary understanding of how the system behaves. In the “presentation” mode, we want to do a careful graphical analysis of the results and make plots that we can present to others. In the latter mode we want to optimize the range of each axis, label the axes, have tick marks at convenient spacings, etc. For this second mode we recommend that you export your data to a separate program that is optimized for making graphs and fits to data. There are many open source, shareware, and commercial programs with these capabilities. These programs allow you to manipulate the data in many ways, do simple curve fitting, and to set up plots in many different formats.

We have developed the `OSP` display package to make plots directly from Java. The core functionality of this library is similar to more sophisticated graphics packages and provides a framework that not only plots (x, y) data sets, but that can also be used for animations and other visualizations. In this chapter we use it only to draw graphs.

As an example, we use the `Calculation` interface and `CalculationControl` class to plot the derivative of the function $(x^3 - \sin x)e^{-x^2}$ that we computed in `Derivative3App`. We also use the `PlottingPanel` and `DrawingFrame` and `Dataset` classes that are part of the `OSP` package.

Listing 4.8: Example of a simple plot using the `Calculation` interface.

```
package org.opensourcephysics.sip.ch4;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.sip.ch3.MyFunction;
import org.opensourcephysics.numerics.Function;
import org.opensourcephysics.sip.extras.MyMath;
// explicitly include classes to emphasize ones that are used for the first time
import org.opensourcephysics.display.PlottingPanel;
import org.opensourcephysics.display.DrawingFrame;
import org.opensourcephysics.display.DatasetCollection;
import org.opensourcephysics.display.Dataset;

public class PlotExampleApp implements Calculation {
    Control myControl;
    double xmin, xmax;
    int numberofPoints;
    Function function = new MyFunction();
    PlottingPanel plottingPanel;
    DrawingFrame drawingFrame;
    DatasetCollection datasetCollection;
    int datasetIndex;

    public PlotExampleApp() {
```

```

plottingPanel = new PlottingPanel("x", "y", "Derivative");
drawingFrame = new DrawingFrame(plottingPanel);
datasetCollection = new DatasetCollection();
plottingPanel.addDrawable(datasetCollection);
}

public void calculate() {
    xmax = myControl.getDouble("xmax");
    xmin = myControl.getDouble("xmin");
    numberOFPtions = myControl.getInt("number of points");
    double dx = (xmax - xmin)/(numberOFPtions - 1);
    double x = xmin;
    datasetIndex++;
    Dataset d = new Dataset();
    plottingPanel.addDrawable(d);
    for (int i = 0; i < numberOFPtions; i++) {
        double derivative = MyMath.derivative(function,x,dx);
        datasetCollection.append(datasetIndex, x, derivative );
        x += dx;
    }
    plottingPanel.repaint ();
}

public void resetCalculation() {
    xmax = 1;
    xmin = 0;
    numberOFPtions = 10;
    datasetIndex = -1;
    datasetCollection.clear ();
    plottingPanel.repaint ();
}

// give Calculation a reference to Control object
public void setControl(Control c) {
    myControl = c;
    resetCalculation ();
}

public static void main(String[] args) {
    PlotExampleApp app = new PlotExampleApp();
    CalculationControl c = new CalculationControl(app);
    app.setControl(c);
}
}

```

Note that the new graphics statements are

```

plottingPanel = new PlottingPanel("x", "y", "Derivative");
drawingFrame = new DrawingFrame(plottingPanel);

```

PlottingPanel is a subclass of **DrawingPanel** and contains *x* and *y* axis labels, a title, and axes

to produce linear, log-log, and semilog plots.

We store the x and y values that we want to plot in a `Dataset`, which contains an array of x and y points (see Appendix 4A). Because we might want to make multiple plots with different colors, we can add each data set to a `datasetCollection`.

We will use `PlotExampleApp` as a template for the programs in Chapter 5, where we consider again the motion of a particle in one and two dimensions.

In Table 4.1 we summarize the most common methods in the `OSP` display classes. The online documentation at <http://www.opensourcephysics.org/> has a complete listing and more extensive descriptions.

Dataset methods

<code>append</code>	Add data to the data set as x, y coordinate pairs. The data can be either doubles or arrays of doubles.
<code>clear</code>	Remove all data.
<code>setConnected</code>	Connect the data points with straight lines. Default is <code>false</code> .
<code>setLineColor</code>	Set the color of the lines connecting the points. Ignored if points are not connected. Default is <code>black</code> .
<code>setMarkerColor</code>	Set the color of the data markers. Ignored if the data marker is set to <code>NO_MARKER</code> . Default is <code>black</code> .
<code>setMarkerShape</code>	Set the shape of the data markers. Valid shapes are <code>NO_MARKER</code> , <code>CIRCLE</code> , <code>SQUARE</code> , <code>FILLED_CIRCLE</code> , and <code>FILLED_SQUARE</code> . The default is <code>SQUARE</code> .

DrawingPanel methods

<code>addDrawable</code>	Add a drawable object to the panel. Objects will be drawn in the order that they are added.
<code>setAutoscaleX</code>	Autoscale the x axis using max and min values from the data. Default is <code>true</code> .
<code>setAutoscaleY</code>	Autoscale the y axis using max and min values from the data. Default is <code>true</code> .
<code>repaint</code>	Repaint all drawable objects. The repaint operation is done at the convenience of the operating system.
<code>removeAll</code>	Clear the panel by removing all the drawable objects.
<code>removeDrawable</code>	Remove one drawable object.
<code>setSquareAspect</code>	Set x and y scale to have equal pixels per unit.
<code>setPreferredMinMaxX</code>	Set scale on x axis (set autoscale to <code>false</code>).
<code>setPreferredMinMaxY</code>	Set scale on y axis.
<code>setPreferredMinMax</code>	Set scale on x and y axes.

Table 4.1: Some of the important methods for making plots.

Exercise 4.4. Plotting options

- Use the `OSP` library to write a simple program that plots the sin function between 0 and 2π . What is the minimum number of points that are necessary in order to recognize the sine curve?

- b. Change the marker style to CIRCLE and set the connected points option to false.

Exercise 4.5. Plotting two curves

Create a second data set in your program and use this data set to add a cosine curve to the plot.

```
DatasetCollection dataset2 = new DatasetCollection();
```

Plot the sine and a cosine curves simultaneously.

Exercise 4.6. Test of numerical derivatives

- a. Modify `PlotExampleApp` so that the function and its derivative can be plotted.
- b. Add the boolean variable, `derivativeMode` and read and store this variables from the control. Boolean variables can be read using the `getBoolean` method. Plot the derivative if `derivativeMode` is true. Test your code by taking derivatives of some analytical functions.
- c. Choose a function whose derivative can be calculated in terms of known functions. Plot the analytical and numerical derivatives on the same graph. One way to do so is to plot the (analytical) derivative (a function) with `derivativeMode` set true and then to plot the numerical result with `derivativeMode` set false. How do the two curves compare? What happens when you change Δx ?
- d. Plot the function $\sin(1/x)$ and its numerical derivative on the interval $(-1, 1)$ using 10000 points. What happens to the plot of the derivative as Δx takes on the values 0.1, 0.01, 0.001, and 0.0001? Why?

4.6 Model-View-Control

Experienced programmers approach a programming project not as a coding task but as a design process. They look for data structures and behaviors that are common to other problems they have solved. Separating the physics from the user interface and the data visualization is one such approach. In keeping with computer science terminology, we refer to the user interface as the *Control*. It is responsible for handling events and passing them on to other objects. The plots that we have constructed presents a visual representation of the data, and is an example of a *View*. By using this design strategy it is possible to have multiple views of the same data. For example, we could show a plot and a table view of the same data. Finally, the physics can be thought of as a *Model* that maintains the data and provides the methods by which that data can change.

The Model-View-Control (MVC) design pattern that we have used for `PlotExampleApp` is one of the most successful software architectures ever devised. It is the basis for the smallTalk programming language and was used extensively in designing the Java Swing components. Java is an excellent language with which to implement the MVC pattern. Java allows us to isolate the model, the control, and the view in separate classes, and makes it easy to reuse these classes in various projects. And Java makes it easy to add new features to a class without having to change the existing code. These features are known as encapsulation, code reuse, and polymorphism, and are among the hallmarks of object oriented programming.

Appendix A: Plotting a Function

We have seen that Java provides many components such as buttons, text fields, and labels and have learned how to use the `OSP` package to create buttons and text labels for graphical input. However, suppose we want to plot a function. No component is provided that is designed to display such an object. The solution is to do our own painting by subclassing a `JPanel`. In this Appendix we examine various Java classes and components by building a simple version of the `OSP` drawing package that allows us to draw a red square on the screen and make a very simple plot.

A `JPanel` is a blank component that has a surface on which we can draw. The painting code goes in the method `paintComponent`, which we override from `JComponent`, the parent class of `JPanel`. Before we do any painting, we invoke `super.paintComponent`, which paints the background of the panel. The `Graphics` object that is passed as a parameter to the `paintComponent` method is used to draw the output. The following example shows how to draw a filled red square on a panel.

Listing 4.9: Red square on a JPanel.

```
package org.opensourcephysics.sip.ch4;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.Container;
import javax.swing.JPanel;
import javax.swing.JFrame;

// draw filled red square on a JPanel
public class SquarePanelApp extends JPanel {

    public void paintComponent(Graphics g) {
        super.paintComponent(g); // paint background
        // set color to red, all subsequent operations on graphics object g use this color
        g.setColor(Color.red);
        int rpix = 30; // width of square in pixels
        int xpix = 50; // center x position of square
        int ypix = 50; // center y position of square
        // draw filled square on panel
        g.fillRect (xpix - rpix,ypix - rpix,2*rpix,2*rpix);
    }

    public static void main(String[] args) {
        SquarePanelApp squarePanel = new SquarePanelApp();
        JFrame frame = new JFrame();
        Container c = frame.getContentPane();
        c.add(squarePanel, BorderLayout.CENTER);
        frame.setSize(400,400);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

To draw a plot, we need to store the values that we want to plot. We will store these values in two arrays. Our first attempt at a dataset class is given in Listing 4.10.

Listing 4.10: Dataset class.

```
package org.opensourcephysics.sip.ch4;

public class Dataset {
    protected double[] xpoints; // array of x points
    protected double[] ypoints; // array of y points
    protected int index; // current index of arrays

    public Dataset(int numberOfPoints) {
        xpoints = new double[numberOfPoints];
        ypoints = new double[numberOfPoints];
        index = 0;
    }

    public void append(double x, double y) {
        xpoints[index] = x;
        ypoints[index] = y;
        index++;
    }
}
```

One limitation of a `JPanel` is that it uses a pixel-based coordinate system with its origin at the top left of the panel; its *y* value increases as we move down the panel. These coordinates are inconvenient, so we need to create methods that transform screen (pixel) coordinates to a more natural coordinate system known as *world coordinates*. We construct a `DatasetPanel` to draw our dataset.

Listing 4.11: DatasetPanel.

```
package org.opensourcephysics.sip.ch4;
import java.awt.Dimension;
import java.awt.Graphics;
import javax.swing.JPanel;

public class DatasetPanel extends JPanel { // draw one dataset
    // Dimension object has width and height
    Dimension size; // size of panel in pixels
    double xmin, ymin, xmax, ymax; // world coordinates
    Dataset dataset; // dataset to be drawn on panel
    int markerSize = 4; // size in pixels of marker in dataset

    public DatasetPanel (Dataset _dataset) {
        dataset = _dataset;
    }

    public void paintComponent (Graphics g) {
        super.paintComponent(g); // paint background
```

```

size = getSize();           // get size of panel, method inherited from Component
for (int i = 0; i < dataset.index; i++) {
    // convert world coordinates of dataset to pixels
    int xpix = xToPix(dataset.xpoints[i]);
    int ypix = yToPix(dataset.ypoints[i]);
    g.fillRect (xpix - markerSize/2, ypix - markerSize/2, markerSize, markerSize);
}
}

// set min and max values in world coordinates that will appear on panel
public void setXYMinMax (double xmin, double xmax, double ymin, double ymax) {
    xmin = _xmin;
    xmax = _xmax;
    ymin = _ymin;
    ymax = _ymax;
}

// convert x from world units to pixel units
public int xToPix (double x) {
    double ratio = (x - xmin)/(xmax - xmin);
    double pix = ratio*size.width;
    if (pix > Integer.MAX_VALUE) {
        return Integer.MAX_VALUE;
    }
    if (pix < Integer.MIN_VALUE) {
        return Integer.MIN_VALUE;
    }
    return (int)Math.floor(pix);
}

// convert y from world units to pixel units
public int yToPix (double y) {
    double ratio = (y - ymin)/(ymax - ymin);
    // pix is defined differently from above because y pixel origin is at top of panel
    double pix = size.height - ratio*size.height;
    if (pix > Integer.MAX_VALUE) {
        return Integer.MAX_VALUE;
    }
    if (pix < Integer.MIN_VALUE) {
        return Integer.MIN_VALUE;
    }
    return (int)Math.floor(pix);
}
}

```

We now test our `Dataset` and `DatasetPanel`, by drawing the function, $(x^3 - \sin x)e^{-x^2}$, that we defined on page 42.

Listing 4.12: Plot of test function.

```
package org.opensourcephysics.sip.ch4;
```

```

import java.awt.BorderLayout;
import javax.swing.JFrame;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.sip.ch3.MyFunction;
import org.opensourcephysics.numerics.Function;
import org.opensourcephysics.sip.extras.MyMath;

// plot function
public class DatasetPanelTestApp {
    public static void main (String[] args) {
        Function function = new MyFunction();
        double xmin = 0;
        double xmax = Math.PI;
        int numberofPoints = 10;
        double dx = (xmax - xmin)/(numberofPoints-1);
        Dataset dataset = new Dataset(numberofPoints);
        double x = xmin;
        for (int i = 0; i < numberofPoints; i++) {
            dataset.append(x, function.evaluate(x));
            x += dx;
        }
        JFrame frame = new JFrame("DatasetPanel Test");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        DatasetPanel panel = new DatasetPanel(dataset);
        panel.setXYMinMax(-1, Math.PI+1, -2, 2);
        frame.getContentPane().add(panel, BorderLayout.CENTER);
        frame.setSize (400, 400);
        frame.setVisible (true);
    }
}

```

The above design of the panel class works fine for the above case of drawing one scatter plot of a dataset. What modifications do we need to make to our panel if we want to draw two plots? Or what if we want to draw a line plot instead of a scatter plot? How about five particles in a box, or what if we want to represent particles as rods instead of circles? The objects that we would like to draw will change frequently from program to program. With our current design, we need to modify the panel class each time we change what we are drawing, which is time consuming and inefficient. We need to keep what and how we draw separate from the panel, whose purpose is to convert between world and pixel coordinates and provide a blank canvas onto which we can draw.

A solution to this problem involves creating the `Drawable` interface that allows any object to draw itself on our panel, which we call the `PixelPanel`. This panel maintains a list of all the objects that we want to be displayed on the panel. When the panel paints itself (using the `paintComponent` method), it iterates through its list of `Drawable` objects and tells each `Drawable` object to draw itself on the panel. The `Drawable` interface is shown below.

Listing 4.13: Drawable interface.

```

package org.opensourcephysics.sip.ch4;

public interface Drawable {

```

```
// panel is the PixelPanel on which we will draw
// call methods xToPix and yToPix on the panel to convert between world and pixel coordinates
// g is graphics object that we will use to perform our drawing onto the PixelPanel
public void draw(PixelPanel panel, Graphics g);
}
```

In Listing 4.14 we define a dataset that knows how to draw itself on a PixelPanel:

Listing 4.14: A Dataset that can draw itself.

```
package org.opensourcephysics.sip.ch4;
import java.awt.Graphics;

public class DrawableDataset extends Dataset implements Drawable {
    int markerSize = 4; // size in pixels of marker

    public DrawableDataset(int number_of_points) {
        super(number_of_points);
    }

    public void draw(PixelPanel drawingPanel, Graphics g) {
        for (int i = 0; i < index; i++) {
            int xp = drawingPanel.xToPix(xpoints[i]);
            int yp = drawingPanel.yToPix(ypoints[i]);
            g.fillRect (xp - markerSize/2, yp - markerSize/2, markerSize,
            markerSize);
        }
    }
}
```

Finally, we define the PixelPanel class, which is a simplified version of the DrawingPanel class in the OSP display package:

Listing 4.15: PixelPanel class.

```
package org.opensourcephysics.sip.ch4;
import java.awt.Dimension;
import java.awt.Graphics;
import javax.swing.JPanel;
import java.util.ArrayList;

public class PixelPanel extends JPanel {
    Dimension size; // same as previous example
    // list of objects that will draw themselves on panel
    ArrayList drawableList = new ArrayList();
    double xmin, xmax, ymin, ymax;

    public void paintComponent(Graphics g) {
        super.paintComponent(g); // paint background
        size = getSize(); // update size of panel for xToPix and yToPix methods
        for (int i = 0; i < drawableList.size(); i++) {
            /* iterate through list of drawable objects, invoking
```

```

        draw method for every drawable object */
        Drawable drawable = (Drawable) drawableList.get(i);
        drawable.draw(this, g);
    }
}

// method that drawable objects can use to tell panel that they want to be drawn on panel
public void addDrawable(Drawable drawable) {
    drawableList.add(drawable);
}

// following methods are the same as the previous example
public void setXYMinMax(double xmin, double xmax, double ymin, double ymax)
public int xToPix(double x)
public int yToPix(double y)
}

```

In the above, we used the `ArrayList` class which implements an array that can vary automatically in size.

In Listing 4.16 we give an example of how to use these classes to plot the sin function.

Listing 4.16: Example of plot of sine function.

```

package org.opensourcephysics.sip.ch4;
import javax.swing.JFrame;
import java.awt.BorderLayout;
import java.awt.Container;

public class PixelPanelTestApp {
    public static void main(String[] args) {
        PixelPanel panel = new PixelPanel();
        JFrame frame = new JFrame();
        Container c = frame.getContentPane();
        c.add(panel, BorderLayout.CENTER);
        frame.setSize(400,400);
        frame.setVisible(true);
        double xmax = Math.PI;
        double xmin = 0;
        double ymax = 1;
        double ymin = -1;
        panel.setXYMinMax(xmin,xmax,ymin,ymax);
        int numberOfPoints = 10;
        double dx = (xmax - xmin)/(numberOfPoints-1);
        DrawableDataset dataset = new DrawableDataset(numberOfPoints);
        panel.addDrawable(dataset);
        double x = xmin;
        while (x <= xmax) {
            double y = Math.sin(x);
            dataset.append(x,y);
            x += dx;
        }
    }
}

```

```
        }  
        panel.repaint();  
    }  
}
```

The `org.opensourcephysics.display` package contains the `PlottingPanel` class that we will use to draw plots. This class is similar to the `PixelPanel` class, with some added functionality, such as an x and y axis. We also will use the `Dataset` class in the `display` package, which is similar to the `DrawableDataset`. Listing 4.17 shows how to use these classes to plot the \sin function.

Listing 4.17: Use of OSP library to plot a function.

```

package org.opensourcephysics.sip.ch4;
import org.opensourcephysics.display.Dataset;
import org.opensourcephysics.display.PlottingPanel;
import org.opensourcephysics.display.DrawingFrame;

// example that shows how to use OSP tools to make a plot
public class SimplePlotApp {
    public static void main(String[] args) {
        PlottingPanel panel = new PlottingPanel("x", "y", "sin x");
        DrawingFrame frame = new DrawingFrame(panel);
        double xmax = Math.PI;
        double xmin = 0;
        double ymax = 1;
        double ymin = -1;
        panel.setPreferredMinMax(xmin,xmax,ymin,ymax);
        int numberOfPoints = 10;
        double dx = (xmax - xmin)/(numberOfPoints-1);
        Dataset dataset = new Dataset();
        panel.addDrawable(dataset);
        double x = xmin;
        while (x <= xmax) {
            double y = Math.sin(x);
            dataset.append(x,y);
            x += dx;
        }
        panel.repaint ();
    }
}

```

References and Suggestions for Further Reading

David M. Geary, *Graphic Java: Mastering the JFC*, Vol. 1, AWT and Vol. 2, Swing, Prentice Hall (1999).

Jonathan Knudsen, *Java 2D Graphics*, O'Reilly (1999).

Chapter 5

Motion in One Dimension

©2002 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
14 February 2002

We introduce finite difference methods for obtaining numerical solutions to Newton's equations of motion and discuss the qualitative and quantitative behavior of bodies falling near the earth's surface.

5.1 Background

A common example of a physics problem that requires the solution of a differential equation is the motion of a particle acted on by a force. For simplicity, we first discuss one-dimensional motion so that only a single vector component of position, velocity, and acceleration are needed. These variables are related using the language of differential calculus:

$$v(t) = \frac{dy(t)}{dt} \quad (5.1)$$

and

$$a(t) = \frac{dv(t)}{dt}. \quad (5.2)$$

These quantities are known as *kinematical* quantities, because they describe the motion without regard to its cause.

Why do we need the concept of acceleration in kinematics? The answer can be found only *a posteriori*. Thanks to Newton, we know that the net force acting on a particle determines its acceleration. Newton's second law of motion tells us that

$$a(t) = \frac{F(y, v, t)}{m}, \quad (5.3)$$

where F is the net *force* and m is the *inertial mass*. In general, the force depends on position, velocity, and time. Note that Newton's law implies that the motion of a particle does not depend on d^2v/dt^2 or on any higher derivative of the velocity. It is a property of nature, not of mathematics, that we can find simple explanations for motion.

The two first-order equations (5.1) and (5.3) can be combined to obtain the familiar *second-order* differential equation for the position:

$$\frac{d^2y(t)}{dt^2} = \frac{F}{m}. \quad (5.4)$$

However, we will find it easier to describe the motion of a particle by two coupled first-order differential equations (5.1) and (5.3).

Because many types of systems can be modeled by differential equations such as (5.1) and (5.3), it is important to know how to solve such equations. In general, *analytical* solutions of differential equations, that is, solutions in terms of well-known functions, do not exist. We are therefore motivated to find numerical solutions of differential equations. However, analytical solutions are very important and often exist in special or limiting cases. We often use them to test our numerical solutions.

To introduce some simple ways of analyzing numerical solutions, we begin by studying a single differential equation before we tackle Newton's second law. Our understanding will be aided by a visual display of the dependence of the solution on the relevant parameters.

5.2 The Euler Algorithm

The standard technique for numerically solving a differential equation is to convert the differential equation to a *finite difference* equation. Let us consider a first-order differential equation of the form

$$\frac{dy}{dx} = f(x, y) \quad (5.5)$$

and analyze its meaning. Suppose that at $x = x_0$, y has the value y_0 . Because (5.5) tells us how y changes at (x_0, y_0) , we can find the *approximate* value of y at the neighboring point $x_1 = x_0 + \Delta x$, if Δx is small. The simplest approximation is to assume that $f(x, y)$, the rate of change of y with respect to x , is constant over the interval x_0 to x_1 . Then the approximate value of y at $x_1 = x_0 + \Delta x$ is given by

$$y_1 = y(x_0) + \Delta y \approx y(x_0) + f(x_0, y_0)\Delta x. \quad (5.6)$$

We can repeat this procedure to find the value of y at the point $x_2 = x_1 + \Delta x$:

$$y_2 = y(x_1 + \Delta x) \approx y(x_1) + f(x_1, y_1)\Delta x. \quad (5.7)$$

This procedure can be generalized to calculate the approximate value of y at any point $x_{n+1} = x_n + \Delta x$ by the iterative formula

$$y_{n+1} = y_n + f(x_n, y_n)\Delta x. \quad (\text{Euler algorithm}) \quad (5.8)$$

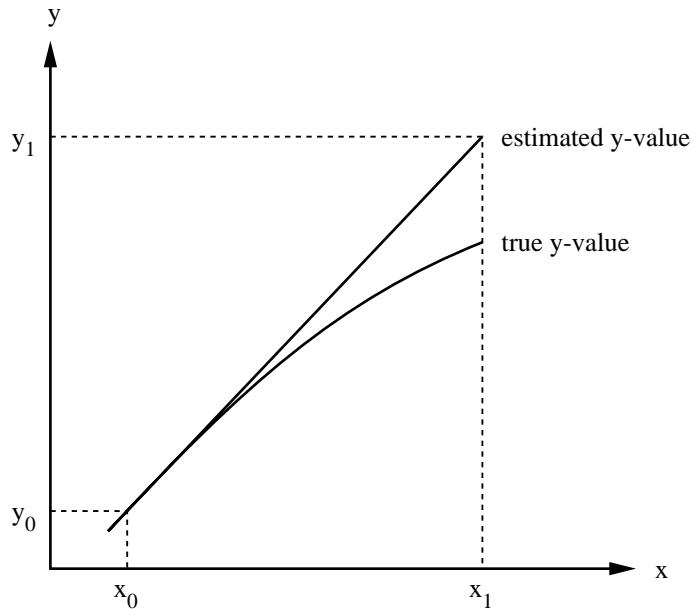


Figure 5.1: Graphical illustration of the Euler algorithm. The slope is evaluated at the beginning of the interval. The results of the Euler algorithm and the true function are represented by a straight line and a curve respectively.

This procedure is called the constant slope or *Euler* algorithm. We expect that (5.8) will yield a good approximation to the exact value of y if Δx is sufficiently small; the degree of “smallness” of Δx depends on our requirements and must be left vague until we consider specific applications.

The Euler algorithm assumes that the rate of change of y is constant over the interval x_n to x_{n+1} , and that the rate of change can be evaluated at the *beginning* of the interval. The graphical interpretation of (5.8) is shown in Figure 5.1. We see that if the slope changes during an interval, a discrepancy occurs between the numerical solution and the exact solution. Nonetheless, the discrepancy can be made smaller if we choose a smaller value of Δx .

5.3 A Simple Example

We first use the Euler algorithm to compute the numerical solution of the differential equation $dy/dx = 2x$ with the initial condition $y_0 = 1$ at $x_0 = 1$. Suppose that we wish to find the approximate value of y at $x = 2$. We choose $\Delta x = 0.1$, so that the number of steps is $n = (x_n - x_0)/\Delta x = (2 - 1)/0.1 = 10$.

The calculation can be arranged as in Table 5.1. The initial condition $x_0 = 1$ determines the initial slope $f(x_0) = 2x_0 = 2$. The value of y at the end of the interval, y_1 , is obtained from y_0 ,

n	x_n	y_n	$f(x) = 2x$	$y_n + \text{slope} \times 0.10$
0	1.00	1.00	2.00	$1.00 + 2.00 \times 0.10 = 1.20$
1	1.10	1.20	2.20	$1.20 + 2.20 \times 0.10 = 1.42$
2	1.20	1.42	2.40	$1.42 + 2.40 \times 0.10 = 1.66$
3	1.30	1.66	2.60	$1.66 + 2.60 \times 0.10 = 1.92$
4	1.40	1.92	2.80	$1.92 + 2.80 \times 0.10 = 2.20$
5	1.50	2.20	3.00	$2.20 + 3.00 \times 0.10 = 2.50$
6	1.60	2.50	3.20	$2.50 + 3.20 \times 0.10 = 2.82$
7	1.70	2.82	3.40	$2.82 + 3.40 \times 0.10 = 3.16$
8	1.80	3.16	3.60	$3.16 + 3.60 \times 0.10 = 3.52$
9	1.90	3.52	3.80	$3.52 + 3.80 \times 0.10 = 3.90$
10	2.00	3.90		

Table 5.1: Iterated solution of the differential equation $dy/dx = 2x$ with $y = 1$ at $x = 1$ using the Euler algorithm. The step size is $\Delta x = 0.1$. Three significant figures are shown.

the value of y at the beginning of the interval, by the relation

$$y_1 = y_0 + \text{slope} \times \Delta x = 1 + 2 \times 0.1 = 1.2. \quad (5.9)$$

This value of y is then transferred to the second line of Table 5.1 and the process is repeated. In this way we find that $y = 3.90$ at $x = 2$. For comparison, the true solution is $y = x^2 = 4$; the error is 2.5%. Convince yourself that a smaller value of Δx improves the accuracy of the solution by redoing Table 5.1 using $\Delta x = 0.05$.

We will now discuss the program in Listing 5.1 that reproduces the steps in Table 5.1. The most important code is in method `calculate` which implements the Euler algorithm using a `while` loop.

Listing 5.1: Implementation of the Euler algorithm.

```
package org.opensourcephysics.sip.ch5;
import java.awt.Color;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;

public class EulerTestApp implements Calculation {
    Control myControl;
    PlottingPanel plottingPanel;
    DatasetCollection datasetCollection;
    DrawingFrame plottingFrame;
    DataTable dataTable;
    DataTableFrame tableFrame;
    double x0 = 1.0;
    // initial value of x
    double y0 = 1.0;
    // initial value of y
    double xmax = 2.0;           // x value that stops the calculation
```

```

double dx = 0.10;           // step size
int datasetIndex;

public EulerTestApp() { // EulerTestApp constructor
    // set up the plotting panel
    plottingPanel = new PlottingPanel("x", "y", "dy/dx");
    plottingFrame = new DrawingFrame(plottingPanel);
    dataTable = new DataTable();           // set up table
    datasetCollection = new DatasetCollection();
    dataTable.add(datasetCollection);
    tableFrame = new DataTableFrame(dataTable);
    plottingPanel.addDrawable(datasetCollection); // add new datasetCollection to table
}

public void setControl(Control c) {
    myControl = c;
    resetCalculation();
}

public double getRate(double x) {
    return 2*x;                      // rate of change of y
}

public void calculate() { // read values from control
    x0 = myControl.getDouble("x0");
    y0 = myControl.getDouble("y0");
    xmax = myControl.getDouble("xmax");
    dx = myControl.getDouble("dx");
    double x = x0;
    double y = y0;
    datasetIndex++;                // set to -1 in reset
    datasetCollection.append(datasetIndex, x, y); // add initial values
    while (x < xmax) {
        y = y + getRate(x)*dx; // iterate y
        x = x + dx;           // increase x
        datasetCollection.append(datasetIndex, x, y); // store result
    }
    plottingPanel.repaint();       // redraw panel including datasetCollection
    dataTable.refreshTable();      // redraw the table
}

public void resetCalculation() {
    datasetIndex = -1;
    datasetCollection.clear();
    myControl.setValue("x0", 1);    // default initial value of x
    myControl.setValue("y0", 1);    // default initial value of y
    myControl.setValue("xmax", 2); // default maximum value of x
    myControl.setValue("dx", 0.1); // default step size
    plottingPanel.repaint();      // redraw panel including datasetCollection
    dataTable.refreshTable();     // redraw table
}

```

```

    }

public static void main(String[] args) {
    EulerTestApp app = new EulerTestApp();
    CalculationControl c = new CalculationControl(app);
    app.setControl(c);
}
}

```

The remaining methods do the usual housekeeping chores of creating display objects, setting default parameter values, and registering the calculation with a control. The **Reset** button restores the default values of the parameters and removes all data sets. The **Calculate** button creates a new data set each time it is pressed and enables the user to run and compare the calculation with different parameters.

Problem 5.1 requires that you study, compile, and modify the **EulerApp** program. This program can be downloaded from the Chapter 5 package on the OSP website.

Problem 5.1. Euler algorithm

- Compile and test the **EulerApp** program. Verify that the output is identical to what is shown in Table 5.1.
- The choice of a **for** loop or a **while** loop is usually a matter of taste or convenience. Modify the program to replace the **while** loop by a **for** loop in the **calculate** method.
- Show that the analytical solution of $dy/dx = 2x$ can be written in the form

$$y(x) = y_0 e^{kx}. \quad (5.10)$$

Note that $y(x=0) = y_0$. What is the value of k in this case?

- Create a second **datasetCollection** that displays the analytical solution (5.10) together with the numerical solution in the **dataTable** and the graph.
- Does the step size Δx have any significance? Create a third data set that displays the difference between the analytical solution and the numerical solution in the data table. Show that the error decreases as the step size Δx is decreased. What value of Δx is sufficiently small so that it does not affect your numerical results? Explain.
- Can the error of the numerical solution be decreased to an arbitrarily small value by decreasing the step size indefinitely? Determining the accuracy of a solution is important and will be discussed in Section 5.8.

5.4 Nuclear Decay

The power of mathematics when applied to physics comes in part from the fact that frequently seemingly unrelated problems can have the same mathematical formulation. Hence, if we can solve one problem, we can solve other problems that might appear to be unrelated. For example, the

growth of bacteria, the cooling of a cup of hot water, the discharge of a capacitor in an RC circuit, and nuclear decay can all be formulated in equivalent ways.

Consider a large number of radioactive nuclei. Although the number of nuclei is discrete, we can often treat this number as a continuous variable. Using this approach, the fundamental law of radioactive decay is that the rate of decay is proportional to the number of nuclei. Thus we can write

$$\frac{dN}{dt} = -\lambda N, \quad (5.11)$$

where N is the number of nuclei and λ is the decay constant. Note that the form of (5.11) is identical to (5.9). Of course, we do not need to use a computer to solve this decay equation, and it can be solved analytically as

$$N(t) = N_0 e^{-\lambda t}, \quad (5.12)$$

where N_0 is the initial number of particles.

Although units are not specified in (5.11) or (5.12), units are important. However, computer programs store numbers and numbers do not have units. For example, as we saw in Section 2.2, we can write the speed of light as

```
public static final double SPEED_OF_LIGHT = 3.0e8;
```

and no explicit units are specified. Because it is awkward to treat very large or very small numbers on a computer, it is convenient to choose units such that the calculated values of the variables are not too far from unity. For example, an astronomical calculation might use a time unit of one year (see Chapter 7) in contrast to the choice of a second as the time unit if we were simulating the motion of a body falling near the earth's surface (see Section 5.5). The different choices can cause confusion because the time units can be anything we want them to be.

In the context of nuclear decay, we want to measure time in terms of the natural time scale in the problem. A commonly used time unit for radioactive decay is the half-life, $T_{1/2}$, the time it takes for one-half of the original nuclei to decay. Another natural time scale is the characteristic time, τ , the time it takes for $1/e$ of the original nuclei to decay.

Problem 5.2. Single nuclear species decay

- a. Modify EulerApp so that nuclear decay notation is used in your program. Add a variable to the control so that can input the decay constant, λ . For $\lambda = 1$ and $\Delta t = 0.01$, compute the difference between the analytical result and the result of the Euler algorithm for $N(t)/N(0)$ at $t = 1$ and $t = 2$.
- b. Use your modified program to verify that the half life, $T_{1/2}$, is $\ln 2/\lambda$. How long does it take for $1/e$ of the original nuclei to decay?
- c. Determine the decay constant λ in units of s^{-1} for $^{238}\text{U} \rightarrow ^{234}\text{Th}$ if the half-life is 4.5×10^9 years. What time step would be appropriate for the numerical solution to the rate equation if this rate were used? How would these values change if the particle being modeled were a muon with a lifetime of 2.2×10^{-6} second?

- d. Modify your program so that the time t is measured in terms of the half-life. That is, at $t = 1$ half the particles should have decayed and at $t = 2$, one quarter of the particles have decayed. Use your program to determine the time (in years) for 100 atoms of ^{238}U to decay to 20% of their original number. Does the graph change if you calculate the decay of muons?

5.5 Constant Acceleration

In the absence of air resistance, all particles, regardless of size, mass, or composition, have the same acceleration at the same point near the earth's surface. This idealized motion is called "free fall." According to (5.3), constant acceleration implies that the force per unit mass, F/m , is a constant. This constant is commonly denoted by the symbol g . Near the earth's surface, the magnitude of g is approximately 9.8 N/kg or 9.8 m/s². Let us adopt the coordinate system shown in Figure 5.2 with the positive direction upward. In this case $a = -g$ and the solution of (5.4) can be written as

$$v(t) = v_0 - gt, \quad (5.13a)$$

and

$$y(t) = y_0 + v_0 t - \frac{1}{2}gt^2, \quad (5.13b)$$

where y_0 and v_0 are the initial position and velocity of the particle respectively.

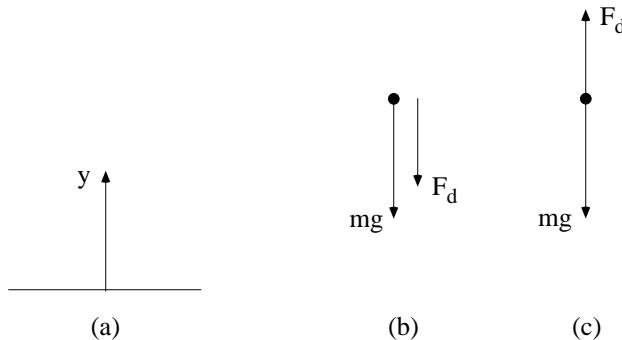


Figure 5.2: (a) Coordinate system with y measured positive upwards from the ground. (b) The force diagram for upward motion. (c) The force diagram for downward motion.

The analysis of free fall under the influence of a constant net force is straightforward, and there is no need to use a computer. However, to ease our introduction to the solution of Newton's second law using numerical methods, we will use a computer in this familiar context. We begin by encapsulating the analytical solution in Listing 5.2:

Listing 5.2: Analytical solution of freely falling particle.

```
public class ParticleFree extends Particle {
```

```

public ParticleFree (double _y0, double _v0) {
    y = _y0;
    v = _v0;
}

// analytical solution of freely falling particle
public void move (double dt) {
    a = -g; // included here to emphasize that acceleration is constant
    y = y + v*dt + 0.5*a*dt*dt;
    double v = v + a*dt;
}
}

```

We next apply Euler's algorithm to Newton's equations of motion. We write Newton's second law (5.4) as a system of coupled first-order differential equations and apply the Euler algorithm to each variable. The variables of interest are the particle's vertical position y and velocity v , and the time. The algorithm can be written as:

$$y_{n+1} = y_n + v_n \Delta t \quad (5.14a)$$

$$v_{n+1} = v_n + a_n \Delta t \quad (5.14b)$$

$$t_{n+1} = t_n + \Delta t. \quad (5.14c)$$

Note that we have added an explicit rate equation for time, $dt/dt = \dot{t} = 1$, to the differential equations for y and v . By placing the independent variable, time, on an equal footing with the other dynamical quantities will allow us to easily introduce time-dependent forces in later chapters.

Rather than including the algorithm (5.14) into the `calculate` method, we encapsulate it in a separate class named `ParticleFreeEuler` as follows:

Listing 5.3: Implementation of Euler algorithm for freely falling particle.

```

public class ParticleFreeEuler extends Particle {

    // compute position and velocity of particle in free fall using Euler algorithm
    public void move (double dt) {
        a = -g; // How would statement be changed if drag resistance present?
        y = y + v*dt;
        v = v + a*dt;
    }
}

```

We next list the code for the target class, `ParticleFreeApp`, which instantiates the classes `ParticleFreeEuler` and `ParticleFree` and compares the numerical and analytical solutions. The most important method in `ParticleFreeApp` is the `calculate` method. This method calculates the Euler solution using an instance of `ParticleFreeEuler` and compares it to the analytical solution using an instance of `ParticleFree`. Note that its structure is similar to calculate methods in the previous programs. It is coded as follows:

Listing 5.4: Implementation of the Calculation interface for the freely falling particle.

```

package org.opensourcephysics.ch5;

```

```

import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;
import java.awt.Color;

public class ParticleFreeApp implements Calculation {
    Control myControl;
    PlottingPanel plottingPanel;
    DrawingFrame plottingFrame;
    DataTable dataTable;
    DataTableFrame tableFrame;
    ParticleFreeEuler euler;
    ParticleFree exact;           // no truncation error
    DatasetCollection datasetCollection;
    double tmin = 0;
    double tmax = 100;
    double y0 = 10.0;            // initial y position
    double v0 = 0;               // initial y velocity
    int numberOfPoints;

    public ParticleFreeApp() {
        plottingPanel = new PlottingPanel("Time t (s)", "Height y (m)", "y vs. t");
        plottingFrame = new DrawingFrame(plottingPanel);
        dataTable = new DataTable();
        tableFrame = new DataTableFrame(dataTable);
        datasetCollection = new DatasetCollection();
        datasetCollection.setXYColumnNames(0, "time", "exact");
        datasetCollection.setXYColumnNames(1, "time", "euler");
        dataTable.add(datasetCollection);
        plottingPanel.addDrawable(datasetCollection);
        // don't display x values of euler data
        dataTable.setColumnVisible(datasetCollection, 2, false);
    }

    public void setControl(Control control) {
        myControl = control;
        resetCalculation ();
    }

    public void calculate() {
        y0 = myControl.getDouble("initial y");
        v0 = myControl.getDouble("initial v");
        tmax = myControl.getDouble("tmax (s)"); // time when calculation should stop
        numberOfPoints = myControl.getInt("number of points");
        plottingPanel.setPreferredMinMaxX(tmin, tmax);
        euler.y = y0;                      // set initial state
        euler.v = v0;
        exact.y = y0;
        exact.v = v0;
        double dt = (tmax - tmin)/(numberOfPoints - 1);
    }
}

```

```

double time = 0;
for (int i = 0; i < numberOFPoInts; i++) {
    exact.move(dt);
    datasetCollection.append(0, time, exact.y); // add data to dataset
    euler.move(dt);
    datasetCollection.append(1, time, euler.y);
    time += dt;
}
plottingPanel.repaint(); // redraw datasets
dataTable.refreshTable(); // redraw table
}

public void resetCalculation() {
    plottingPanel.clear();
    dataTable.clear();
    datasetCollection.clear();
    y0 = 10;
    v0 = 0;
    tmax = 100;
    numberOFPoInts = 20;
    myControl.setValue("initial y", y0); // store initial height
    myControl.setValue("initial v", v0); // store initial velocity
    myControl.setValue("tmax (s)", tmax); // store time when calculation should stop
    myControl.setValue("number of points", numberOFPoInts); // store number of points in the plot
    plottingPanel.repaint(); // redraw datasets
    dataTable.refreshTable(); // redraw table
}

public static void main(String[] args) {
    Calculation app = new ParticleFreeApp();
    Control c = new CalculationControl(app);
    app.setControl(c);
}
}

```

After reading the input parameters, the `calculate` method uses a `while` loop to compute y and v until the particle hits the ground at $y = 0$. The only difference is that each Euler data point is generated by iterating the difference equation rather than by evaluating a function as in the `move` method in `ParticleFree`. Exercise 5.3 asks you to run the `ParticleFreeApp` program to see if it works as expected.

Exercise 5.3. Free Fall Test

- Compile and run `ParticleFreeApp`. Test the correctness of the program by comparing the results of the calculation of the analytical solution to a pencil and paper calculation.
- Using the default values $y = 20$ and $v = 0$, what value of Δt is required so that the computed value of y at $t = 1$ using the Euler algorithm is accurate to one percent? Does the required value of Δt change at $t = 2$ or does the numerical solution retain the same one percent accuracy for all times?

5.6 Some Other Simple Algorithms

The algorithm for obtaining a numerical solution of a differential equation is not unique, and there are many algorithms that reduce to the same differential equation in the limit $\Delta t \rightarrow 0$. For example, a simple variation of (5.14) is to determine y_{n+1} using v_{n+1} , the velocity at the *end* of the interval rather than at the beginning. We write this modified Euler algorithm as

$$v_{n+1} = v_n + a_n \Delta t \quad (5.15a)$$

$$y_{n+1} = y_n + v_{n+1} \Delta t. \quad (\text{Euler-Cromer algorithm}) \quad (5.15b)$$

We refer to (5.15) as the Euler-Cromer algorithm. Is there any *a priori* reason to prefer the Euler algorithm over the Euler-Cromer algorithm?

It might occur to you that it would be better to compute the velocity at the middle of the interval rather than at the beginning or at the end of the interval. The Euler-Richardson algorithm is based on this idea. This algorithm is particularly useful for velocity-dependent forces, but does as well as other simple algorithms for forces that do not depend on the velocity. The algorithm consists of using the Euler algorithm to find the intermediate position y_{mid} and velocity v_{mid} at a time $t_{\text{mid}} = t + \Delta t/2$. Then we compute the force, $F(y_{\text{mid}}, v_{\text{mid}}, t_{\text{mid}})$ and the acceleration a_{mid} at t_{mid} . The new position y_{n+1} and velocity v_{n+1} at time t_{n+1} is found using v_{mid} and a_{mid} . We summarize the Euler-Richardson algorithm as follows:

$$a_n = F(y_n, v_n, t_n)/m \quad (5.16a)$$

$$v_{\text{mid}} = v_n + \frac{1}{2} a_n \Delta t, \quad (5.16b)$$

$$y_{\text{mid}} = y_n + \frac{1}{2} v_n \Delta t, \quad (5.16c)$$

$$a_{\text{mid}} = F(y_{\text{mid}}, v_{\text{mid}}, t + \frac{1}{2} \Delta t)/m, \quad (5.16d)$$

and

$$v_{n+1} = v_n + a_{\text{mid}} \Delta t. \quad (5.17a)$$

$$y_{n+1} = y_n + v_{\text{mid}} \Delta t. \quad (\text{Euler-Richardson algorithm}) \quad (5.17b)$$

Although we need to do twice as many computations per time step, the Euler-Richardson algorithm is much faster than the Euler algorithm because we can make the time step larger and still obtain better accuracy than with either the Euler or Euler-Cromer algorithms. A derivation of the Euler-Richardson algorithm is given in Appendix 5A. You are asked to implement the Euler-Cromer and Euler-Richardson algorithms in Exercise 5.4.

Exercise 5.4. Algorithm Test

- Write a class named `ParticleFreeEulerCromer` that encapsulates the Euler-Cromer algorithm and modify the `ParticleFreeApp` program to test this class. What time step is needed to achieve one percent accuracy after an elapsed time of one second?
- Repeat part (a) using the Euler-Richardson algorithm. Which algorithm works best for the case of a freely falling particle?

5.7 Forces on Falling Objects

The analytical solution for free fall near the earth's surface, (5.13), is well known. However, it is not difficult to think of more realistic models of motion near the earth's surface for which the equations of motion do not have simple analytical solutions. For example, if we take into account the variation of the earth's gravitational field with the distance from the center of the earth, then the force on a particle is not constant. According to Newton's law of gravitation, the force due to the earth on a particle of mass m is given by

$$F = \frac{GMm}{(R+y)^2} = \frac{GMm}{R^2(1+y/R)^2} = mg\left(1 - 2\frac{y}{R} + \dots\right), \quad (5.18)$$

where y is measured from the earth's surface, R is the radius of the earth, M is the mass of the earth, G is the gravitational constant, and $g = GM/R^2$.

Problem 5.5. Position-dependent force

Modify ParticleFreeApp to simulate the fall of a particle with the position-dependent force law (5.18). Assume that a particle is dropped from a height h with zero initial velocity and compute its impact velocity (speed) when it hits the ground at $y = 0$. Determine the value of h for which this impact velocity differs by one percent from its value with a constant acceleration $g = 9.8 \text{ m/s}^2$. Take the radius of the earth to be $6.37 \times 10^6 \text{ m}$. Make sure that the one percent error is due to the physics of the force law and not the accuracy of the algorithm.

For particles near the earth's surface, a more important modification of the free fall problem is the retarding drag force due to air resistance. The direction of the drag force is opposite to the velocity of the particle. We first discuss the motion of a falling body. The direction of the drag force F_d is opposite to the motion and is upward as shown in Figure 5.2c. Hence, the total force F on the particle can be written as

$$F = -mg + F_d. \quad (5.19)$$

In general, it is necessary to determine the velocity dependence of F_d empirically over a limited range of conditions. One way to find the form of $F_d(v)$ is to measure y as a function of t and to determine the velocity and acceleration as a function of t . From this information it is possible to find the acceleration as a function of v and to extract $F_d(v)$ from (5.19). However, this algorithm is not useful in general, because errors are introduced by taking the derivatives needed to find the velocity and acceleration. A better method is to reverse the procedure, that is, assume an explicit form for the v dependence of $F_d(v)$, and use it to solve for $y(t)$. If the calculated $y(t)$ is consistent with the experimental values of $y(t)$, then the assumed v dependence of $F_d(v)$ is justified empirically.

The two most commonly assumed forms of the velocity dependence of $F_d(v)$ are

$$F_d(v)/m = C_1 v \quad (5.20a)$$

and

$$F_d(v)/m = C_2 v^2, \quad (5.20b)$$

where the parameters C_1 and C_2 depend on the properties of the medium and the shape of the object. We stress that the forms (5.20) are not exact laws of physics, but instead are useful *phenomenological* expressions that yield approximate results for $F_d(v)$ over a limited range of v .

Because $F_d(v)$ increases as v increases, there is a limiting or *terminal velocity* (speed) at which $F_d = mg$ and the acceleration equals zero. This terminal speed can be found from (5.19) and (5.20) and is given by

$$v_t = \frac{g}{C_1} \quad (5.21a)$$

or

$$v_t = \left(\frac{g}{C_2} \right)^{1/2} \quad (5.21b)$$

for the linear and quadratic cases, respectively. It frequently is convenient to measure velocities in terms of the terminal velocity. We can use (5.20) and (5.21) to write F_d in the linear and quadratic cases as

$$F_{1,d}/m = C_1 v_t \left(\frac{v}{v_t} \right) = mg \frac{v}{v_t} \quad (5.22a)$$

and

$$F_{2,d}/m = C_2 v_t^2 \left(\frac{v}{v_t} \right)^2 = mg \left(\frac{v}{v_t} \right)^2. \quad (5.22b)$$

Hence, we can write the net force on a falling object in the convenient form

$$F_1(v) = -mg \left(1 - \frac{v}{v_t} \right), \quad (5.23a)$$

and

$$F_2(v) = -mg \left(1 - \frac{v^2}{v_t^2} \right). \quad (5.23b)$$

To determine if the effects of air resistance are important in the fall of ordinary objects, consider the fall of a pebble of mass $m = 10^{-2}$ kg. To a good approximation, the drag force is proportional to v^2 . For a spherical pebble of radius 0.01 m, C_2 is found empirically to be approximately 10^{-2} kg. From (5.21b) we find the terminal velocity to be about 30 m/s. Because this speed would be achieved by a freely falling body in a vertical fall of approximately 50 m in a time of about 3 s, we expect that the effects of air resistance would be appreciable for comparable times and distances. Hence, many of the textbook problems we encounter in elementary mechanics courses are not realistic. More realistic physics is treated in Problems 5.7 and 5.6.

Problem 5.6. The fall of a styrofoam ball

- Use the empirical data for the displacement $y(t)$ in Table 5.2 to estimate the velocity $v(t)$ using the central difference approximation given in (3.10):

$$v(t) \approx \frac{y(t + \Delta t) - y(t - \Delta t)}{2\Delta t}. \quad (\text{central difference approximation}) \quad (5.24)$$

Show that if we write the acceleration as $a(t) \approx [v(t + \Delta t) - v(t)]/\Delta t$ and use the forward difference approximation for the velocity

$$v(t) \approx \frac{y(t + \Delta t) - y(t)}{\Delta t}, \quad (\text{forward difference approximation}) \quad (5.25)$$

t (s)	position (m)
-0.132	0.0
0.0	0.075
0.1	0.260
0.2	0.525
0.3	0.870
0.4	1.27
0.5	1.73
0.6	2.23
0.7	2.77
0.8	3.35

Table 5.2: Results by Greenwood, Hanna, and Milton (see references) for the vertical fall of a styrofoam ball of mass 0.254 gm and radius 2.54 cm. Note that the initial time is negative and not an integer multiple of 0.1.

we can express the acceleration as

$$a(t) \approx \frac{y(t + \Delta t) - 2y(t) + y(t - \Delta t)}{(\Delta t)^2}. \quad (5.26)$$

Use (5.24) and (5.26) to estimate the velocity and acceleration. Estimate the terminal velocity from the data given in Table 5.2. This estimation is nontrivial, in part because the terminal velocity has not yet been reached during the time interval shown in the table. Use your approximate results for $v(t)$ and $a(t)$ to plot a as a function of v and, if possible, determine the nature of the velocity-dependence of a . Discuss the accuracy of your results for the acceleration.

- b. Adopt one of the numerical algorithms that we have discussed and write a class that encapsulates this algorithm for the motion of a particle with quadratic drag resistance.
- c. Use your program with the net force given by either (5.23a) or (5.23b). Choose the terminal velocity as one of the input parameters, and take as your first guess for the terminal velocity the value you found in part (a). Make sure that your computed results for the displacement of the particle, $y(t) - y(0)$, do not depend on Δt . Compare your plot of the computed values of $y(t) - y(0)$ for the linear and quadratic form of the drag resistance with the empirical values of $y(t) - y(0)$ and visually determine which form of the drag force yields the best overall fit. What is your criteria for the “best” fit? Should you match your results with the experimental data at early times or at later times? Or should you adopt another strategy? What are the qualitative differences between the two computed forms of $y(t) - y(0)$?

Problem 5.7. Effect of air resistance on the ascent and descent of a pebble

- a. Verify the claim made in Section 5.7 that the effects of air resistance on a falling pebble can be appreciable. Compute the speed at which a pebble reaches the ground if it is dropped from rest at a height of 50 m. Compare this speed to that of a freely falling object under the same conditions. Assume that the drag force is proportional to v^2 and that the terminal velocity is 30 m/s.

- b. Suppose a pebble is thrown vertically upward with an initial velocity v_0 . In the absence of air resistance, we know that the maximum height reached by the pebble is $v_0^2/2g$, its velocity upon return to the earth equals v_0 , the time of ascent equals the time of descent, and the total time in the air is $2v_0/g$. Before performing a numerical simulation, give a simple qualitative explanation of how you think these quantities will be affected by air resistance. In particular, how will the time of ascent compare with the time of decent?
- c. Perform a simulation to determine if your qualitative answers are correct. Assume that the drag force is proportional to v^2 . From (5.22) we see that we can characterize the magnitude of the drag force by a terminal velocity even if the motion of the pebble is upward and even if the pebble never attains this velocity. Choose the terminal velocity $v_t = 30 \text{ m/s}$. Suggestions: It is a good idea to choose an initial velocity that allows the pebble to remain in the air for a total time such that the time of ascent differs appreciably from the time of descent. A reasonable choice is $v(t=0) = 50 \text{ m/s}$. Choose the coordinate system shown in Figure 5.2 with y positive upward. What is the net force for $v > 0$ and $v < 0$? You might find it convenient to the drag force in the form $F_d = -v * \text{Math.abs}(v)$. One way to determine the maximum height of the pebble is to use the statement

```
if (v*vold < 0)
    System.out.println("maximum height = " + y);
```

where $v = v_{n+1}$ and $v_{\text{old}} = v_n$.

Multiple nuclear decays can also produce systems of first-order differential equations. Problem 5.8 asks you to model such a system using the techniques that we have developed for falling particles.

Problem 5.8. Multiple nuclear decays

1. ^{76}Kr decays to ^{76}Br via electron capture with a half-life of 14.8 h, and ^{76}Br decays to ^{76}Se via electron capture and positron emission with a half-life of 16.1 h. Suppose that the sample initially contains 1 gm of pure ^{76}Kr . In this case there are two half-lives, and it is convenient to measure time in units comparable to the smallest half-life. Write a program to compute the time dependence of the amount of ^{76}Kr and ^{76}Se over an interval of one week.
2. ^{28}Mn decays via beta emission to ^{28}Al with a half-life of 21 h, and ^{28}Al decays by positron emission to ^{28}Si with a half-life of 2.31 min. If we were to use minutes as the unit of time, our program would have to do many iterations before we would see a significant decay of the ^{28}Mn . What simplifying assumption can you make to speed up the computation?
3. ^{211}Rn decays via two branches as shown in Figure 5.3. Make any necessary approximations and compute the amount of each isotope as a function of time, assuming that the sample initially consists of 1 μg of ^{211}Rn .

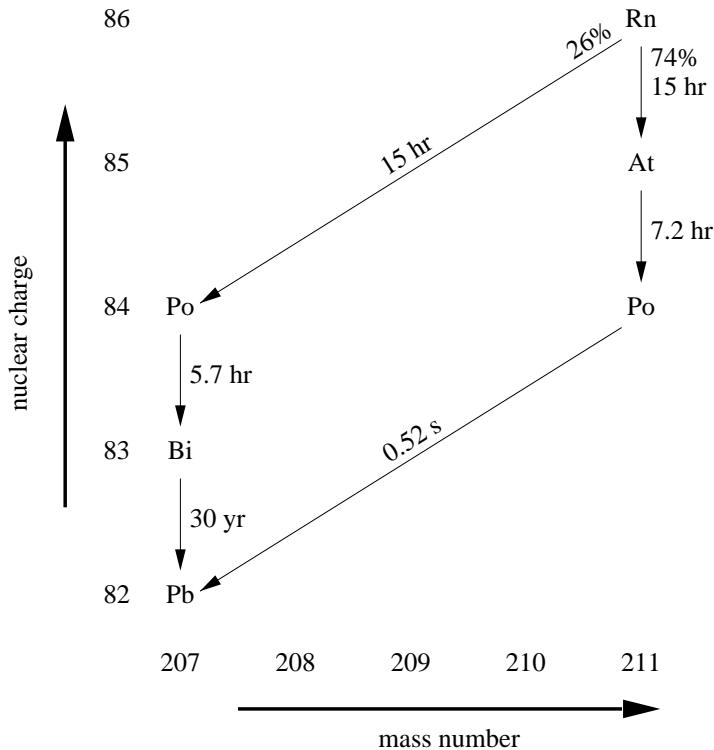


Figure 5.3: The decay scheme of ^{211}Rn . Note that ^{211}Rn decays via two branches, and the final product is the stable isotope ^{207}Pb . All vertical transitions are by electron capture (a form of beta decay), and all diagonal transitions are by alpha decay. The times represent half-lives.

5.8 Accuracy and Stability

Now that we have learned how to use the numerical methods to find a numerical solution to simple first-order differential equations, we need to develop some practical guidelines to help us estimate the accuracy of various methods. Because we have replaced a differential equation by a difference equation, our numerical solution cannot be identically equal, in general, to the “true” solution of the original differential equation. In general, the discrepancy between the two solutions has two causes. One cause is that computers do not store numbers with infinite precision, but rather to a maximum number of digits that is hardware and software dependent. Most programming languages allow the programmer to distinguish between *floating point* or *real* numbers, that is, numbers with decimal points, and *integer* or *fixed point* numbers. Arithmetic with numbers represented by integers is exact. However in general, we cannot solve a differential equation using integer arithmetic. Hence, arithmetic operations such as addition and division, which involve real numbers, can introduce an error, called the *roundoff error*. For example, if a computer only stored real numbers to two significant figures, the product 2.1×3.2 would be stored as 6.7 rather than the exact value 6.72. The

significance of roundoff errors is that they accumulate as the number of mathematical operations increases. Ideally we should choose algorithms that do not significantly magnify the roundoff error, for example, we should avoid subtracting numbers that are nearly the same in magnitude.

The other source of the discrepancy between the true answer and the computed answer is the error associated with the choice of algorithm. This error is called the *truncation error*. A truncation error would exist even on an idealized computer that stored floating point numbers with infinite precision and hence had no roundoff error. Because this error depends on the choice of algorithm and hence can be controlled by the programmer, you should be motivated to learn more about numerical analysis and the estimation of truncation errors. However, there is no general prescription for the best method for obtaining numerical solutions of differential equations. We will find in later chapters that each method has advantages and disadvantages and the proper selection depends on the nature of the solution, which you might not know in advance, and on your objectives. How accurate must the answer be? Over how large an interval do you need the solution? What kind of computer are you using? How much computer time and personal time do you have?

In practice, we usually can determine the accuracy of our numerical solution by reducing the value of Δt until the numerical solution is unchanged at the desired level of accuracy. Of course, we have to be careful not to choose Δt too small, because too many steps would be required and the total computation time and roundoff error would increase.

In addition to accuracy, another important consideration is the stability of an algorithm. For example, it might happen that the numerical results are very good for short times, but diverge from the true solution for longer times. This divergence might occur if small errors in the algorithm are multiplied many times, causing the error to grow geometrically. Such an algorithm is said to be *unstable* for the particular problem. We consider the accuracy and the stability of the Euler method in Problems 5.9 and 5.10.

Problem 5.9. Accuracy of the Euler method

- Use the modified version of the Euler program that you created for Problem 5.1 to compute the value of y at $x = 2 \text{ min}$ with $\Delta x = 0.1, 0.05, 0.025, 0.01$, and 0.005 . Your program should already have a table column showing the difference between the exact solution

$$y(x) = y_0 e^{kx} \quad \text{where} \quad y(x=0) = y_0 \quad (5.27)$$

and the numerical solution. Is the difference between these solutions a decreasing function of Δx ? That is, if Δx is decreased by a factor of two, how does the difference change? Plot the difference as a function of Δx . If your points fall approximately on a straight line, then the difference is proportional to Δx (for $\Delta x \ll 1$). A numerical method is called *n*th order, if the difference between the analytical solution and the numerical solution is proportional to $(\Delta x)^n$ at a fixed value of x . What is the order of the Euler method?

- One way to determine the accuracy of a numerical solution is to repeat the calculation with a smaller step size and compare the results. If the two calculations agree to p decimal places, we can reasonably assume that the results are correct to p decimal places. What value of Δt is necessary for 0.1% accuracy at $x = 2$? What value of Δt is necessary for 0.1% accuracy at $x = 4$?

Problem 5.10. Stability of the Euler method

- a. Consider the differential equation

$$R \frac{dQ}{dt} = V - \frac{Q}{C}, \quad (5.28)$$

with $Q = 0$ at $t = 0$. This equation represents the charging of a capacitor in an RC circuit with an applied voltage V . Measure t in seconds and choose $R = 2000\Omega$, $C = 10^{-6}$ farads, and $V = 10$ volts. Do you expect $Q(t)$ to increase with t ? Does $Q(t)$ increase indefinitely or does it reach a steady-state value? Write a program to solve (5.28) numerically using the Euler method. What value of Δt is necessary to obtain three decimal accuracy at $t = 0.005\text{s}$?

- b. What is the nature of your numerical solution to (5.28) at $t = 0.005\text{s}$ for $\Delta t = 0.005$, 0.0025 , and 0.001 ? Does a small change in Δt lead to a large change in the computed value of Q ? Is the Euler method stable in this calculation for any value of Δt ?

Problem 5.11. Second-order algorithm

As we saw in the Problem 5.9, the Euler method is only first-order in accuracy. We can increase the accuracy of Equation 5.9 to second-order by the following reasoning. Expand $y(x)$ in a Taylor series:

$$y(x + \Delta x) = y(x) + \frac{dy(x)}{dx} \Delta x + \frac{1}{2} \frac{d^2y(x)}{dx^2} (\Delta x)^2 + \dots, \quad (5.29)$$

and use (5.9) to write $d^2y(x)/dx^2 = k$, $dy(x)/dx$. We also use (5.9) to write $dy(x)/dx$ in terms of $y(x)$ and express (5.29) as

$$y(x + \Delta x) = y(x) + ky(x)\Delta x + \frac{k^2}{2}y(x)(\Delta x)^2 + \dots \quad (5.30)$$

If we write the Euler estimate as $y_e = y(x) + ky(x)\Delta x$, we can rewrite (5.30) as

$$y(x + \Delta x) = y(x) + \frac{k}{2}(y(x) + y_e)\Delta x. \quad (5.31)$$

From the form of (5.31), we see that a second-order algorithm can be obtained by using the average of the Euler estimate for the value of y at $x + \Delta x$ and the value of y at x in an Euler-like expression. Modify your program so that the second-order algorithm (5.31) is used and repeat Problem 5.9.

5.9 Levels of Simulation

So far we have done simulations in which the microscopic complexity of the system has been simplified considerably. Consider for example, the motion of a pebble falling through the air. First we reduced the complexity by representing the pebble as a particle. Then we reduced the number of degrees of freedom even more by representing the collisions of the pebble with the billions of molecules in the air by a velocity-dependent friction term. It is remarkable that the resultant

phenomenological model is a fairly accurate representation of realistic physical systems. However, what we gain in simplicity, we lose in range of applicability.

In a more detailed model, the individual physical processes would be represented in a more microscopic manner. For example, we can imagine doing a simulation in which the effects of the air are represented by a fluid of particles that collide with one another and with the falling particle. How accurately do we need to represent the potential energy of interaction between the fluid particles? Clearly the level of detail that is needed in a model depends on the accuracy of the corresponding experimental data and the type of information in which we are interested. For example, we do not need to take into account the influence of the moon on a pebble falling near the earth's surface. And the level of detail that we can simulate depends in part on the available computer resources.

The words *simulation* and *modeling* are frequently used interchangeably and their precise meaning is not important, especially because people who work with models and who do simulations do not use them precisely. Most practitioners would say that in Chapter 5 we have solved several mathematical models numerically. Beginning with Chapter 6, we will be able to say that we actually are doing simulations. The difference is that our models will represent physical systems in more detail, and we will give more attention to what physical quantities we should measure. In other words, our simulations will become more analogous to laboratory experiments.

Appendix 5A: The Euler-Richardson Method

We motivate the Euler-Richardson method (5.16) in the following. We write $y(t + \Delta t)$ as a Taylor series to second-order in Δt :

$$y_1 = y(t + \Delta t) = y(t) + v(t)\Delta t + \frac{1}{2}a(t)(\Delta t)^2, \quad (5.32)$$

where $a(t) = a(y(t), v(t), t)$. The notation y_1 implies that $y(t + \Delta t)$ is related to $y(t)$ by one time step. We also divide the step Δt into half steps and write the first half step, $y(t + \frac{1}{2}\Delta t)$, as

$$y(t + \frac{1}{2}\Delta t) = y(t) + v(t)\frac{\Delta t}{2} + \frac{1}{2}a(t)(\frac{\Delta t}{2})^2. \quad (5.33)$$

The second half step, $y_2(t + \Delta t)$, can be written as

$$y_2(t + \Delta t) = y(t + \frac{1}{2}\Delta t) + v(t + \frac{1}{2}\Delta t)\frac{\Delta t}{2} + \frac{1}{2}a(t + \frac{1}{2}\Delta t)(\frac{\Delta t}{2})^2. \quad (5.34)$$

We substitute (5.33) into (5.34) and obtain

$$y_2(t + \Delta t) = y(t) + \frac{1}{2}[v(t) + v(t + \frac{1}{2}\Delta t)]\Delta t + \frac{1}{2}[a(t) + a(t + \frac{1}{2}\Delta t)](\frac{1}{2}\Delta t)^2. \quad (5.35)$$

Now $a(t + \frac{1}{2}\Delta t) = a(t) + \frac{1}{2}a'(t)\Delta t + \dots$. Hence to order $(\Delta t)^2$, (5.35) becomes

$$y_2(t + \Delta t) = y(t) + \frac{1}{2}[v(t) + v(t + \frac{1}{2}\Delta t)]\Delta t + \frac{1}{2}[2a(t)](\frac{1}{2}\Delta t)^2. \quad (5.36)$$

We can create an approximation that is accurate to order $(\Delta t)^3$ by combining (5.32) and (5.36) so that the terms to order $(\Delta t)^2$ cancel. The combination that works is $2y_2 - y_1$, which gives the Euler-Richardson result:

$$y_{er}(t + \Delta t) = 2y_2(t + \Delta t) - y_1(t + \Delta t) = y(t) + v(t + \frac{1}{2}\Delta t)\Delta t + O(\Delta t)^3. \quad (5.37)$$

The same reasoning leads to an approximation for the velocity accurate to $(\Delta t)^3$ giving

$$v_{er} = 2v_2(t + \Delta t) - v_1(t + \Delta t) = v(t) + a(t + \frac{1}{2}\Delta t)\Delta t + O(\Delta t)^3. \quad (5.38)$$

A bonus of the Euler-Richardson method is that the quantities $|y_2 - y_1|$ and $|v_2 - v_1|$ give an estimate for the error in the procedure. We can use these estimates to change the time step so that the error is always within some desired level of precision.

References and Suggestions for Further Reading

William R. Bennett, *Scientific and Engineering Problem-Solving with the Computer*, Prentice Hall (1976). One of the first books to incorporate computer problem solving. Many one- and two-dimensional falling body problems are considered in Chapter 5.

Byron L. Coulter and Carl G. Adler, “Can a body pass a body falling through the air?,” *Am. J. Phys.* **47**, 841 (1979). The authors discuss the limiting conditions for which the drag force is linear or quadratic in the velocity.

Alan Cromer, “Stable solutions using the Euler approximation,” *Am. J. Phys.* **49**, 455 (1981). The author shows that a minor modification of the usual Euler approximation yields stable solutions for oscillatory systems including planetary motion and the harmonic oscillator (see Chapter 6).

R. M. Eisberg, *Applied Mathematical Physics with Programmable Pocket Calculators*, McGraw-Hill (1976). Chapter 3 of this handy paperback is similar in spirit to the present discussion.

Richard P. Feynman, Robert B. Leighton and Matthew Sands, *The Feynman Lectures on Physics*, Vol. 1, Addison-Wesley (1963). Feynman discusses the numerical solution of Newton’s equations in Chapter 9.

A. P. French, *Newtonian Mechanics*, W. W. Norton & Company (1971). Chapter 7 has an excellent discussion of air resistance and a detailed analysis of motion in the presence of drag resistance.

Ian R. Gatland, “Numerical integration of Newton’s equations including velocity-dependent forces,” *Am J. Phys.* **62**, 259 (1994). The author discusses the Euler-Richardson algorithm.

Margaret Greenwood, Charles Hanna, and John Milton, “Air resistance acting on a sphere: numerical analysis, strobe photographs, and videotapes,” *Phys. Teacher* **24**, 153 (1986). More experimental data and theoretical analysis are given for the fall of ping-pong and styrofoam balls. Also see Mark Peastrel, Rosemary Lynch, and Angelo Armenti, “Terminal velocity of a shuttlecock in vertical fall,” *Am. J. Phys.* **48**, 511 (1980).

K. S. Krane, “The falling raindrop: variations on a theme of Newton,” *Am. J. Phys.* **49**, 113 (1981). The author discusses the problem of mass accretion by a drop falling through a cloud of droplets.

Rabindra Mehta, “Aerodynamics of sports balls” in *Ann. Rev. Fluid Mech.* **17**, 151 (1985).

Neville de Mestre, *The Mathematics of Projectiles in Sport*, Cambridge University Press (1990).

The emphasis of this text is on solving many problems in projectile motion, for example, baseball, basketball, and golf, in the context of mathematical modeling. Many references to the relevant literature are given.

Nancy Roberts, David Andersen, Ralph Deal, Michael Jaret, and William Shaffer, *Introduction to Computer Simulation: The System Dynamics Approach*, Addison-Wesley (1983). A book on computer simulation in the social sciences.

R. R. Rogers, *A Short Course in Cloud Physics*, Pergamon Press (1976).

Emilio Segré, *Nuclei and Particles*, second edition, W. A. Benjamin (1977). Chapter 5 discusses decay cascades. The decay schemes described briefly in Problem 5.8 are taken from C. M. Lederer, J. M. Hollander, and I. Perlman, *Table of Isotopes*, sixth edition, John Wiley & Sons (1967).

It is impossible to list all the excellent books on numerical analysis. We list a few of our favorites.

Forman S. Acton, *Numerical Methods That Work*, Harper & Row (1970); corrected edition, Mathematical Association of America (1990). A somewhat advanced, but clearly written text.

Paul DeVries, *A First Course in Computational Physics*, John Wiley (1994).

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes*, second edition, Cambridge University Press (1992).

Chapter 6

The Chaotic Motion of Dynamical Systems

©2001 by Harvey Gould and Jan Tobochnik
27 March 2001

We study simple deterministic nonlinear models which exhibit complex behavior.

6.1 Introduction

Most natural phenomena are intrinsically nonlinear. Weather patterns and the turbulent motion of fluids are everyday examples. Although we have explored some of the properties of nonlinear physical systems in Chapter 5, it is easier to introduce some of the important concepts in the context of ecology. Our first goal will be to motivate and analyze the one-dimensional difference equation

$$x_{n+1} = 4rx_n(1 - x_n), \quad (6.1)$$

where x_n is the ratio of the population in the n th generation to a reference population. We shall see that the dynamical properties of (6.1) are surprisingly intricate and have important implications for the development of a more general description of nonlinear phenomena. The significance of the behavior of (6.1) is indicated by the following quote from the ecologist Robert May:

“... Its study does not involve as much conceptual sophistication as does elementary calculus. Such study would greatly enrich the student’s intuition about nonlinear systems. Not only in research but also in the everyday world of politics and economics we would all be better off if more people realized that simple nonlinear systems do not necessarily possess simple dynamical properties.”

The study of chaos is currently very popular, but the phenomena is not new and has been of interest to astronomers and mathematicians for about one hundred years. Much of the current

interest is due to the use of the computer as a tool for making empirical observations. We will use the computer in this spirit.

6.2 A Simple One-Dimensional Map

Many biological populations effectively consist of a single generation with no overlap between successive generations. We might imagine an island with an insect population that breeds in the summer and leaves eggs that hatch the following spring. Because the population growth occurs at discrete times, it is appropriate to model the population growth by *difference equations* rather than by differential equations. A simple model of density-independent growth that relates the population in generation $n + 1$ to the population in generation n is given by

$$P_{n+1} = aP_n, \quad (6.2)$$

where P_n is the population in generation n and a is a constant. In the following, we assume that the time interval between generations is unity, and refer to n as the time.

If $a > 1$, each generation will be a times larger than the previous one. In this case (6.2) leads to geometrical growth and an unbounded population. Although the unbounded nature of geometrical growth is clear, it is remarkable that most of us do not integrate our understanding of geometrical growth into our everyday lives. Can a bank pay 4% interest each year indefinitely? Can the world's human population grow at a constant rate forever?

It is natural to formulate a more realistic model in which the population is bounded by the finite carrying capacity of its environment. A simple model of density-dependent growth is

$$P_{n+1} = P_n(a - bP_n). \quad (6.3)$$

Equation (6.3) is nonlinear due to the presence of the quadratic term in P_n . The linear term represents the natural growth of the population; the quadratic term represents a reduction of this natural growth caused, for example, by overcrowding or by the spread of disease.

It is convenient to rescale the population by letting $P_n = (a/b)x_n$ and rewriting (6.3) as

$$x_{n+1} = ax_n(1 - x_n). \quad (6.4)$$

The replacement of P_n by x_n changes the system of units used to define the various parameters. To write (6.4) in the form (6.1), we define the parameter $r = a/4$ and obtain

$$x_{n+1} = f(x_n) = 4rx_n(1 - x_n). \quad (6.5)$$

The rescaled form (6.5) has the desirable feature that its dynamics are determined by a single control parameter r . Note that if $x_n > 1$, x_{n+1} will be negative. To avoid this unphysical feature, we impose the conditions that x is restricted to the interval $0 \leq x \leq 1$ and $0 < r \leq 1$.

Because the function $f(x)$ defined in (6.5) transforms any point on the one-dimensional interval $[0, 1]$ into another point in the same interval, the function f is called a *one-dimensional map*. The form of $f(x)$ in (6.5) is known as the *logistic map*. The logistic map is a simple example of a *dynamical system*, that is, the map is a deterministic, mathematical prescription for finding the future state of a system.

The sequence of values x_0, x_1, x_2, \dots is called the *trajectory* or the *orbit*. To check your understanding, suppose that the initial value of x_0 or *seed* is $x_0 = 0.5$ and $r = 0.2$. Use a calculator to show that the trajectory is $x_1 = 0.2, x_2 = 0.128, x_3 = 0.089293, \dots$ In Figure 6.1 the first thirty iterations of (6.5) are shown for two values of r .

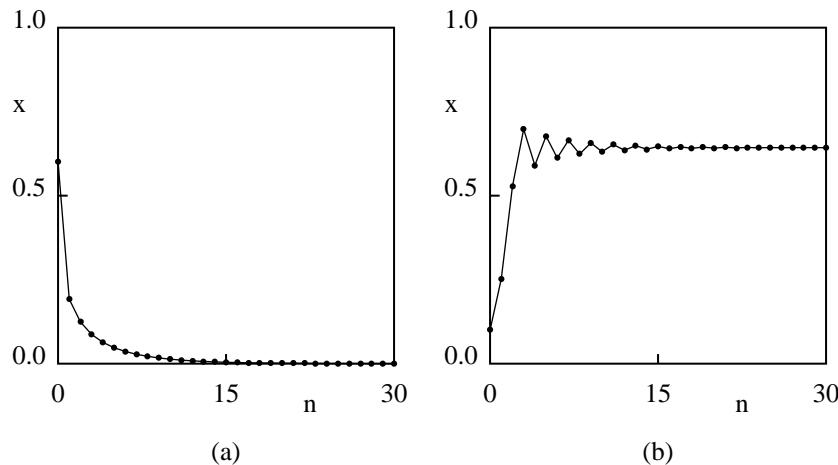


Figure 6.1: (a) Time series for $r = 0.2$ and $x_0 = 0.6$. Note that the stable fixed point is $x = 0$. (b) Time series for $r = 0.7$ and $x_0 = 0.1$. Note the initial transient behavior. The lines between the points are a guide to the eye.

The following class `IterateMap` computes the trajectory of the logistic map (6.5).

```
// updated 3/6/01, 7:54 pm
package edu.clarku.sip.chapter6;
import edu.clarku.sip.plot.*;
import edu.clarku.sip.templates.*;

public class IterateMap implements Model
{
    private double r;
    private double x;
    private int iterations;
    private Control myControl = new SControl(this);
    private Plot plot;

    public IterateMap()
    {
        plot = new Plot("iterations", "x", "Iterate Map");
    }

    public void reset()
```

```

{
    myControl.setValue("r", 0.2 );
    myControl.setValue("x", 0.6);
    myControl.setValue("iterations", 50);
}

public void calculate()
{
    r = myControl.getValue("r");
    x = myControl.getValue("x");
    // method getValue returns double
    iterations = (int) myControl.getValue("iterations");
    plot.deleteAllPoints();
    for (int i = 0; i <= iterations; i++)
    {
        plot.addPoint(0, i, x);    // 0 is data index
        map();
    }
    plot.repaint();
}

public void map()
{
    x = 4*r*x*(1 - x);      // iterate map
}

public static void main(String[] args)
{
    IterateMap map = new IterateMap();
    map.reset();
}
}

```

In Problems 6.1 and 6.3 we use this program to explore the dynamical properties of the logistic map.

Problem 6.1. Exploration of period-doubling

1. Explore the dynamical behavior of (6.5) with $r = 0.24$ for different values of x_0 . Show that $x = 0$ is a *stable fixed point*. That is, for sufficiently small r , the iterated values of x converge to $x = 0$ independently of the value of x_0 . If x represents the population of insects, describe the qualitative behavior of the population.
2. Explore the dynamical behavior of (6.5) for $r = 0.26, 0.5, 0.74$, and 0.748 . A fixed point is *unstable* if for almost all x_0 near the fixed point, the trajectories diverge from it. Verify that $x = 0$ is an unstable fixed point for $r > 0.25$. Show that for the suggested values of r , the iterated values of x do not change after an initial *transient*, that is, the long time dynamical

behavior is *period 1*. In Appendix 6A we show that for $r < 3/4$ and for x_0 in the interval $0 < x_0 < 1$, the trajectories approach the *attractor* at $x = 1 - 1/4r$. The set of initial points that iterate to the attractor is called the *basin* of the attractor. For the logistic map, the interval $0 < x < 1$ is the basin of attraction of the attractor $x = 1 - 1/4r$.

3. Explore the dynamical properties of (6.5) for $r = 0.752, 0.76, 0.8$, and 0.862 . For $r = 0.752$ and 0.862 approximately 1000 iterations are necessary to obtain convergent results. Show that if r is increased slightly beyond 0.75, x oscillates between two values after an initial transient behavior. That is, instead of a stable cycle of period 1 corresponding to one fixed point, the system has a stable cycle of period 2. The value of r at which the single fixed point x^* splits or *bifurcates* into two values x_1^* and x_2^* is $r = b_1 = 3/4$. The pair of x values, x_1^* and x_2^* , form a *stable attractor* of period 2.
4. Describe an ecological scenario of an insect population that exhibits dynamical behavior similar to that observed in part (c).
5. What are the stable attractors of (6.5) for $r = 0.863$ and 0.88 ? What is the corresponding period?
6. What are the stable attractors and corresponding periods for $r = 0.89, 0.891$, and 0.8922 ?

Another way to determine the behavior of (6.5) is to plot the values of x as a function of r (see Figure 6.2). The iterated values of x are plotted after the initial transient behavior is discarded. Such a plot is generated by class **Bifurcate**. For each value of r , the first **ntransient** values of x are computed but not plotted. Then the next **nplot** values of x are plotted, with the first half in red and the second half in blue. This process is repeated for a new value of r until the desired range of r values is reached. A typical value of **ntransient** should be in the range of 100–1000 iterations. The magnitude of **nplot** should be at least as large as the longest period that you wish to observe.

```
// updated 3/6/01, 10:30 pm
package edu.clarku.sip.chapter6;
import edu.clarku.sip.plot.*;
import edu.clarku.sip.templates.*;

public class Bifurcate implements Model
{
    private double r;           // control parameter
    private double rmax;        // maximum value of r (<= 1)
    private double dr;          // incremental change of r, // suggest dr <= 0.01
    private double x;            // initial value
    private double xmax;        // maximum value of x
    private int ntransient;     // number of iterations not plotted
    private int nplot;          // number of iterations plotted
    private int nvalues;        // number of r values plotted
    private Control myControl = new SControl(this);
    private Plot plot;
```

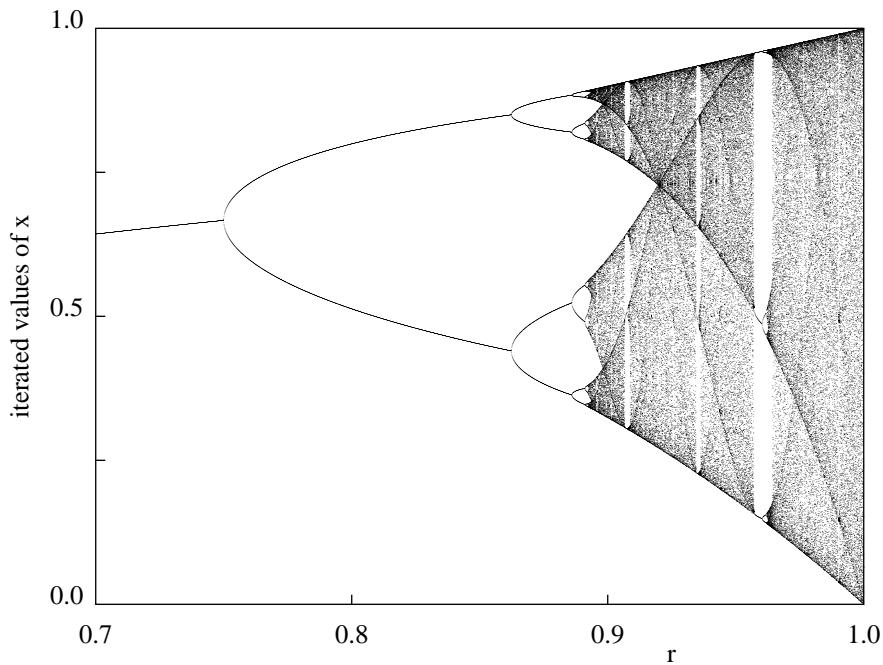


Figure 6.2: Bifurcation diagram of the logistic map. For each value of r , the iterated values of x_n are plotted after the first 1000 iterations are discarded. Note the transition from periodic to chaotic behavior and the narrow windows of periodic behavior within the region of chaos.

```

public Bifurcate()
{
    plot = new Plot("r", "x", "Bifurcate");
}

public void calculate()
{
    r = myControl.getValue("r");
    rmax = myControl.getValue("rmax");
    dr = myControl.getValue("dr");
    ntransient = (int) myControl.getValue("ntransient");
    nplot = (int) myControl.getValue("nplot");
    x = 0.5;
    nvalues = (int)((rmax - r)/dr);
    xmax = 1;
}

```

```

plot.deleteAllPoints();
for (int ir = 0; ir <= nvalues; ir++)
{
    for ( int i = 1; i <= ntransient; i++)      // x values not plotted
        x = f(x,r);
    for (int i = 1; i <= 0.5*nplot; i++)
    {
        x = f(x,r);
        plot.addPoint(0,r,x); // show different x-values for given value of r
    }
    int i  = (int)(0.5*nplot + 1);
    while (i <= nplot)
    {
        x = f(x,r);
        // different data sets automatically in different color
        plot.addPoint (1, r,x); // note different data set than above
        i++;
    }
    r = r + dr;
}
plot.repaint(); // display points after all points have been added
}

public void reset()
{
    myControl.setValue("r", 0.2);
    myControl.setValue("rmax", 0.8);
    myControl.setValue("dr", 0.005);
    myControl.setValue("ntransient", 200);
    myControl.setValue("nplot", 20);
}

public double f(double x,double r)
{
    return 4*r*x*(1 - x);
}

public static void main(String[] args)
{
    Bifurcate b = new Bifurcate();
    b.reset();
}
}

```

Problem 6.2. Qualitative features of the logistic map

1. Use **Bifurcate** to identify period 2, period 4, and period 8 behavior as in Figure 6.2. It

might be necessary to “zoom in” on a portion of the plot. How many period-doublings can you find?

2. Change the scale so that you can follow the iterations of x from period 4 to period 16 behavior. How does the plot look on this scale in comparison to the original scale?
3. Describe the shape of the trajectory near the bifurcations from period $2 \rightarrow$ period 4, period $4 \rightarrow 8$, etc. These bifurcations are frequently called *pitchfork bifurcations*.

The final state or bifurcation diagram in Figure 6.2 indicates that the period-doubling behavior ends at $r \approx 0.892$. This value of r is known very precisely and is given by $r = r_\infty = 0.892486417967\dots$. At $r = r_\infty$, the sequence of period-doublings accumulate to a trajectory of infinite period. In Problem 6.3 we explore the behavior of the trajectories for $r > r_\infty$.

Problem 6.3. The chaotic regime

1. For $r > r_\infty$, two initial conditions that are very close to one another can yield very different trajectories after a small number of iterations. As an example, choose $r = 0.91$ and consider $x_0 = 0.5$ and 0.5001 . How many iterations are necessary for the iterated values of x to differ by more than ten percent? What happens for $r = 0.88$ for the same choice of seeds?
2. The accuracy of floating point numbers retained on a digital computer is finite. To test the effect of the finite accuracy of your computer, choose $r = 0.91$ and $x_0 = 0.5$ and compute the trajectory for 200 iterations. Then modify your program so that after each iteration, the operations $\mathbf{x} = \mathbf{x}/10$ followed by $\mathbf{x} = 10*\mathbf{x}$ are performed. This combination of operations truncates the last digit that your computer retains. Compute the trajectory again and compare your results. Do you find the same discrepancy for $r < r_\infty$?
3. What are the dynamical properties for $r = 0.958$? Can you find other windows of periodic behavior in the interval $r_\infty < r < 1$?

6.3 Period-Doubling

The results of the numerical experiments that we did in Section 6.2 have led us to adopt a new vocabulary to describe our observations and probably have convinced you that the dynamical properties of simple deterministic nonlinear systems can be quite complicated.

To gain more insight into how the dynamical behavior depends on r , we introduce a simple graphical method for iterating (6.5). In Figure 6.3, we show a graph of $f(x)$ versus x for $r = 0.7$. A diagonal line corresponding to $y = x$ intersects the curve $y = f(x)$ at the two fixed points $x^* = 0$ and $x^* = 9/14 \approx 0.642857$. If x_0 is not one of the fixed points, we can find the trajectory in the following manner. Draw a vertical line from $(x = x_0, y = 0)$ to the intersection with the curve $y = f(x)$ at $(x_0, y_0 = f(x_0))$. Next draw a horizontal line from (x_0, y_0) to the intersection with the diagonal line at (y_0, y_0) . On this diagonal line $y = x$, and hence the value of x at this intersection is the first iteration $x_1 = y_0$. The second iteration x_2 can be found in the same way. From the point (x_1, y_0) , draw a vertical line to the intersection with the curve $y = f(x)$. Keep y fixed at $y = y_1 = f(x_1)$, and draw a horizontal line until it intersects the diagonal line; the value of x at this intersection is x_2 . Further iterations can be found by repeating this process.

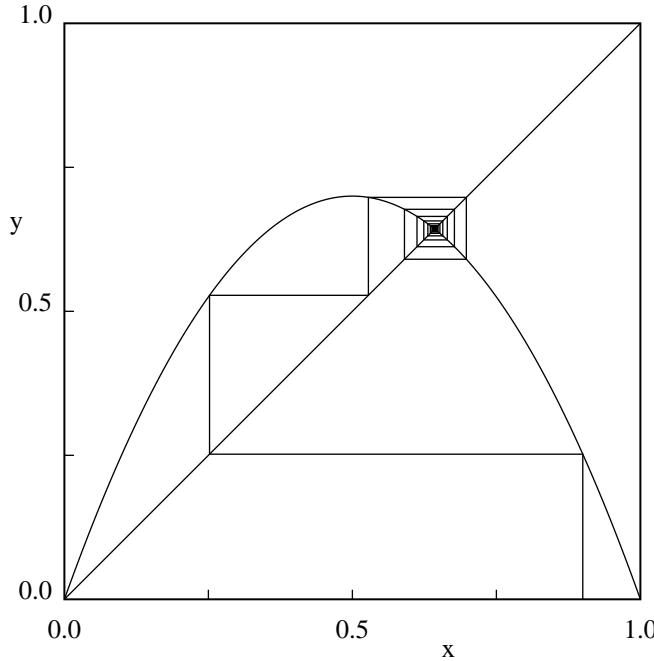


Figure 6.3: Graphical representation of the iteration of the logistic map (6.5) with $r = 0.7$ and $x_0 = 0.9$. Note that the graphical solution converges to the fixed point $x^* \approx 0.643$.

This graphical method is illustrated in Figure 6.3 for $r = 0.7$ and $x_0 = 0.9$. If we begin with any x_0 (except $x_0 = 0$ and $x_0 = 1$), continued iterations will converge to the fixed point $x^* \approx 0.642857$. Repeat the procedure shown in Figure 6.3 by hand and convince yourself that you understand the graphical solution of the iterated values of the map. For this value of r , the fixed point is stable (an attractor of period 1). In contrast, no matter how close x_0 is to the fixed point at $x = 0$, the iterates diverge away from it, and this fixed point is unstable.

How can we explain the qualitative difference between the fixed point at $x = 0$ and $x^* = 0.642857$ for $r = 0.7$? The local slope of the curve $y = f(x)$ determines the distance moved horizontally each time f is iterated. A slope steeper than 45° leads to a value of x further away from its initial value. Hence, the criterion for the stability of a fixed point is that the magnitude of the slope at the fixed point must be less than 45° . That is, if $|df(x)/dx|_{x=x^*}$ is less than unity, then x^* is stable; conversely, if $|df(x)/dx|_{x=x^*}$ is greater than unity, then x^* is unstable. Inspection of $f(x)$ in Figure 6.3 shows that $x = 0$ is unstable because the slope of $f(x)$ at $x = 0$ is greater than unity. In contrast, the magnitude of the slope of $f(x)$ at $x = x^*$ is less than unity and the fixed point is stable. In Appendix 6A, we use similar analytical arguments to show that

$$x^* = 0 \text{ is stable for } 0 < r < 1/4 \quad (6.6a)$$

and

$$x^* = 1 - \frac{1}{4r} \text{ is stable for } 1/4 < r < 3/4. \quad (6.6b)$$

Thus for $0 < r < 3/4$, the eventual behavior after many iterations is known.

What happens if r is greater than $3/4$? From our observations we have found that if r is slightly greater than $3/4$, the fixed point of f becomes unstable and gives birth (bifurcates) to a cycle of period 2. Now x returns to the same value only after every second iteration, and the fixed points of $f(f(x))$ are the attractors of $f(x)$. In the following, we adopt the notation $f^{(2)}(x) = f(f(x))$, and write $f^{(n)}(x)$ for the n th iterate of $f(x)$. (Do not confuse $f^{(n)}(x)$ with the n th derivative of $f(x)$.) For example, the second iterate $f^{(2)}(x)$ is given by the fourth-order polynomial:

$$f^{(2)}(x) = 16r^2x(1-x)[1 - 4rx(1-x)] \quad (6.7)$$

$$= 16r^2x[-4rx^3 + 8rx^2 - (1+4r)x + 1]. \quad (6.8)$$

What happens if we increase r still further? Eventually the magnitude of the slope of the fixed points of $f^{(2)}(x)$ exceeds unity and the fixed points of $f^{(2)}(x)$ become unstable. Now the cycle of f is period 4, and we can study the stability of the fixed points of the fourth iterate $f^{(4)}(x) = f^{(2)}(f^{(2)}(x)) = f(f(f(f(x))))$. These fixed points also eventually bifurcate, and we are led to the phenomena of *period-doubling* as we observed in Problem 6.2.

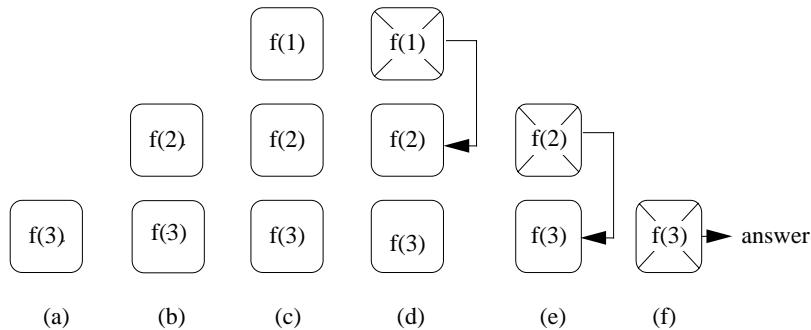


Figure 6.4: Example of the calculation of $f(0.4, 0.8, 3)$ using the recursive function defined in `GraphicalSolution`. The number in each box is the value of the variable `iterate`. The values of $x = 0.4$ and $r = 0.8$ are not shown. The value of $f(x, r, 3) = 0.7842$.

Class `GraphicalSolution` implements the graphical analysis of $f(x)$. The n th order iterates are defined in `f(x,r,iterate)` using *recursion*. (The quantity `iterate` is 1, 2, and 4 for the functions $f(x)$, $f^{(2)}(x)$, and $f^{(4)}(x)$ respectively.) Recursion is an idea that is simple once you understand it, but it can be difficult to grasp the idea initially. One way to understand how recursion works is to think of a stack, such as a stack of trays in a cafeteria. The first time a recursive function is called, the function is placed on the top of the stack. Each time the function calls itself, an exact copy of the function, with possibly different values of the input parameters, is placed on top of the stack. When a copy of the function is finished, this copy is popped off the top of the stack. To understand the function `f(x,r,iterate)`, suppose we want to compute $f(0.4, 0.8, 3)$. First we write $f(0.4, 0.8, 3)$ on a piece of paper (see Figure 6.4a). Follow the statements within the function until another call to `f(0.4, 0.8, iterate)` occurs. In this case,

the call is to $f(0.4, 0.8, \text{iterate}-1)$ which equals $f(0.4, 0.8, 2)$. Write $f(0.4, 0.8, 2)$ above $f(0.4, 0.8, 3)$ (see Figure 6.4b). When you come to the end of the definition of the function, write down the value of f that is actually returned, and remove the function from the stack by crossing it out (see Figure 6.4d). This returned value for f equals y if $\text{iterate} > 1$, or it is the output of the function for $\text{iterate} = 1$. Continue deleting copies of f as they are finished, until there are no copies left on the paper. The final value of f is the value returned by the computer. Write a miniprogram that defines $f(x, r, \text{iterate})$ and prints the value of $f(0.4, 0.8, 3)$. Is the answer the same as your hand calculation?

```
// 03/05/01 4:30 pm
package edu.clarku.sip.chapter6;
import edu.clarku.sip.plot.*;
import edu.clarku.sip.templates.*;

public class GraphicalSolution implements AnimationModel, Runnable
{
    private double r;
    private int iterate;
    private Control myControl = new SAnimationControl(this);
    private Plot plot;
    private double x, y;
    private Thread animationThread;
    double x0, y0;

    public GraphicalSolution()
    {
        plot = new Plot("iterations", "x", "Graph Sol");
        plot.setPlotType(1, Plot.LINE);
        plot.setAutoUpdate(true);
    }

    public void reset()
    {
        myControl.setValue("r", 0.89 );
        myControl.setValue("x", 0.2);
        myControl.setValue("iterate", 1);
    }

    public void stopCalculation()
    {
        animationThread = null;
    }

    public void clear(){}
    public void continueCalculation(){}
    public void startCalculation()
```

```

{
    r = myControl.getValue("r"); //control parameter r
    x = myControl.getValue("x");
    x0 = x;
    iterate = (int) myControl.getValue("iterate"); // iterate of f(x)
    plot.deleteAllPoints();
    drawFunction();
    y0 = 0;
    x = x0;
    animationThread = new Thread(this);
    animationThread.start();
}

public void run()
{
    while(animationThread != null)
    {
        step();
        try
        {
            Thread.sleep(100);
        }
        catch(InterruptedException e){}
    }
}

public void step()
{
    y = f(x,r,iterate);
    plot.addPoint(1,x0,y0);
    plot.addPoint(1,x0,y);
    plot.addPoint(1,y,y);
    x0 = y;
    y0 = y;
    x = y;
}

void drawFunction()
{
    double nplot = 200;           // # of points at which function computed
    double delta = 1/nplot;
    x = 0;
    y = 0;
    for (int i = 0; i<= nplot; i++)
    {
        y = f(x,r,iterate);
}

```

```

        plot.addPoint(0, x,y);
        x = x + delta;
    }
}

double f(double x, double r, double iterate) // f defined by recursive procedure
{
    if (iterate > 1)
    {
        double y = f(x,r,iterate - 1);
        return 4*r*y*(1 - y);
    }
    else
        return 4*r*x*(1 - x);
}

public static void main(String[] args)
{
    GraphicalSolution d = new GraphicalSolution();
    d.reset();
}
}
}

```

Problem 6.4. Qualitative properties of the fixed points

1. Use **GraphicalSolution** to show graphically that there is a single stable fixed point of $f(x)$ for $r < 3/4$. It would be instructive to insert a pause between each iteration of the map and to show the value of the slope at $y_n = f(x_n)$ in a separate window. At what value of r does the absolute value of this slope exceed unity? Let b_1 denote the value of r at which the fixed point of $f(x)$ bifurcates and becomes unstable. Verify that $b_1 = 0.75$.
2. Describe the trajectory of $f(x)$ for $r = 0.785$. What is the nature of the fixed point given by $x = 1 - 1/4r$? What is the nature of the trajectory if $x_0 = 1 - 1/4r$? What is the period of $f(x)$ for all other choices of x_0 ? What are the numerical values of the two-point attractor?
3. The function $f(x)$ is symmetrical about $x = \frac{1}{2}$ where $f(x)$ is a maximum. What are the qualitative features of the second iterate $f^{(2)}(x) = f(f(x))$ for $r = 0.785$? Is $f^{(2)}(x)$ symmetrical about $x = \frac{1}{2}$? For what value of x does $f^{(2)}(x)$ have a minimum? Iterate $x_{n+1} = f^{(2)}(x_n)$ for $r = 0.785$ and find its two fixed points x_1^* and x_2^* . (Try $x_0 = 0.1$ and $x_0 = 0.3$.) Are the fixed points of $f^{(2)}(x)$ stable or unstable? How do these values of x_1^* and x_2^* compare with the values of the two-point attractor of $f(x)$? Verify that the slopes of $f^{(2)}(x)$ at x_1^* and x_2^* are equal.
4. Verify the following properties of the fixed points of $f^{(2)}(x)$. As r is increased, the fixed points of $f^{(2)}(x)$ move apart and the slope of $f^{(2)}(x)$ at the fixed points decreases. What is the value of $r = s_2$ at which one of the two fixed points of $f^{(2)}$ equals $\frac{1}{2}$? What is the value of the other fixed point? What is the slope of $f^{(2)}(x)$ at $x = \frac{1}{2}$? What is the slope

at the other fixed point? As r is further increased, the slopes at the fixed points become negative. Finally at $r = b_2 \approx 0.8623$, the slopes at the two fixed points of $f^{(2)}(x)$ equal -1 , and the two fixed points of $f^{(2)}$ become unstable. (It can be shown that the exact value of b_2 is $b_2 = (1 + \sqrt{6})/4$.)

5. Show that for r slightly greater than b_2 , for example, $r = 0.87$, there are four stable fixed points of the function $f^{(4)}(x)$. What is the value of $r = s_3$ when one of the fixed points equals $\frac{1}{2}$? What are the values of the three other fixed points at $r = s_3$?
6. Estimate the value of $r = b_3$ at which the four fixed points of $f^{(4)}$ become unstable.
7. Choose $r = s_3$ and estimate the number of iterations that are necessary for the trajectory to converge to period 4 behavior. How does this number of iterations change when neighboring values of r are considered? Choose several values of x_0 so that your results do not depend on the initial conditions.

Problem 6.5. Periodic windows in the chaotic regime

1. If you look closely at the bifurcation diagram in Figure 6.2, you will see that the region of chaotic behavior for $r > r_\infty$ is interrupted by intervals of periodic behavior. Magnify your bifurcation diagram so that you can look at the interval $0.957107 \leq r \leq 0.960375$, where a periodic trajectory of period 3 occurs. (Period 3 behavior starts at $r = (1 + \sqrt{8})/4$.) What happens to the trajectory for slightly larger r , for example, for $r = 0.9604$?
2. Plot the map $f^{(3)}(x)$ versus x at $r = 0.96$, a value of r in the period 3 window. Draw the line $y = x$ and determine the intersections with $f^{(3)}(x)$. (Use **GraphicalSolution** without calling method **trajectory**.) The stable fixed points satisfy the condition $x^* = f^{(3)}(x^*)$. Because $f^{(3)}(x)$ is an eighth-order polynomial, there are eight solutions (including $x = 0$). Find the intersections of $f^{(3)}(x)$ with $y = x$ and identify the three stable fixed points. What are the slopes of $f^{(3)}(x)$ at these points? Then decrease r to $r = 0.957107$, the (approximate) value of r below which the system is chaotic. Draw the line $y = x$ and determine the number of intersections with $f^{(3)}(x)$. Note that at this value of r , the curve $y = f^{(3)}(x)$ is tangent to the diagonal line at the three stable fixed points. For this reason, this type of transition is called a *tangent bifurcation*. Note that there also is an unstable point at $x \approx 0.76$.
3. Plot $x_{n+1} = f^{(3)}(x_n)$ versus n for $r = 0.9571$, a value of r just below the onset of period 3 behavior. How would you describe the behavior of the trajectory? This type of chaotic motion is an example of *intermittency*, that is, nearly periodic behavior interrupted by occasional irregular bursts.
4. Modify **GraphicalSolution** so that you can study the graphical solution of $x_{n+1} = f^{(3)}(x_n)$ for the same value of r as in part (c). That is, “zoom in” on the values of x near the stable fixed points that you found in part (b) for r in the period 3 regime. Note the three narrow channels between the diagonal line $y = x$ and the plot of $f^{(3)}(x)$. The trajectory requires many iterations to squeeze through the channel, and we see period 3 behavior during this time. Eventually, the trajectory escapes from the channel and bounces around until it is sent into a channel at some unpredictable later time.

k	b_k
1	0.750 000
2	0.862 372
3	0.886 023
4	0.891 102
5	0.892 190
6	0.892 423
7	0.892 473
8	0.892 484

Table 6.1: Values of the control parameter b_k for the onset of the k th bifurcation. Six decimal places are shown.

6.4 Universal Properties and Self-Similarity

In Sections 6.2 and 6.3 we found that the trajectory of the logistic map has remarkable properties as a function of the control parameter r . In particular, we found a sequence of period-doublings accumulating to a chaotic trajectory of infinite period at $r = r_\infty$. For most values of $r > r_\infty$, we saw that the trajectory is very sensitive to the initial conditions. We also found “windows” of period 3, 6, 12, … embedded in the broad regions of chaotic behavior. How typical is this type of behavior? In the following, we will find further numerical evidence that the general behavior of the logistic map is independent of the details of the form (6.5) of $f(x)$.

You might have noticed that the range of r between successive bifurcations becomes smaller as the period increases (see Table 6.1). For example, $b_2 - b_1 = 0.112398$, $b_3 - b_2 = 0.023624$, and $b_4 - b_3 = 0.00508$. A good guess is that the decrease in $b_k - b_{k-1}$ is geometric, that is, the ratio $(b_k - b_{k-1})/(b_{k+1} - b_k)$ is a constant. You can check that this ratio is not exactly constant, but converges to a constant with increasing k . This behavior suggests that the sequence of values of b_k has a limit and follows a geometrical progression:

$$b_k \approx r_\infty - \text{constant} \delta^{-k}, \quad (6.9)$$

where δ is known as the Feigenbaum number. From (6.9) it is easy to show that δ is given by the ratio

$$\delta = \lim_{k \rightarrow \infty} \frac{b_k - b_{k-1}}{b_{k+1} - b_k}. \quad (6.10)$$

Problem 6.6. Estimation of the Feigenbaum constant

- Plot $\delta_k = (b_k - b_{k-1})/(b_{k+1} - b_k)$ versus k using the values of b_k in Table 6.1 and estimate the value of δ . Are the number of decimal places given in Table 6.1 for b_k sufficient for all the values of k shown? The best estimate of δ is

$$\delta = 4.669\,201\,609\,102\,991\dots \quad (6.11)$$

The number of decimal places in (6.11) is shown to indicate that δ is known precisely. Use (6.9) and (6.11) and the values of b_k to estimate the value of r_∞ .

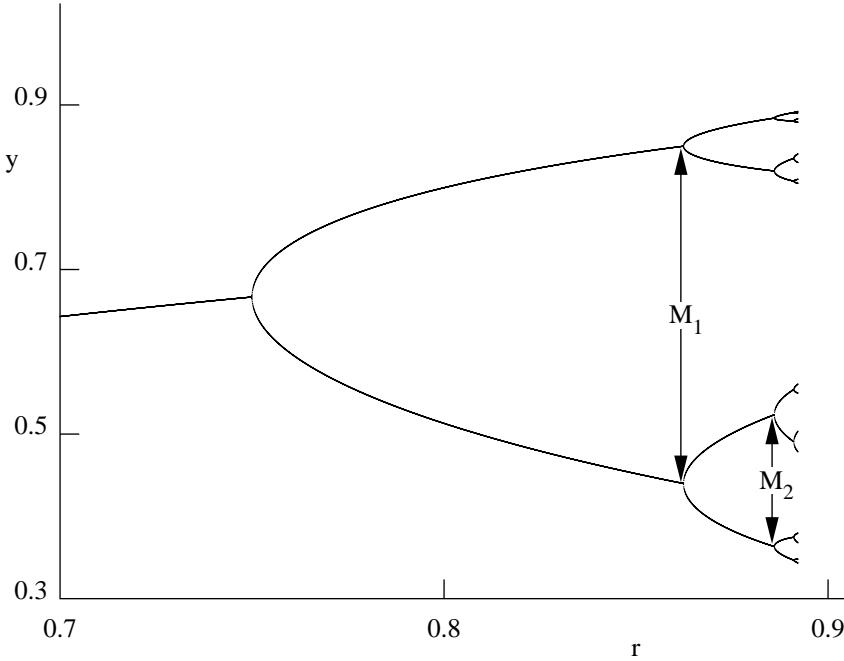


Figure 6.5: The first few bifurcations of the logistic equation showing the scaling of the maximum distance M_k between the asymptotic values of x describing the bifurcation.

2. In Problem 6.4 we found that one of the four fixed points of $f^{(4)}(x)$ is at $x^* = \frac{1}{2}$ for $r = s_3 \approx 0.87464$. We also found that the convergence to the fixed points of $f^{(4)}(x)$ is more rapid than at nearby values of r . In Appendix 6A we show that these *superstable* trajectories occur whenever one of the fixed points is at $x = \frac{1}{2}$. The values of $r = s_m$ that give superstable trajectories of period 2^{m-1} are much better defined than the points of bifurcation, $r = b_k$. The rapid convergence to the final trajectories also gives better numerical estimates, and we always know one member of the trajectory, namely $x = \frac{1}{2}$. It is reasonable that δ can be defined as in (6.10) with b_k replaced by s_m . Use the values of $s_1 = 0.5, s_2 \approx 0.809017$, and $s_3 = 0.874640$ to estimate δ . The numerical values of s_m are found in Project 6.19 by solving the equation $f^{(m)}(x = \frac{1}{2}) = \frac{1}{2}$ numerically; the first eight values of s_m are listed in Table 6.2.

We can associate another number with the series of pitchfork bifurcations. From Figure 6.3 and Figure 6.5 we see that each pitchfork bifurcation gives birth to “twins” with the new generation more densely packed than the previous generation. One measure of this density is the maximum distance M_k between the values of x describing the bifurcation (see Figure 6.5). The disadvantage of using M_k is that the transient behavior of the trajectory is very long at the boundary between two different periodic behaviors. A more convenient measure of the density is the quantity $d_k = x_k^* - \frac{1}{2}$, where x_k^* is the value of the fixed point nearest to the fixed point $x^* = \frac{1}{2}$. The first two values of d_k are shown in Figure 6.6 with $d_1 \approx 0.3090$ and $d_2 \approx -0.1164$. The next value is $d_3 \approx 0.0460$. Note that the fixed point nearest to $x = \frac{1}{2}$ alternates from one side of $x = \frac{1}{2}$ to the other. We

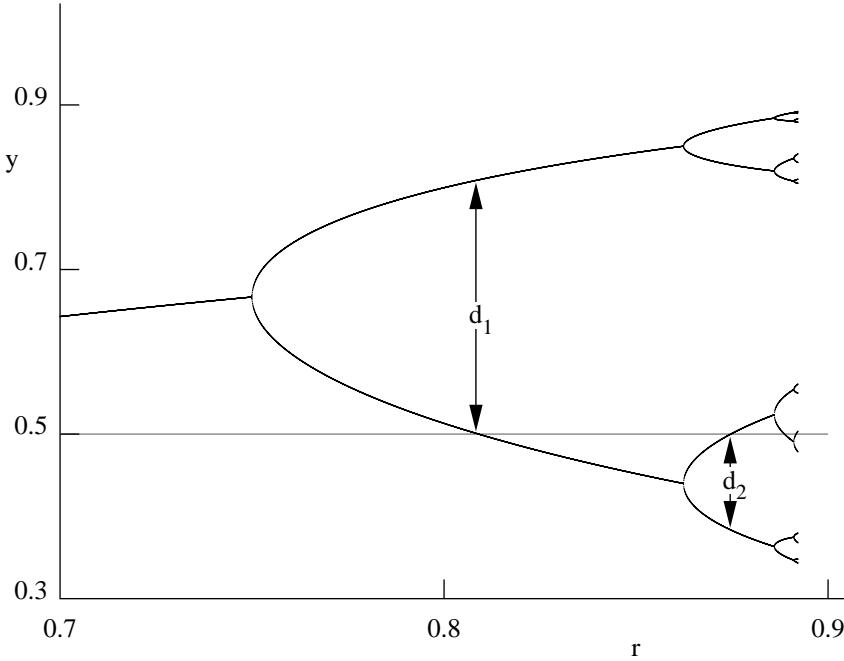


Figure 6.6: The quantity d_k is the distance from $x^* = 1/2$ to the nearest element of the attractor of period 2^k . It is convenient to use this quantity to determine the exponent α .

define the quantity α by the ratio

$$\alpha = \lim_{k \rightarrow \infty} -\left(\frac{d_k}{d_{k+1}}\right). \quad (6.12)$$

The estimates $\alpha = 0.3090/0.1164 = 2.65$ for $k = 1$ and $\alpha = 0.1164/0.0460 = 2.53$ for $k = 2$ are consistent with the asymptotic limit $\alpha = 2.5029078750958928485\dots$

We now give qualitative arguments that suggest that the general behavior of the logistic map in the period-doubling regime is independent of the detailed form of $f(x)$. As we have seen, period-doubling is characterized by self-similarities, for example, the period-doublings look similar except for a change of scale. We can demonstrate these similarities by comparing $f(x)$ for $r = s_1 = 0.5$ for the superstable trajectory with period 1 to the function $f^{(2)}(x)$ for $r = s_2 \approx 0.809017$ for the superstable trajectory of period 2 (see Figure 6.7). The function $f(x, r = s_1)$ has unstable fixed points at $x = 0$ and $x = 1$ and a stable fixed point at $x = \frac{1}{2}$. Similarly the function $f^{(2)}(x, r = s_2)$ has a stable fixed point at $x = \frac{1}{2}$ and an unstable fixed point at $x \approx 0.69098$. Note the similar shape, but different scale of the curves in the square box in part (a) and part (b) of Figure 6.7. This similarity is an example of scaling. That is, if we scale $f^{(2)}$ and change (renormalize) the value of r , we can compare $f^{(2)}$ to f . (See Chapter 13 for a discussion of scaling and renormalization in another context.)

Our graphical comparison is meant only to be suggestive. A precise approach shows that if

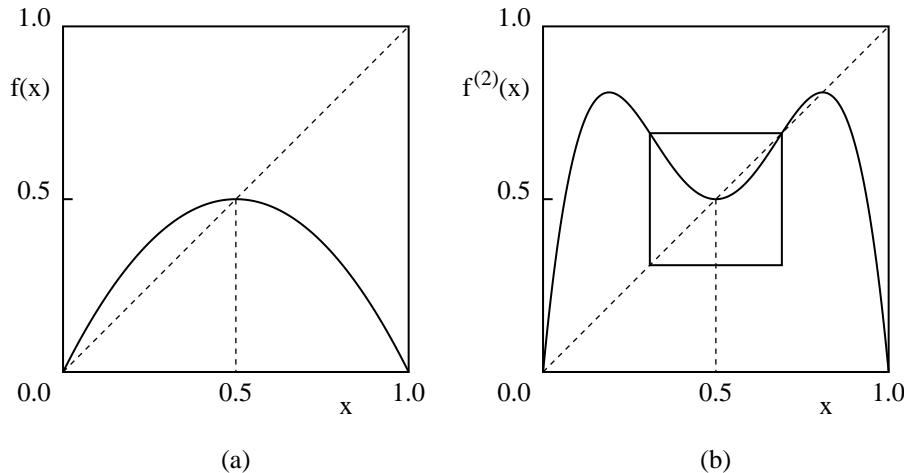


Figure 6.7: Comparison of $f(x, r)$ for $r = s_1$ with the second iterate $f^{(2)}(x)$ for $r = s_2$. (a) The function $f(x, r = s_1)$ has unstable fixed points at $x = 0$ and $x = 1$ and a stable fixed point at $x = \frac{1}{2}$. (b) The function $f^{(2)}(x, r = s_1)$ has a stable fixed point at $x = \frac{1}{2}$. The unstable fixed point of $f^{(2)}(x)$ nearest to $x = \frac{1}{2}$ occurs at $x \approx 0.69098$, where the curve $f^{(2)}(x)$ intersects the line $y = x$. The upper right-hand corner of the square box in (b) is located at this point, and the center of the box is at $(\frac{1}{2}, \frac{1}{2})$. Note that if we reflect this square about the point $(\frac{1}{2}, \frac{1}{2})$, the shape of the reflected graph in the square box is nearly the same as it is in part (a), but on a smaller scale.

we continue the comparison of the higher-order iterates, for example, $f^{(4)}(x)$ to $f^{(2)}(x)$, etc., the superposition of functions converges to a universal function that is independent of the form of the original function $f(x)$.

Problem 6.7. Further estimates of the exponents α and δ

1. Write a subroutine to find the appropriate scaling factor and superimpose f and the rescaled form of $f^{(2)}$ found in Figure 6.7.
 2. Use arguments similar to those discussed in the text in Figure 6.7 and compare the behavior of $f^{(4)}(x, r = s_3)$ in the square about $x = \frac{1}{2}$ with $f^{(2)}(x, r = s_2)$ in its square about $x = \frac{1}{2}$. The size of the squares are determined by the unstable fixed point nearest to $x = \frac{1}{2}$. Find the appropriate scaling factor and superimpose $f^{(2)}$ and the rescaled form of $f^{(4)}$.

It is easy to modify your programs to consider other one-dimensional maps. In Problem 6.8 we consider several one-dimensional maps and determine if they also exhibit the period-doubling route to chaos.

** Problem 6.8.* Other one-dimensional maps

Determine the qualitative properties of the one-dimensional maps:

$$f(x) = x e^{r(1-x)} \quad (6.13)$$

$$f(x) = r \sin \pi x. \quad (6.14)$$

The map in (6.13) has been used by ecologists (cf. May) to study a population that is limited at high densities by the effect of epidemic disease. Although it is more complicated than (6.5), its advantage is that the population remains positive no matter what (positive) value is taken for the initial population. There are no restrictions on the maximum value of r , but if r becomes sufficiently large, x eventually becomes effectively zero, rendering the population extinct. What is the behavior of the time series of (6.13) for $r = 1.5, 2$, and 2.7 ? Describe the qualitative behavior of $f(x)$. Does it have a maximum?

The sine map (6.14) with $0 < r \leq 1$ and $0 \leq x \leq 1$ has no special significance, except that it is nonlinear. If time permits, estimate the value of δ for both maps. What limits the accuracy of your determination of δ ?

The above qualitative arguments and numerical results suggest that the quantities α and δ are *universal*, that is, independent of the detailed form of $f(x)$. In contrast, the values of the accumulation point r_∞ and the constant in (6.9) depend on the detailed form of $f(x)$. Feigenbaum has shown that the period-doubling route to chaos and the values of δ and α are universal property of maps that have a quadratic maximum, that is, $f'(x)|_{x=x_m} = 0$ and $f''(x)|_{x=x_m} < 0$.

Why is the universality of period-doubling and the numbers δ and α more than a curiosity? The reason is that because this behavior is independent of the details, there might exist realistic systems whose underlying dynamics yield the same behavior as the logistic map. Of course, most physical systems are described by differential rather than difference equations. Can these systems exhibit period-doubling behavior? Several workers (cf. Testa et al.) have constructed nonlinear RLC circuits driven by an oscillatory source voltage. The output voltage shows bifurcations, and the measured values of the exponents δ and α are consistent with the predictions of the logistic map.

Of more general interest is the nature of turbulence in fluid systems. Consider a stream of water flowing past several obstacles. We know that at low flow speeds, the water flows past obstacles in a regular and time-independent fashion, called *laminar* flow. As the flow speed is increased (as measured by a dimensionless parameter called the Reynolds number), some swirls develop, but the motion is still time-independent. As the flow speed is increased still further, the swirls break away and start moving downstream. The flow pattern as viewed from the bank becomes time-dependent. For still larger flow speeds, the flow pattern becomes very complex and looks random. We say that the flow pattern has made a transition from laminar flow to *turbulent* flow.

This qualitative description of the transition to chaos in fluid systems is superficially similar to the description of the logistic map. Can fluid systems be analyzed in terms of the simple models of the type we have discussed here? In a few instances such as turbulent convection in a heated saucepan, period doubling and other types of transitions to turbulence have been observed. The type of theory and analysis we have discussed has suggested new concepts and approaches, and the study of turbulent flows is a subject of much current research.

6.5 Measuring Chaos

How do we know if a system is chaotic? The most important characteristic of chaos is *sensitivity to initial conditions*. In Problem 6.3 for example, we found that the trajectories starting from $x_0 = 0.5$ and $x_0 = 0.5001$ for $r = 0.91$ become very different after a small number of iterations. Because computers only store floating numbers to a certain number of digits, the implication of this result is that our numerical predictions of the trajectories are restricted to small time intervals. That is, sensitivity to initial conditions implies that even though the logistic map is deterministic, our ability to make numerical predictions is limited.

How can we quantify this lack of predictability? In general, if we start two identical dynamical systems from different initial conditions, we expect that the difference between the trajectories will change as a function of n . In Figure 6.8 we show a plot of the difference $|\Delta x_n|$ versus n for the same conditions as in Problem 6.3a. We see that roughly speaking, $\ln |\Delta x_n|$ is a linearly increasing function of n . This result indicates that the separation between the trajectories grows exponentially if the system is chaotic. This divergence of the trajectories can be described by the *Lyapunov exponent*, which is defined by the relation:

$$|\Delta x_n| = |\Delta x_0| e^{\lambda n}, \quad (6.15)$$

where Δx_n is the difference between the trajectories at time n . If the Lyapunov exponent λ is positive, then nearby trajectories diverge exponentially. Chaotic behavior is characterized by exponential divergence of nearby trajectories.

A naive way of measuring the Lyapunov exponent λ is to run the same dynamical system twice with slightly different initial conditions and measure the difference of the trajectories as a function of n . We used this method to generate Figure 6.8. Because the rate of separation of the trajectories might depend on the choice of x_0 , a better method would be to compute the rate of separation for many values of x_0 . This method would be tedious, because we would have to fit the separation to (6.15) for each value of x_0 and then determine an average value of λ .

A more important limitation of the naive method is that because the trajectory is restricted to the unit interval, the separation $|\Delta x_n|$ ceases to increase when n becomes sufficiently large. However, to make the computation of λ as accurate as possible, we would like to average over as many iterations as possible. Fortunately, there is a better procedure. To understand the procedure, we take the natural logarithm of both sides of (6.15) and write λ as

$$\lambda = \frac{1}{n} \ln \left| \frac{\Delta x_n}{\Delta x_0} \right|. \quad (6.16)$$

Because we want to use the data from the entire trajectory after the transient behavior has ended, we use the fact that

$$\frac{\Delta x_n}{\Delta x_0} = \frac{\Delta x_1}{\Delta x_0} \frac{\Delta x_2}{\Delta x_1} \cdots \frac{\Delta x_n}{\Delta x_{n-1}}. \quad (6.17)$$

Hence, we can express λ as

$$\lambda = \frac{1}{n} \sum_{i=0}^{n-1} \ln \left| \frac{\Delta x_{i+1}}{\Delta x_i} \right|. \quad (6.18)$$

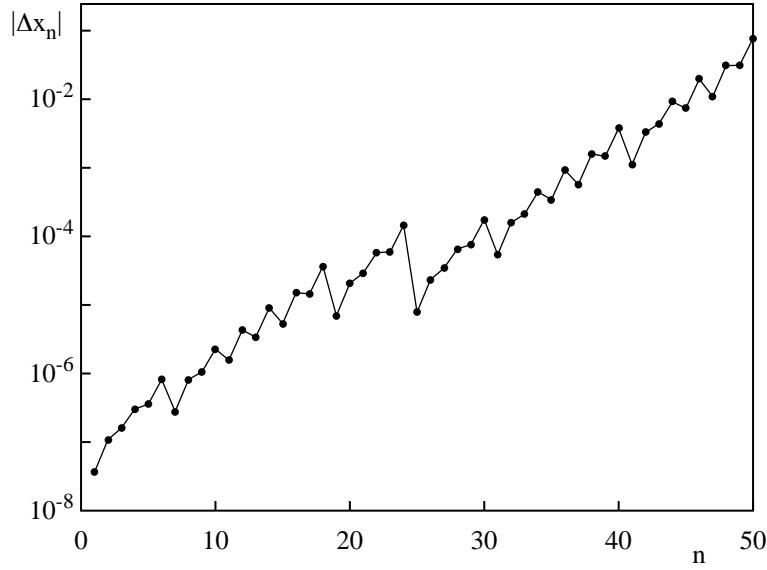


Figure 6.8: The evolution of the difference Δx_n between the trajectories of the logistic map at $r = 0.91$ for $x_0 = 0.5$ and $x_0 = 0.5001$. The separation between the two trajectories increases with n , the number of iterations, if n is not too large. (Note that $|\Delta x_1| \sim 10^{-8}$ and that the trend is not monotonic.)

The form (6.18) implies that we can consider x_i for any i as the initial condition.

We see from (6.18) that the problem of computing λ has been reduced to finding the ratio $\Delta x_{i+1}/\Delta x_i$. Because we want to make the initial difference between the two trajectories as small as possible, we are interested in the limit $\Delta x_i \rightarrow 0$. The idea of the more sophisticated procedure is to compute the differential dx_i from the equation of motion at the same time that the equation of motion is being iterated. We use the logistic map as an example. The differential of (6.5) can be written as

$$\frac{dx_{i+1}}{dx_i} = f'(x_i) = 4r(1 - 2x_i). \quad (6.19)$$

We can consider x_i for any i as the initial condition and the ratio dx_{i+1}/dx_i as a measure of the rate of change of x_i . Hence, we can iterate the logistic map as before and use the values of x_i and the relation (6.19) to compute dx_{i+1}/dx_i at each iteration. The Lyapunov exponent is given by

$$\lambda = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=0}^{n-1} \ln |f'(x_i)|, \quad (6.20)$$

where we begin the sum in (6.20) after the transient behavior is completed. We have included explicitly the limit $n \rightarrow \infty$ in (6.20) to remind ourselves to choose n sufficiently large. Note that this procedure weights the points on the attractor correctly, that is, if a particular region of the attractor is not visited often by the trajectory, it does not contribute much to the sum in (6.20).

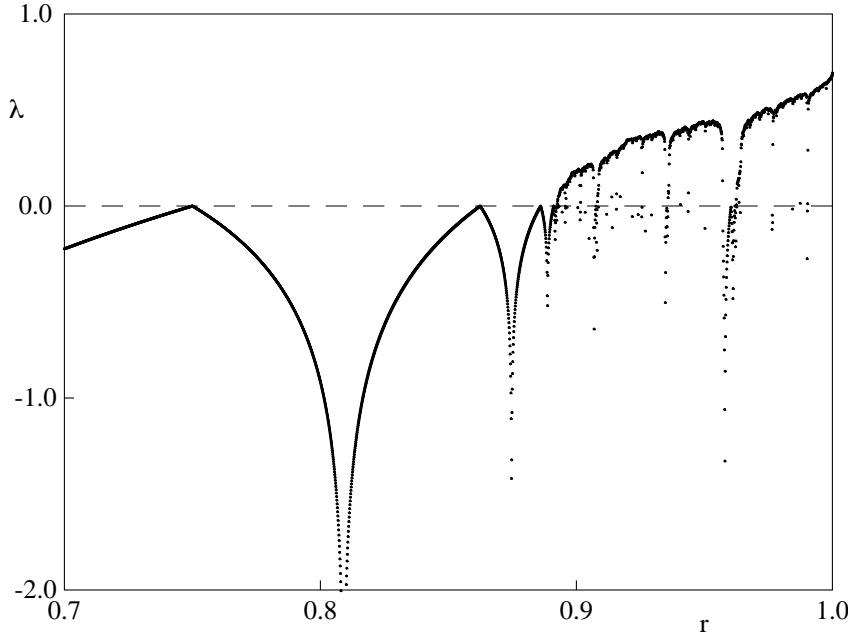


Figure 6.9: The Lyapunov exponent calculated using the method in (6.20) as a function of the control parameter r . Compare the behavior of λ to the bifurcation diagram in Figure 6.2. Note that $\lambda < 0$ for $r < 3/4$ and approaches zero at a period doubling bifurcation. A negative spike corresponds to a superstable trajectory. The onset of chaos is visible near $r = 0.892$, where λ first becomes positive. For $r > 0.892$, λ generally increases except for dips below zero whenever a periodic window occurs. Note the large dip due to the period 3 window near $r = 0.96$. For each value of r , the first 1000 iterations were discarded, and 10^5 values of $\ln |f'(x_n)|$ were used to determine λ .

Problem 6.9. Lyapunov exponent for the logistic map

1. Compute the Lyapunov exponent λ for the logistic map using the naive approach. Choose $r = 0.91$, $x_0 = 0.5$, and $\Delta x_0 = 10^{-6}$, and plot $\ln |\Delta x_n / \Delta x_0|$ versus n . What happens to $\ln |\Delta x_n / \Delta x_0|$ for large n ? Estimate λ for $r = 0.91$, $r = 0.97$, and $r = 1.0$. Does your estimate of λ for each value of r depend significantly on your choice of x_0 or Δx_0 ?
2. Compute λ using the algorithm discussed in the text for $r = 0.76$ to $r = 1.0$ in steps of $\Delta r = 0.01$. What is the sign of λ if the system is not chaotic? Plot λ versus r , and explain your results in terms of behavior of the bifurcation diagram shown in Figure 6.2. Compare your results for λ with those shown in Figure 6.9. How does the sign of λ correlate with the behavior of the system as seen in the bifurcation diagram? If $\lambda < 0$, then the two trajectories converge and the system is not chaotic. If $\lambda = 0$, then the trajectories diverge algebraically, that is, as a power of n . For what value of r is λ a maximum?

3. In Problem 6.3b we saw that roundoff errors in the chaotic regime make the computation of individual trajectories meaningless. That is, if the system's behavior is chaotic, then small roundoff errors are amplified exponentially in time, and the actual numbers we compute for the trajectory starting from a given initial value are not "real." Given this limitation, how meaningful is our computation of the Lyapunov exponent? Repeat your calculation of λ for $r = 1$ by changing the roundoff error as you did in Problem 6.3b. Does your computed value of λ change? We will encounter a similar question in Chapter 8 where we compute the trajectories of a system of many particles. The answer appears to be that although the trajectory we compute is not the one we thought we were trying to compute, the computed trajectory is close to a possible trajectory of the system. Quantities such as λ that are averaged over many possible trajectories are independent of the detailed behavior of an individual trajectory.

6.6 Controlling Chaos

The dream of classical physics has been that if the initial conditions and all the forces acting on a system are known, then we can predict the future with as much precision as we desire. The existence of chaos has shattered that dream, not just for some esoteric systems, but for the majority of dynamical systems observed in nature. However, even if a system is chaotic, we still might be able to control its behavior with small, but carefully chosen perturbations of the system. We will illustrate the method for the logistic map. The application of the method to other one-dimensional systems is straightforward, but the extension to higher dimensional systems is more complicated (cf. Ott, Lai).

Suppose that we want the logistic map to have periodic behavior even though the parameter r equals a value in the chaotic regime. How can we make the trajectory have periodic behavior without drastically changing r or imposing an external perturbation that is so large that the internal dynamics of the map become irrelevant? The key to the solution is that for any value of r in the chaotic regime, there is an infinite number of trajectories that have unstable periods. This property of the chaotic regime means that if we choose the value of the seed x_0 to be precisely equal to a point on an unstable trajectory with period p , the subsequent trajectory will have this period. However, if we choose a value of x_0 that differs ever so slightly from this special value, the trajectory will not be periodic. Our goal is to make slight perturbations to the system to keep it on the desired unstable periodic trajectory.

The first step is to find the values of $x(i)$, $i = 1$ to p , that constitute the unstable periodic trajectory. It is an interesting numerical problem to find the values of $x(i)$, and we consider this problem first. The trick is to find a fixed point of the map $f^{(p)}$. That is, we need to find the value of x^* such that

$$g^{(p)}(x^*) \equiv f^{(p)}(x^*) - x^* = 0. \quad (6.21)$$

The algorithms for finding the solution to (6.21) are called root finding algorithms. You might have heard of Newton's method, which we will describe in Chapter 10. Here we use the simplest root-finding algorithm, the *bisection* method. The algorithm works as follows:

1. Choose two values, x_{left} and x_{right} , with $x_{\text{left}} < x_{\text{right}}$, such that the product $g^{(p)}(x_{\text{left}})g^{(p)}(x_{\text{right}}) < 0$. There must be a value of x such that $g^{(p)}(x) = 0$ in the interval $[x_{\text{left}}, x_{\text{right}}]$.
2. Choose the midpoint, $x_{\text{mid}} = x_{\text{left}} + \frac{1}{2}(x_{\text{right}} - x_{\text{left}}) = \frac{1}{2}(x_{\text{left}} + x_{\text{right}})$, as the guess for x^* .
3. If $g^{(p)}(x_{\text{mid}})$ has the same sign as $g^{(p)}(x_{\text{left}})$, then replace x_{left} by x_{mid} ; otherwise, replace x_{right} by x_{mid} . The interval for the location of the root is now reduced.
4. Repeat steps 2 and 3 until the desired level of precision is achieved.

The following program implements this algorithm for the logistic map. One possible problem is that some of the roots of $g^{(p)}(x) = 0$ also are roots of $g^{(p')}(x) = 0$ for p' equal to a factor of p . As p increases, it might become more difficult to find a root that is part of a period p trajectory and not part of a period p' trajectory.

```
// 3/6/01, 3:22 pm
package edu.clarku.sip.chapter6;
import edu.clarku.sip.plot.*;
import edu.clarku.sip.templates.*;

// find fixed point of f(x) iterated p times
public class Period implements Model
{
    private double r;
    private int p;
    private double epsilon;
    private double xleft;
    private double xright;
    private double gleft;
    private double gright;
    private Control myControl = new SControl(this);
    private double y;
    private double x;

    public void reset()
    {
        myControl.setValue("r", 0.8);           // control parameter r
        myControl.setValue("p", 2);             // period
        myControl.setValue("epsilon", 0.0000001); // desired precision
        myControl.setValue("xleft", 0.01);       // guess for xleft
        myControl.setValue("xright", 0.99);      // guess for xright
    }

    public void calculate()
    {
        r = myControl.getValue("r");
        p = (int) myControl.getValue("p");
    }
}
```

```

epsilon = myControl.getValue("epsilon");
xleft = myControl.getValue("xleft");
xright = myControl.getValue("xright");
gleft = f(xleft, r, p) - xleft;
gright = f(xright, r, p) - xright;
if (gleft*gright < 0)
{
    do
        bisection();
    while(Math.abs(xleft - xright) > epsilon);
    x = 0.5*(xleft + xright);
    myControl.println("explicit demonstration of period " + p + " behavior");
    myControl.println(0 + "\t" + x); // result
    for (int i = 1; i <= 2*p + 1; i++)
    {
        x = f(x, r, 1);
        myControl.println(i + "\t" + x);
    }
}
else
    myControl.println("range does not enclose a root");
}

void bisection()
{
    // midpoint between xleft and xright
    double xmid = 0.5*(xleft + xright);
    double gmid = f(xmid,r,p) - xmid;
    if (gmid*gleft > 0)
    {
        xleft = xmid; // change xleft
        gleft = gmid;
    }
    else
    {
        xright = xmid; // change xright
        gright = gmid;
    }
}

// f defined by recursive procedure
double f(double x, double r, double p)
{
    if (p > 1)
    {
        double y = f(x, r, p-1);

```

```

        return 4*r*y*(1-y);
    }
    else
        return 4*r*x*(1-x);
}

public static void main(String[] args)
{
    Period fixed = new Period();
    fixed.reset();
}
}

```

Problem 6.10. Unstable periodic trajectories for the logistic map

1. Test class `Period` for values of r for which the logistic map has a stable period with $p = 1$ and $p = 2$. Set the desired precision ϵ equal to 10^{-7} . Initially use $x_{\text{left}} = 0.01$ and $x_{\text{right}} = 0.99$. Calculate the stable attractor analytically and compare the results of your program with the analytical results.
2. Set $r = 0.95$ and find the periodic trajectories for $p = 1, 2, 5, 6, 7, 12, 13$, and 19.
3. Modify class `Period` so that n_b , the number of bisections needed to obtain the unstable trajectory, is listed. Choose three of the cases considered in part (c), and compute n_b for the precision ϵ equal to 0.01, 0.001, 0.0001, and 0.00001. Determine the functional dependence of n_b on ϵ .

Now that we know how to find the values of the unstable periodic trajectories, we discuss an algorithm for stabilizing this period. Suppose that we wish to stabilize the unstable trajectory of period p for a choice of $r = r_0$. The idea is to make small adjustments of $r = r_0 + \Delta r$ at each iteration so that the difference between the actual trajectory and the target periodic trajectory is small. If the actual trajectory is x_n and we wish the trajectory to be at $x(i)$, we make the next iterate x_{n+1} equal to $x(i+1)$ by expanding the difference $x_{n+1} - x(i+1)$ in a Taylor series and setting the difference to zero to first-order. We have $x_{n+1} - x(i+1) = f(x_n, r) - f(x(i), r_0)$. If we expand $f(x_n, r)$ about $(x(i), r_0)$, we have to first-order:

$$x_{n+1} - x(i+1) = \frac{\partial f(x, r)}{\partial x} [x_n - x(i)] + \frac{\partial f(x, r)}{\partial r} \Delta r = 0. \quad (6.22)$$

The partial derivatives in (6.22) are evaluated at $x = x(i)$ and $r = r_0$. The result can be expressed as

$$4r_0[1 - 2x(i)][x_n - x(i)] + 4x(i)[1 - x(i)]\Delta r = 0. \quad (6.23)$$

The solution of (6.23) for Δr can be written as

$$\Delta r = -r_0 \frac{[1 - 2x(i)][x_n - x(i)]}{x(i)[1 - x(i)]}. \quad (6.24)$$

The procedure is to iterate the logistic map at $r = r_0$ until x_n is sufficiently close to $x(i)$. The nature of chaotic systems is that the trajectory is guaranteed to come close to the desired unstable trajectory eventually. Then we use (6.24) to change the value of r so that the next iteration is closer to $x(i + 1)$. We summarize the algorithm for controlling chaos as follows:

1. Find the unstable periodic trajectory $x(i), i = 1 \text{ to } p$, for the desired value of r_0 .
2. Iterate the map with $r = r_0$ until x_n is within ϵ of $x(1)$. Then use (6.24) to determine r .
3. To turn off the control, set $r = r_0$.

Problem 6.11. Controlling chaos

1. Write a program that allows the user to turn the control on and off by hitting any key. The trajectory can be displayed either by listing the values of x_n or by plotting x_n versus n . The program should incorporate as input the desired unstable periodic trajectory $x(i)$, the period p , the value of r_0 , and the parameter ϵ .
2. Test your program with $r_0 = 0.95$ and the periods $p = 1, 5$, and 13 . Use $\epsilon = 0.02$.
3. Modify your program so that the values of r are shown as well as the values of x_n . How does r change if we vary ϵ ? Try $\epsilon = 0.05, 0.01$, and 0.005 .
4. Add a subroutine to compute n_c , the number of iterations necessary for the trajectory x_n to be within ϵ of $x(1)$ when control is turned on. Find $\langle n_c \rangle$, the average value of n_c , by starting with 100 random values of x_0 . Compute $\langle n_c \rangle$ as a function of ϵ for $\delta = 0.05, 0.005, 0.0005$, and 0.00005 . What is the functional dependence of $\langle n_c \rangle$ on ϵ ?

6.7 *Higher-Dimensional Models

So far we have discussed the logistic map as a mathematical model that has some remarkable properties and produces some interesting computer graphics. In this section we discuss some two- and three-dimensional systems that also might seem to have little to do with realistic physical systems. However, as we will see in Sections 6.8 and 6.9, similar behavior is found in realistic physical systems under the appropriate conditions.

We begin with a two-dimensional map and consider the sequence of points (x_n, y_n) generated by

$$x_{n+1} = y_n + 1 - ax_n^2 \quad (6.25a)$$

$$y_{n+1} = bx_n. \quad (6.25b)$$

The map (6.25) was proposed by Hénon who was motivated by the relevance of this dynamical system to the behavior of asteroids and satellites.

Problem 6.12. The Hénon map

1. Iterate (6.25) for $a = 1.4$ and $b = 0.3$ and plot 10^4 iterations starting from $x_0 = 0, y_0 = 0$. Make sure you compute the new value of y using the old value of x and not the new value of x . Do not plot the initial transient. Choose the `SET window` statement so that all values of the trajectory within the box drawn by the statement `BOX LINES -1.5,1.5,-0.45,0.45` are plotted. Make a similar plot beginning from the second initial condition, $x_0 = 0.63135448, y_0 = 0.18940634$. Compare the shape of the two plots. Is the shape of the two curves independent of the initial conditions?
2. Increase the scale of your plot so that all points with the box drawn by the statement `BOX LINES 0.50,0.75,0.15,0.21` are shown. Begin from the second initial condition and increase the number of computed points to 10^5 . Then make another plot showing all points within the box drawn by `BOX LINES 0.62,0.64,0.185,0.191`. If patience permits, make an additional enlargement and plot all points within the box drawn by `BOX LINES 0.6305,0.6325,0.1889,0.1895`. (You have to increase the number of computed points to order 10^6 .) What is the structure of the curves within each box? Does the attractor appear to have a similar structure on smaller and smaller length scales? Is there a region in the plane from which the points cannot escape? The region of points that do not escape is the basin of the Hénon attractor. The attractor itself is the set of points to which all points in the basin are attracted. That is, two trajectories that begin from different conditions will soon lie on the attractor. We will find in Section ?? that the Hénon attractor is an example of a strange attractor.
3. Determine if the system is chaotic, that is, sensitive to initial conditions. Start two points very close to each other and watch their trajectories for a fixed time. Choose different colors for the two trajectories.
4. It is straightforward in principle to extend the method for computing the Lyapunov exponent that we used for a one-dimensional map to higher-dimensional maps. The idea is to linearize the difference (or differential) equations and replace dx_n by the corresponding vector quantity $d\mathbf{r}_n$. This generalization yields the Lyapunov exponent corresponding to the divergence along the fastest growing direction. If a system has f degrees of freedom, it has a set of f Lyapunov exponents. A method for computing all f exponents is discussed in Project 6.22.

One of the earliest indications of chaotic behavior was in an atmospheric model developed by Lorenz. His goal was to describe the motion of a fluid layer that is heated from below. The result is convective rolls, where the warm fluid at the bottom rises, cools off at the top, and then falls down later. Lorenz simplified the description by restricting the motion to two spatial dimensions. This situation has been modeled experimentally in the laboratory and is known as a Rayleigh-Benard cell. The equations that Lorenz obtained are

$$\frac{dx}{dt} = -\sigma x + \sigma y \quad (6.26a)$$

$$\frac{dy}{dt} = -xz + rx - y \quad (6.26b)$$

$$\frac{dz}{dt} = xy - bz, \quad (6.26c)$$

where x is a measure of the fluid flow velocity circulating around the cell, y is a measure of the temperature difference between the rising and falling fluid regions, and z is a measure of the

difference in the temperature profile between the bottom and the top from the normal equilibrium temperature profile. The dimensionless parameters σ , r , and b are determined by various fluid properties, the size of the Raleigh-Benard cell, and the temperature difference in the cell. Note that the variables x , y , and z have nothing to do with the spatial coordinates, but are measures of the state of the system. Although it is not expected that you will understand the relation of the Lorenz equations to convection, we have included these equations here to reinforce the idea that simple sets of equations can exhibit chaotic behavior.

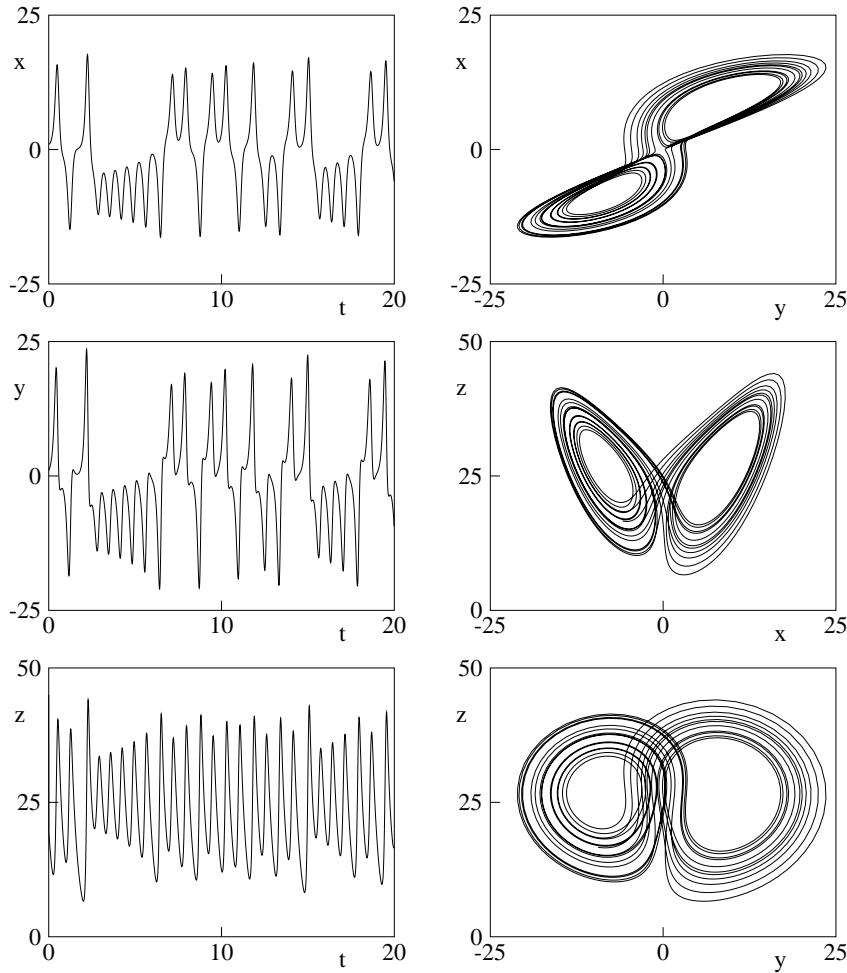


Figure 6.10: A trajectory of the Lorenz model with $\sigma = 10$, $b = 8/3$, and $r = 28$ and the initial condition $x_0 = 1$, $y_0 = 1$, $z_0 = 20$. A time interval of $t = 20$ is shown with points plotted at intervals of 0.01. The fourth-order Runge-Kutta algorithm was used with $\Delta t = 0.0025$.

Problem 6.13. The Lorenz Model

1. Use either the simple Euler algorithm or one of the Runge-Kutta methods (see Appendix 5A) to obtain a numerical solution to the Lorenz equations (6.26). Generate plots of x versus y , y versus z , and x versus z as in Figure 6.10 or use a separate graphics program to make three-dimensional plots. Explore the basin of the attractor with $\sigma = 10$, $b = 8/3$, and $r = 28$.
2. Determine qualitatively the sensitivity to initial conditions. Start two points very close to each other and watch their trajectories for approximately 10,000 time steps.
3. Let z_m denote the value of z where z is a relative maximum for the m th time. You can determine the value of z_m by finding the average of the two values of z when the right-hand side of (6.26) changes sign. Plot z_{m+1} versus z_m and describe what you find. This procedure is one way that a continuous system can be mapped onto a discrete map. What is the slope of the z_{m+1} versus z_m curve? Is its magnitude always greater than unity? If so, then this behavior is an indication of chaos. Why?

The application of the Lorenz equations to weather prediction has led to a popular metaphor known as the *butterfly effect*. That is, if the conditions are such that the atmosphere displays chaotic behavior, then even a small effect such as the flapping of a butterfly's wings would make our long-term predictions of the weather meaningless. This metaphor is made even more meaningful by inspection of Figure 6.10.

6.8 Forced Damped Pendulum

We now consider the dynamics of nonlinear mechanical systems described by classical mechanics. The general problem in classical mechanics is the determination of the positions and velocities of a system of particles subjected to certain forces. For example, we considered in Chapter 4 the celestial two-body problem and were able to predict the motion at any time. We will find that we cannot make long-time predictions for the trajectories of nonlinear classical systems when these systems exhibit chaos.

A familiar example of a nonlinear mechanical system is the simple pendulum (see Chapter 5). To make its dynamics more interesting, we assume that there is a linear damping term present and that the pivot is forced to move vertically up and down. Newton's second law for this system is (cf. McLaughlin or Percival and Richards)

$$\frac{d^2\theta}{dt^2} = -\gamma \frac{d\theta}{dt} - [\omega_0^2 + 2A \cos \omega t] \sin \theta, \quad (6.27)$$

where θ is the angle the pendulum makes with the vertical axis, γ is the damping coefficient, $\omega_0^2 = g/L$ is the natural frequency of the pendulum, and ω and A are the frequency and amplitude of the external force. Note that the effect of the vertical acceleration of the pivot is equivalent to a time-dependent gravitational field.

How do we expect the driven, damped simple pendulum to behave? Because there is damping present, we expect that if there is no external force, the pendulum would come to rest. That is, $(x = 0, v = 0)$ is a stable attractor. As A is increased from zero, this attractor remains stable for sufficiently small A . At a value of A equal to A_c , this attractor becomes unstable. How does the driven nonlinear oscillator behave as we increase the amplitude A ? Because we are mainly

interested in the (stable and unstable) fixed points of the motion, it is convenient to analyze the motion by plotting a point in phase space after every cycle of the external force. Such a phase space plot is called a *Poincaré map*. Hence, we will plot $d\theta/dt$ versus θ for values of t equal to nT for n equal to $1, 2, 3, \dots$. If the system has a period T , then the Poincaré map consists of a single point. If the period of the system is nT , there will be n points.

Class `Poincare` uses the fourth-order Runge-Kutta algorithm to compute $\theta(t)$ and the angular velocity $d\theta(t)/dt$ for the pendulum described by (6.27). A phase diagram for $d\theta(t)/dt$ versus $\theta(t)$ is shown in one frame. In the other frame the Poincaré map is represented by drawing a small box at the point $(\theta, d\theta/dt)$ at time $t = nT$. If the system has period 1, that is, if the same values of $(\theta, d\theta/dt)$ are drawn at $t = nT$, we would see only one box; otherwise we would see several boxes. Because the first few values of $(\theta, d\theta/dt)$ show the transient behavior, it is desirable to clear the display and draw a new Poincaré map without changing A , θ , or $d\theta/dt$.

```
// 03/06/01 12:43 pm
package edu.clarku.sip.ch6;
import edu.clarku.sip.plot.*;
import edu.clarku.sip.templates.*;
import edu.clarku.sip.ode_solvers.*;

// plot phase diagram and Poincare map for damped, driven pendulum
public class Poincare implements AnimationModel, RateEquation, Runnable
{
    private double gamma;
    private double A;
    private double dt;
    private double nstep;
    private Control myControl = new SAnimationControl(this);
    private Plot phaseSpace;
    private Plot poincare;
    private RK4 ode_method = new RK4(3, this);
    private double state[] = new double[3];      // (time, theta, angular velocity)
    private double pi = Math.PI;
    private Thread animationThread;

    public Poincare()
    {
        phaseSpace = new Plot("theta", "ang vel", "Phase space plot");
        poincare = new Plot("theta", "ang vel", "Poincare plot");
    }

    public void reset()
    {
        myControl.setValue("theta", 0.2);
        myControl.setValue("ang_vel", 0.6);
        myControl.setValue("gamma", 0.2);      // damping constant
        myControl.setValue("A", 0.85);        // amplitude
    }
}
```

```
}

public void run()
{
    while(animationThread == Thread.currentThread())
    {
        step();
        try
        {
            Thread.sleep(100);
        }
        catch(InterruptedException e){}
    }
}

public void step()
{
    for (int istep = 1; istep <= nstep; istep++)
    {
        ode_method.step(state);
        if (state[1] > pi)
            state[1] = state[1] - 2*pi;
        else if ( state[1] < -pi )
            state[1] = state[1] + 2*pi;

        phaseSpace.addPoint(0,state[1],state[2]);
    }
    poincare.addPoint(0,state[1],state[2]);
    phaseSpace.render();
    poincare.render();
}

public void clear()
{
    phaseSpace.deleteAllPoints();
    poincare.deleteAllPoints();
}

public void continueCalculation()
{
    animationThread = new Thread(this);
    animationThread.start();
}

public void stopCalculation()
{
```

```

        animationThread = null;
    }

    public void startCalculation()
    {
        state[1] = myControl.getValue("theta");      // initial angle
        state[2] = myControl.getValue("ang_vel");    // initial angular velocity
        gamma = myControl.getValue("gamma");         // damping constant
        A = myControl.getValue("A");                 // amplitude of external force
        state[0] = 0;                                // time
        nstep = 100;       // # iterations between Poincare plot
        // angular frequency of external force equals two and
        // hence period of external force equals pi
        double T = pi;           // period of external force
        dt = T/nstep;          // time step
        ode_method.setStepSize(dt);
        animationThread = new Thread(this);
        animationThread.start();
    }

    public void calculateRates(double state[], double rate[])
    {
        rate[0] = 1;           // rate of change of time
        rate[1] = state[2];   // state[2] = angular velocity
        rate[2] = -gamma*rate[1] - (1.0 + 2*A*Math.cos(2*state[0])) *Math.sin(state[1]);
    }

    public static void main(String[] args)
    {
        Poincare p = new Poincare();
        p.reset();
    }
}

```

Problem 6.14. Dynamics of a driven, damped simple pendulum

1. Use **Poincare** to simulate the driven, damped simple pendulum. In the program $\omega = 2$ so that the period T of the external force equals π . The program also assumes that $\omega_0 = 1$. Use $\gamma = 0.2$ and $A = 0.85$ and compute the phase space trajectory. After the initial transient, how many points do you see in the Poincaré plot? What is the period of the pendulum? Vary the initial values of θ and $d\theta/dt$. Is the attractor independent of the initial conditions? Remember to ignore the initial transient behavior.
2. Modify **Poincare** so that it plots θ and $d\theta/dt$ as a function of t . Describe the qualitative relation between the Poincaré plot, the phase space plot, and the t dependence of θ and $d\theta/dt$.

3. The amplitude A plays the role of the control parameter for the dynamics of the system. Use the behavior of the Poincaré plot to find the value $A = A_c$ at which the $(0, 0)$ attractor becomes unstable. Start with $A = 0.1$ and continue increasing A until the $(0, 0)$ attractor becomes unstable.
4. Find the period for $A = 0.1, 0.25, 0.5, 0.7, 0.75, 0.85, 0.95, 1.00, 1.02, 1.031, 1.033, 1.036$, and 1.05 . Note that for small A , the period of the oscillator is twice that of the external force. The steady state period is 2π for $A_c < A < 0.71$, π for $0.72 < A < 0.79$, and then 2π again.
5. The first period-doubling occurs for $A \approx 0.79$. Find the approximate values of A for further period-doubling and use these values of A to compute the exponent δ defined by (6.11). Compare your result for δ with the result found for the one-dimensional logistic map. Are your results consistent with those that you found for the logistic map? An analysis of this system can be found in the article by McLaughlin.
6. Sometimes a trajectory does not approach a steady state even after a very long time, but a slight perturbation causes the trajectory to move quickly onto a steady state attractor. Consider $A = 0.62$ and the initial condition $(\theta = 0.3, d\theta/dt = 0.3)$. Describe the behavior of the trajectory in phase space. During the simulation, change θ by 0.1. Does the trajectory move onto a steady state trajectory? Do similar simulations for other values of A and other initial conditions.
7. Repeat the calculations of parts (b)–(d) for $\gamma = 0.05$. What can you conclude about the effect of damping?
8. Replace the fourth-order Runge-Kutta algorithm by the lower-order Euler-Richardson algorithm. Which algorithm gives the better trade-off between accuracy and speed?

Problem 6.15. Basin of an attractor

1. For $\gamma = 0.2$ and $A > 0.79$ the pendulum rotates clockwise or counterclockwise in the steady state. Each of these two rotations is an attractor. The set of initial conditions that lead to a particular attractor is called the basin of the attractor. Modify `Poincare` so that the program draws the basin of the attractor with $d\theta/dt > 0$. For example, your program might simulate the motion for about 20 periods and then determine the sign of $d\theta/dt$. If $d\theta/dt > 0$ in the steady state, then the program plots a point in phase space at the coordinates of the initial condition. The program repeats this process for many initial conditions. Describe the basin of attraction for $A = 0.85$ and increments of the initial values of θ and $d\theta/dt$ equal to $\pi/10$.
2. Repeat part (a) using increments of the initial values of θ and $d\theta/dt$ equal to $\pi/20$ or as small as possible given your computer resources. Does the boundary of the basin of attraction appear smooth or rough? Is the basin of the attractor a single object or is it disconnected into more than one piece?
3. Repeat parts (a) and (b) for other values of A , including values near the onset of chaos and in the chaotic regime. Is there a qualitative difference between the basins of periodic and chaotic attractors? For example, can you always distinguish the boundaries of the basin?

6.9 *Hamiltonian Chaos

Hamiltonian systems are a very important class of dynamical systems. The most familiar are mechanical systems without friction, and the most important of these is the solar system. The linear harmonic oscillator and the simple pendulum that we considered in Chapter 5 are two simple examples. Many other systems can be included in the Hamiltonian framework, for example, the motion of charged particles in electric and magnetic fields, and ray optics. The Hamiltonian dynamics of charged particles is particularly relevant to confinement issues in particle accelerators, storage rings, and plasmas. In each case a function of all the coordinates and momenta called the Hamiltonian is formed. For many mechanical systems this function can be identified with the total energy. The Hamiltonian for a particle in a potential $V(x, y, z)$ is

$$H = \frac{1}{2m}(p_x^2 + p_y^2 + p_z^2) + V(x, y, z). \quad (6.28)$$

Typically we write (6.28) using the notation

$$H = \sum_i \left[\frac{p_i^2}{2m} + V(\{q_i\}) \right], \quad (6.29)$$

where $p_1 \equiv p_x$, $q_1 \equiv x$, etc. This notation emphasizes that the p_i and the q_i are generalized coordinates. For example, in some systems p can represent the angular momentum and q can represent an angle. For a system of N particles in three dimensions, the sum in (6.29) runs from 1 to $3N$, where $3N$ is the number of degrees of freedom.

The methods for constructing the generalized momenta and the Hamiltonian are described in standard classical mechanics texts. The time dependence of the generalized momenta and coordinates is given by

$$\dot{p}_i \equiv \frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i} \quad (6.30a)$$

$$\dot{q}_i \equiv \frac{dq_i}{dt} = \frac{\partial H}{\partial p_i} \quad (6.30b)$$

Check that (6.30) leads to the usual form of Newton's second law by considering the simple example of a single particle in a potential.

As we found in Chapter 5, an important property of conservative systems is preservation of areas in phase space. Consider a set of initial conditions of a dynamical system that form a closed surface in phase space. For example, if phase space is two-dimensional, this surface would be a one-dimensional loop. As time evolves, this surface in phase space will typically change its shape. For Hamiltonian systems the volume enclosed by this surface remains constant in time. For dissipative systems this volume will decrease, and hence dissipative systems are not described by a Hamiltonian. One consequence of the constant phase space volume is that Hamiltonian systems do not have phase space attractors.

In general, the motion of Hamiltonian systems is very complex, but seems to fall into three classes. In the first class the motion is regular, and there is a constant of the motion (a quantity that does not change with time) for each degree of freedom. Such a system is said to be *integrable*.

For time independent systems an obvious constant of the motion is the total energy. At the other end of the spectrum are fully chaotic systems. An important example which we will consider in Chapter 8 is a collection of molecules inside a box. Their chaotic motion is essential for the system to be described by the methods of statistical mechanics. For regular motion the change in shape of a closed surface in phase space would be rather uninteresting. For chaotic motion, nearby trajectories must exponentially diverge from each other, but be confined to a finite region of phase space. Hence, there will be local stretching of the surface accompanied by repeated folding to ensure confinement. There is another class of systems whose behavior is in between, that is, the system behaves regularly for some initial conditions, and chaotically for others. We will study these *mixed* systems in this section.

Consider the Hamiltonian for a system of N particles. If the system is integrable, there are $3N$ constants of the motion. It is natural to identify the generalized momenta with these constants. The coordinates that are associated with each of these constants will vary linearly with time. If the system is confined in phase space, then the coordinates must be periodic. If we have just one coordinate, we can think of the motion as being a point moving on a circle in phase space. In two dimensions the motion is a point moving in two circles at once, that is, a point moving on the surface of a torus. In three dimensions we can imagine a generalized torus with three circles, and so on. If the period of motion along each circle is a rational fraction of the period of all the other circles, then the torus is called a resonant torus, and the motion in phase space is periodic. If the periods are not rational fractions of each other, then the torus is called nonresonant.

If we take an integrable Hamiltonian and change it slightly, what happens to these tori? A partial answer is given by a theorem due to Kolmogorov, Arnold, and Moser (KAM), which states that, under certain circumstances, these tori will remain. When the perturbation of the Hamiltonian becomes large enough, these KAM tori are destroyed.

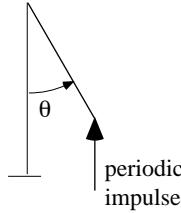


Figure 6.11: A kicked rotor consisting of a rigid rod with moment of inertia I . Gravity and friction at the pivot is ignored.

To understand the basic ideas associated with mixed systems, we consider a simple model known as the *standard map*. Consider the rotor shown in Figure 6.11. The rod has moment of inertia I and length L and is fastened at one end to a frictionless pivot. The other end is subjected to a vertical periodic impulsive force of strength k/L applied at time $t = 0, \tau, 2\tau, \dots$. Gravity is ignored. The motion of the rotor can be described by the angle θ and the corresponding angular momentum p_θ . The Hamiltonian for this system can be written as

$$H(\theta, p_\theta, t) = \frac{p_\theta^2}{2I} + k \cos \theta \sum_n \delta(t - n\tau). \quad (6.31)$$

The term $\delta(t - n\tau)$ is zero everywhere except at $t = n\tau$; its integral over time is unity if $t = n\tau$ is within the limits of integration. If we use (6.30) and (6.31), it is easy to show that the corresponding equations of motion are given by

$$\frac{dp_\theta}{dt} = k \sin \theta \sum_n \delta(t - n\tau) \quad (6.32a)$$

$$\frac{d\theta}{dt} = \frac{p_\theta}{I}. \quad (6.32b)$$

From (6.32) we see that p_θ is constant between kicks (remember that gravity is assumed to be absent), but changes discontinuously at each kick. The angle θ varies linearly with t between kicks and is continuous at each kick.

It is convenient to know the values of θ and p_θ at times just after the kick. We let θ_n and p_n be the values of $\theta(t)$ and $p_\theta(t)$ at times $t = n\tau + 0^+$, where 0^+ is a infinitesimally small positive number. If we integrate (6.32a) from $t = (n+1)\tau - 0^+$ to $t = (n+1)\tau + 0^+$, we obtain

$$p_{n+1} - p_n = k \sin \theta_{n+1}. \quad (6.33a)$$

(Remember that p is constant between kicks and the delta function contributes to the integral only when $t = (n+1)\tau$.) From (6.32b) we have

$$\theta_{n+1} - \theta_n = (\tau/I)p_n. \quad (6.33b)$$

If we choose units such that $\tau/I = 1$, we obtain the standard map

$$\theta_{n+1} = (\theta_n + p_n) \text{ modulo } 2\pi, \quad (6.34a)$$

$$p_{n+1} = p_n + k \sin \theta_{n+1}. \quad (\text{standard map}) \quad (6.34b)$$

We have added the requirement in (6.34a) that the value of the angle θ is restricted to be between zero and 2π .

Before we iterate (6.34), let us check that (6.34) represents a Hamiltonian system, that is, the area in q - p space is constant as n increases. (Here q refers to θ .) Suppose we start with a rectangle of points of length dq_n and dp_n . After one iteration, this rectangle will be deformed into a parallelogram of sides dq_{n+1} and dp_{n+1} . From (6.34) we have

$$dq_{n+1} = dq_n + dp_n \quad (6.35a)$$

$$dp_{n+1} = dp_n + k \cos q_{n+1} dq_n. \quad (6.35b)$$

If we substitute (6.35a) in (6.35b), we obtain

$$dp_{n+1} = (1 + k \cos q_{n+1}) dp_n + k \cos q_{n+1} dq_n. \quad (6.36)$$

To find the area of a parallelogram, we take the magnitude of the cross product of the vectors $d\mathbf{q}_{n+1} = (dq_n, dp_n)$ and $d\mathbf{p}_{n+1} = (1 + k \cos q_n dq_n, k \cos q_n dp_n)$. The result is $dq_n dp_n$, and hence the area in phase space has not changed. We say that the standard map is an example of an *area-preserving map*.

The qualitative properties of the standard map are explored in Problem 6.16. A summary of its properties follows. For $k = 0$, the rod rotates with a fixed angular velocity determined by

the momentum $p_n = p_0 = \text{constant}$. If p_0 is a rational number times 2π , then the trajectory in phase space consists of a sequence of isolated points lying on a horizontal line (resonant tori). Can you see why? If p_0 is not a rational number times 2π or if your computer does not have sufficient precision, then after a long enough time, the trajectory will consist of a horizontal line in phase space. As we increase k , these horizontal lines are deformed into curves that run from $q = 0$ to $q = 2\pi$, and the isolated points of the resonant tori are converted into closed loops. For some initial conditions, the trajectories will become chaotic after the map has been iterated long enough so that the transient behavior has died out.

Problem 6.16. The standard map

1. Write a program to iterate the standard map and plot its trajectory in phase space. Design the program so that more than one trajectory for the same value of the parameter k can be shown at the same time (using different colors). Choose a set of initial conditions that form a rectangle (see Problem 5.3). Does the shape of this area change with time? What happens to the total area?
2. Begin with $k = 0$ and choose an initial value of p that is a rational number times 2π . What types of trajectories do you obtain? If you obtain trajectories consisting of isolated points, do these points appear to shift due to numerical roundoff errors? How can you tell? What happens if p_0 is an irrational number times 2π ? Remember that a computer can only approximate an irrational number.
3. Consider $k = 0.2$ and explore the nature of the phase space trajectories. What structures appear that do not appear at $k = 0$? Discuss the motion of the rod corresponding to some of the typical trajectories that you find.
4. Increase k until you first find some chaotic trajectories. How can you tell that they are chaotic? Do these chaotic trajectories fill all of phase space? If there is one trajectory that is chaotic at a particular value of k , are all trajectories chaotic? What is the approximate value for k_c above which chaotic trajectories appear?

We now discuss a discrete map that models the rings of the planet Saturn (see Frøyland). The assumption is that the rings of Saturn are due to perturbations produced by Mimas, one of Saturn's moons, which is a distance of $\sigma = 185.7 \times 10^3$ km from Saturn. There are two important forces acting on objects near Saturn. The force due to Saturn can be incorporated as follows. We know that each time Mimas completes an orbit, it traverses a total angle of 2π . Hence, the angle θ of any other moon of Saturn relative to Mimas can be expressed as

$$\theta_{n+1} = \theta_n + 2\pi \frac{\sigma^{3/2}}{r_n^{3/2}}, \quad (6.37)$$

where r_n is the radius of the orbit after n revolutions. The other important force is due to Mimas and causes the radial distance r_n to change. A discrete approximation to the radial acceleration

dv_r/dt is

$$\begin{aligned}\frac{\Delta v_r}{\Delta t} &\approx \frac{v_r(t + \Delta t) - v_r(t)}{\Delta t} \\ &\approx \frac{r(t + \Delta t) - r(t)}{(\Delta t)^2} - \frac{r(t) - r(t - \Delta t)}{(\Delta t)^2} \\ &= \frac{r(t + \Delta t) - 2r(t) + r(t - \Delta t)}{(\Delta t)^2}.\end{aligned}\quad (6.38)$$

The acceleration equals the radial force due to Mimas. If we average over a complete period, then a reasonable approximation for the change in r_n due to Mimas is

$$r_{n+1} - 2r_n + r_{n-1} = f(r_n, \theta_n), \quad (6.39)$$

where $f(r_n, \theta_n)$ is the radial force. In general, the form of $f(r_n, \theta_n)$ is very complicated. We make a major simplifying assumption and take f to be proportional to $-(r_n - \sigma)^{-2}$ and to be periodic in θ_n . For simplicity, we express this periodicity in the simplest possible way, that is, as $\cos \theta_n$. We also want the map to be area conserving. These considerations lead us to the following two-dimensional map:

$$\theta_{n+1} = \theta_n + 2\pi \frac{\sigma^{3/2}}{r_n^{3/2}} \quad (6.40a)$$

$$r_{n+1} = 2r_n - r_{n-1} - a \frac{\cos \theta_n}{(r_n - \sigma)^2}. \quad (6.40b)$$

The constant a for Saturn's rings is approximately $2 \times 10^{12} \text{ km}^3$. We can show, using a similar technique as before, that the volume in (r, θ) space is preserved, and hence (6.40) is a Hamiltonian map.

The purpose of the above discussion was only to motivate and not to derive the form of the map (6.40). In Problem 6.17 we investigate how the map (6.40) yields the qualitative structure of Saturn's rings. In particular, what happens to the values of r_n if the period of a moon is related to the period of Mimas by the ratio of two integers?

Problem 6.17. A simple model of the rings of Saturn

1. Write a program to implement the map (6.40). Be sure to save the last two values of r so that the values of r_n are updated correctly. The radius of Saturn is $60.4 \times 10^3 \text{ km}$. Express all lengths in units of 10^3 km . In these units $a = 2000$. Plot the points $(r_n \cos \theta_n, r_n \sin \theta_n)$. Choose initial values for r between the radius of Saturn and σ , the distance of Mimas from Saturn, and find the bands of r_n values where stable trajectories are found.
2. What is the effect of changing the value of a ? Try $a = 200$ and $a = 20000$ and compare your results with part (a).
3. Vary the force function. Replace $\cos \theta$ by other trigonometric functions. How do your results change? If the changes are small, does that give you some confidence that the model has something to do with Saturn's rings?

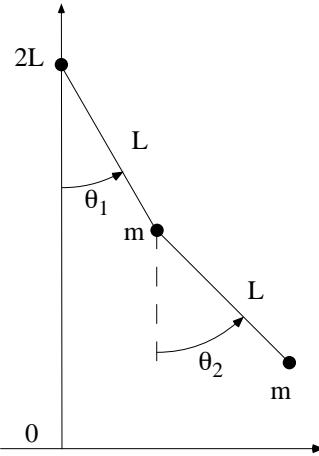


Figure 6.12: The double pendulum.

A more realistic dynamical system is the double pendulum, a system that can be demonstrated in the laboratory. This system consists of two equal point masses m , with one suspended from a fixed support by a rigid weightless rod of length L , and the other suspended from the first by a similar rod (see Figure 6.12). Because there is no friction, this system is clearly an example of a Hamiltonian system. The four rectangular coordinates x_1, y_1, x_2 , and y_2 of the two masses can be expressed in terms of two generalized coordinates θ_1, θ_2 :

$$x_1 = L \sin \theta_1 \quad (6.41a)$$

$$y_1 = 2L - L \cos \theta_1 \quad (6.41b)$$

$$x_2 = L \sin \theta_1 + L \sin \theta_2 \quad (6.41c)$$

$$y_2 = 2L - L \cos \theta_1 - L \cos \theta_2. \quad (6.41d)$$

The kinetic energy is given by

$$K = \frac{1}{2}m(\dot{x}_1^2 + \dot{x}_2^2 + \dot{y}_1^2 + \dot{y}_2^2) = \frac{1}{2}mL^2[2\dot{\theta}_1^2 + \dot{\theta}_2^2 + 2\dot{\theta}_1\dot{\theta}_2 \cos(\theta_1 - \theta_2)], \quad (6.42)$$

and the potential energy is given by

$$U = mgL(3 - 2 \cos \theta_1 - \cos \theta_2). \quad (6.43)$$

To use Hamilton's equations of motion (6.30), we need to express the sum of the kinetic energy and potential energy in terms of the generalized momenta and coordinates. In rectangular coordinates we know that the momenta are equal to $p_i = \partial K / \partial \dot{q}_i$, where for example, $q_i = x_1$ and p_i is the x -component of $m\mathbf{v}_1$. This prescription works for generalized momenta as well, and the generalized momentum corresponding to θ_1 is given by $p_1 = \partial K / \partial \dot{\theta}_1$. If we calculate the appropriate

derivatives, we can show that the generalized momenta can be written as

$$p_1 = mL^2 \left[2\dot{\theta}_1 + \dot{\theta}_2 \cos(\theta_1 - \theta_2) \right] a \quad (6.44a)$$

$$p_2 = mL^2 \left[\dot{\theta}_2 + \dot{\theta}_1 \cos(\theta_1 - \theta_2) \right]. \quad (6.44b)$$

The Hamiltonian or total energy becomes

$$\begin{aligned} H = \frac{1}{2mL^2} & \frac{p_1^2 + 2p_2^2 - 2p_1p_2 \cos(q_1 - q_2)}{1 + \sin^2(q_1 - q_2)} \\ & + mgL(3 - 2\cos q_1 - \cos q_2), \end{aligned} \quad (6.45)$$

where $q_1 = \theta_1$ and $q_2 = \theta_2$. The equations of motion can be found by using (6.45) and (6.30).

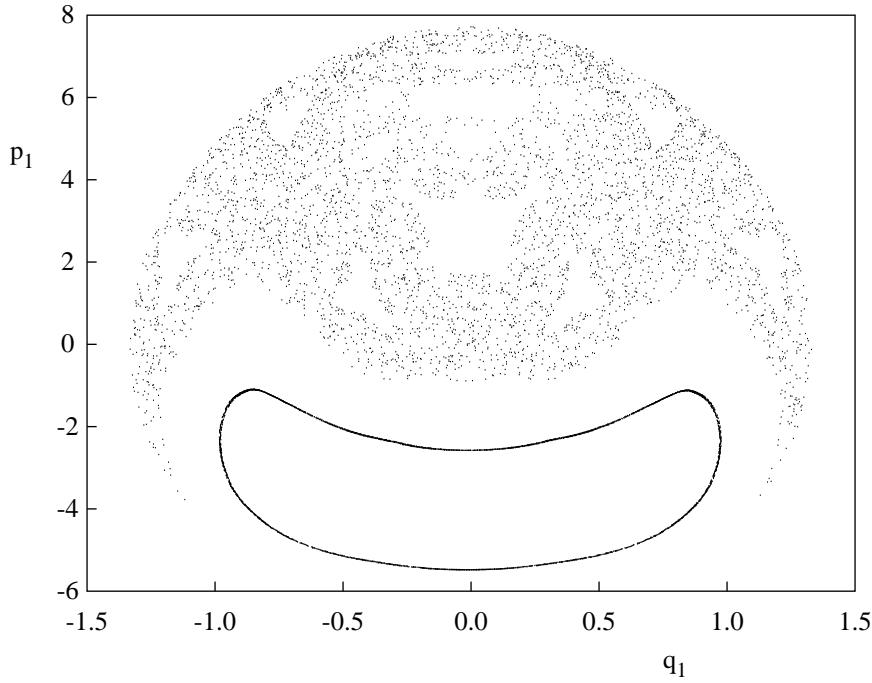


Figure 6.13: Poincaré plot for the double pendulum with p_1 plotted versus q_1 at $q_2 = 0$ and $p_2 > 0$. Two sets of initial conditions, $(q_1, q_2, p_1) = (0, 0, 0)$ and $(1.1, 0, 0)$ respectively, were used to create the plot. The initial value of the coordinate p_2 is found from (6.45) by requiring that $E = 15$.

Figure 6.13 shows a Poincaré map for the double pendulum. The coordinate p_1 is plotted versus q_1 for the same total energy $E = 15$, but for two different initial conditions. The map includes the points in the trajectory for which $q_2 = 0$ and $p_2 > 0$. Note the resemblance between Figure 6.13 and plots for the standard map above the critical value of k , that is, there is a regular trajectory and a chaotic trajectory for the same parameters, but different initial conditions.

Problem 6.18. Double pendulum

1. Use either the fourth-order Runge-Kutta algorithm (with $\Delta t = 0.003$) or the second-order Euler-Richardson algorithm (with $\Delta t = 0.001$) to simulate the double pendulum. Choose $m = 1$, $L = 1$, and $g = 9.8$ (mks units). The input parameter is the total energy E . The initial values of q_1 and q_2 can be either chosen randomly within the interval $-\pi < q_i < \pi$ or by the user. Then set the initial $p_1 = 0$, and solve for p_2 using (6.45) with $H = E$. First explore the pendulum's behavior by plotting the generalized coordinates and momenta as a function of time in four windows. Consider the energies $E = 1, 5, 10, 15$, and 40 . Try a few initial conditions for each energy. Determine visually whether the steady state behavior is regular or appears to be chaotic. Are there some values of E for which all the trajectories appear regular? Are there values of E for which all trajectories appear chaotic? Are there values of E for which both types of trajectories occur?
2. Repeat your investigations of part (a), but plot the phase space diagrams p_1 versus q_1 and p_2 versus q_2 . Are these plots more useful for determining the nature of the trajectories than those drawn in part (a)?
3. Draw the Poincaré plot with p_1 plotted versus q_1 only when $q_2 = 0$ and $p_2 > 0$. Overlay trajectories from different initial conditions, but with the same total energy on the same plot. Duplicate the plot shown in Figure 6.13. Then produce Poincaré plots for the values of E given in part (a), with at least five different initial conditions per plot. Describe the different types of behavior.
4. Is there a critical value of the total energy at which some chaotic trajectories first occur?
5. Write a subroutine to animate the double pendulum, showing the two masses moving back and forth. Describe how the motion of the pendulum is related to the behavior of the Poincaré plot.

Hamiltonian chaos has important applications in physical systems such as the solar system, the motion of the galaxies, and plasmas. It also has helped us understand the foundation for statistical mechanics. One of the most fascinating applications has been to quantum mechanics which has its roots in the Hamiltonian formulation of classical mechanics. A current area of much interest is the quantum analogue of classical Hamiltonian chaos. The meaning of this analogue is not obvious because well-defined trajectories do not exist in quantum mechanics. Moreover, Schrödinger's equation is linear and can be shown to have only periodic and quasiperiodic solutions. An introduction to problems in quantum chaos can be found in the text by Gutzwiller and in the series of articles by Srivastava, Kaufman, and Müller.

6.10 Perspective

As the many recent books and review articles on chaos can attest, it is impossible to discuss all aspects of chaos in a single chapter. We will revisit chaotic systems in Chapter ?? where we introduce the concept of fractals. We will find that one of the characteristics of chaotic dynamics is that the resulting attractors often have an intricate geometrical structure.

The most general idea that we have discussed in this chapter is that *simple systems can exhibit complex behavior*. We also have learned that computers allow us to explore the behavior of dynamical systems and visualize the numerical output. However, the simulation of a system does not automatically lead to understanding. If you are interested in learning more about the phenomena of chaos and the associated theory, the suggested readings at the end of the chapter are a good place to start. We also invite you to explore chaotic phenomenon in more detail in the following projects.

6.11 Projects

The first several projects are on various aspects of the logistic map. These projects do not exhaust the possible investigations of the properties of the logistic map.

Project 6.19. A more accurate determination of δ and α

We have seen that it is difficult to estimate δ accurately by finding the sequence of values of b_k at which the trajectory bifurcates for the k th time. A better way to estimate δ is to compute it from the sequence s_m of superstable trajectories of period 2^{m-1} . We already have found that $s_1 = 1/2$, $s_2 \approx 0.80902$, and $s_3 \approx 0.87464$. The parameters s_1, s_2, \dots can be computed directly from the equation

$$f^{(2^{m-1})}(x = \frac{1}{2}) = \frac{1}{2}. \quad (6.46)$$

For example, s_2 satisfies the relation $f^{(2)}(x = \frac{1}{2}) = \frac{1}{2}$. This relation, together with the analytical form for $f^{(2)}(x)$ given in (6.8), yields:

$$8r^2(1 - r) - 1 = 0. \quad (6.47)$$

Clearly $r = s_1 = \frac{1}{2}$ solves (6.47) with period 1. If we wish to solve (6.47) numerically for s_2 , we need to be careful not to find the irrelevant solutions corresponding to a lower period. In this case we can factor out the solution $r = \frac{1}{2}$ and solve the resultant quadratic equation analytically to find $s_2 = (1 + \sqrt{5})/4$.

1. It is straightforward to adapt the bisection method discussed in Section 6.6. Adapt class `Period` to find the numerical solutions of (6.46). Good starting values for the left-most and right-most values of r are easy to obtain. The left-most value is $r = r_\infty \approx 0.8925$. If we already know the sequence s_1, s_2, \dots, s_m , then we can estimate δ by

$$\delta_m = \frac{s_{m-1} - s_{m-2}}{s_m - s_{m-1}}. \quad (6.48)$$

We use this estimate for δ_m to estimate the right-most value of r :

$$r_{\text{right}}^{(m+1)} = \frac{s_m - s_{m-1}}{\delta_m}. \quad (6.49)$$

We choose the desired precision to be 10^{-16} . A summary of our results is given in Table 6.2. Verify these results and estimate δ .

m	period	s_m
1	1	0.500 000 000
2	2	0.809 016 994
3	4	0.874 640 425
4	8	0.888 660 970
5	16	0.891 666 899
6	32	0.892 310 883
7	64	0.892 448 823
8	128	0.892 478 091

Table 6.2: Values of the control parameter s_m for the superstable trajectories of period 2^{m-1} . Nine decimal places are shown.

2. Use your values of s_m to obtain a better estimate of α .

Project 6.20. Entropy of the logistic map

Another quantitative measure of chaos besides the Lyapunov exponent is based on the concept of entropy. Suppose that an experiment has M possible outcomes and p_1, p_2, \dots, p_M are the probabilities of each outcome. The probabilities satisfy the normalization condition $\sum_{i=1}^M p_i = 1$. If we define the entropy S as

$$S = - \sum_{i=1}^M p_i \ln p_i, \quad (6.50)$$

then S is a measure of the uncertainty of the outcome of the experiment. For example, if $p_1 = 1$, there is no uncertainty since event 1 always occurs, and (6.50) gives $S = 0$, its minimum value. Maximum uncertainty corresponds to all M events being equally probable. In this case, $p_i = 1/M$ for all i , and $S = \ln M$.

We can determine S for the logistic map by dividing the interval $[0, 1]$ into M bins or subintervals and determining the relative number of times the trajectory falls into each bin. Write a program to compute S and determine its r dependence in the range $0.7 \leq r \leq 1$. Choose a bin size of $\Delta r = 0.01$. The following fragment of code suggests how to determine p_i .

```

int prob[] = new int[100];
int nbin = 100; // number of bins
double deltax = 1.0/nbin;
double x = 0.6;
int nmax = 10000;
for (int n = 0; n <= nmax; n++)
{
    double x = 4*x*(1-x);
    int ibin = (int) (x/deltax);
    prob[ibin] = prob[ibin] + 1;
}

```

```

for (int ibin = 0; ibin <= nbin; ibin++)
    if (prob[ibin] > 0)
    {
        double temp = prob[ibin]/nmax;
        println(ibin + "\t" + temp);
    }
}

```

Plot the histogram p_i as a function of x for $r = 1$. For what value(s) of x is the histogram a maximum?

Project 6.21. From chaos to order

The bifurcation diagram of the logistic map (see Figure 6.2) has many interesting features that we do not have space to explore. For example, you might have noticed that there are several smooth dark bands in the chaotic region for $r > r_\infty$. Use `Bifurcate` to generate the bifurcation diagram for $r_\infty \leq r \leq 1$. Note that the points are not uniformly distributed in each vertical line. For example, if we start at $r = 1.0$ and decrease r , there is a band that narrows and eventually splits into two parts at $r \approx 0.9196$. If you look closely, you will see that the band splits into four parts at $r \approx 0.899$. In fact, if you look closely, you will see many more bands. What type of change occurs near the splitting (merging) of these bands? Use `IterateMap` to look at the time series of $x_{n+1} = f(x_n)$ for $r = 0.9175$. You will notice that although the trajectory looks random, it oscillates back and forth between two bands. This behavior can be seen more clearly if you look at the time series of $x_{n+1} = f^{(2)}(x_n)$. A detailed discussion of the splitting of the bands can be found in Peitgen et al.

Project 6.22. Calculation of the Lyapunov spectrum

In Section 6.5 we discussed the calculation of the Lyapunov exponent for the logistic map. If a dynamical system has a multidimensional phase space, for example, the Hénon map and the Lorenz model, there is a set of Lyapunov exponents, called the Lyapunov spectrum, that characterize the divergence of the trajectory. As an example, consider a set of initial conditions that forms a filled sphere in phase space for the (three-dimensional) Lorenz model. If we iterate the Lorenz equations, then the set of phase space points will deform into another shape. If the system has a fixed point, this shape contracts to a single point. If the system is chaotic, then, typically, the sphere will diverge in one direction, but become smaller in the other two directions. In this case we can define three Lyapunov exponents to measure the deformation in three mutually perpendicular directions. These three directions generally will not correspond to the axes of the original variables. Instead, we must use a Gram-Schmidt orthogonalization procedure.

The algorithm for finding the Lyapunov spectrum is as follows:

1. Linearize the dynamical equations. If \mathbf{r} is the f -component vector containing the dynamical variables, then define $\Delta\mathbf{r}$ as the linearized difference vector. For example, the linearized

Lorenz equations are

$$\frac{d\Delta x}{dt} = -\sigma \Delta x + \sigma \Delta y \quad (6.51a)$$

$$\frac{d\Delta y}{dt} = -x \Delta z - z \Delta x + r \Delta x - \Delta y \quad (6.51b)$$

$$\frac{d\Delta z}{dt} = x \Delta y + y \Delta x - b \Delta z. \quad (6.51c)$$

2. Define f orthonormal initial values for $\Delta \mathbf{r}$. For example, $\Delta \mathbf{r}_1(0) = (1, 0, 0)$, $\Delta \mathbf{r}_2(0) = (0, 1, 0)$, and $\Delta \mathbf{r}_3(0) = (0, 0, 1)$. Because these vectors appear in a linearized equation, they do not have to be small in magnitude.
3. Iterate the original and linearized equations of motion. One iteration yields a new vector from the original equation of motion and f new vectors $\Delta \mathbf{r}_\alpha$ from the linearized equations.
4. Find the orthonormal vectors $\Delta \mathbf{r}'_\alpha$ from the $\Delta \mathbf{r}_\alpha$ using the Gram-Schmidt procedure. That is,

$$\Delta \mathbf{r}'_1 = \frac{\Delta \mathbf{r}_1}{|\Delta \mathbf{r}_1|} \quad (6.52a)$$

$$\Delta \mathbf{r}'_2 = \frac{\Delta \mathbf{r}_2 - (\Delta \mathbf{r}'_1 \cdot \Delta \mathbf{r}_2) \Delta \mathbf{r}'_1}{|\Delta \mathbf{r}_2 - (\Delta \mathbf{r}'_1 \cdot \Delta \mathbf{r}_2) \Delta \mathbf{r}'_1|} \quad (6.52b)$$

$$\Delta \mathbf{r}'_3 = \frac{\Delta \mathbf{r}_3 - (\Delta \mathbf{r}'_1 \cdot \Delta \mathbf{r}_3) \Delta \mathbf{r}'_1 - (\Delta \mathbf{r}'_2 \cdot \Delta \mathbf{r}_3) \Delta \mathbf{r}'_2}{|\Delta \mathbf{r}_3 - (\Delta \mathbf{r}'_1 \cdot \Delta \mathbf{r}_3) \Delta \mathbf{r}'_1 - (\Delta \mathbf{r}'_2 \cdot \Delta \mathbf{r}_3) \Delta \mathbf{r}'_2|}. \quad (6.52c)$$

It is straightforward to generalize the method to higher dimensional models.

5. Set the $\Delta \mathbf{r}_\alpha(t)$ equal to the orthonormal vectors $\Delta \mathbf{r}'_\alpha(t)$.
6. Accumulate the running sum, S_α as $S_\alpha \rightarrow S_\alpha + \log |\Delta \mathbf{r}_\alpha(t)|$.
7. Repeat steps (3)–(6) and periodically output the estimate of the Lyapunov exponents $\lambda_\alpha = (1/n)S_\alpha$, where n is the number of iterations.

To obtain estimates for the Lyapunov spectrum that represent the steady state attractor, only include data after the transient behavior has died out.

1. Compute the Lyapunov spectrum for the Lorenz model with $\sigma = 16$, $b = 4$, and $r = 45.92$. Try other values of the parameters and compare your results.
2. Linearize the equations for the Hénon map and find the Lyapunov spectrum with $a = 1.4$ and $b = 0.3$ in (6.25).

Project 6.23. A spinning magnet

Consider a compass needle that is free to rotate in a periodically reversing magnetic field that is perpendicular to the axis of the needle. The equation of motion of the needle is given by

$$\frac{d^2\phi}{dt^2} = -\frac{\mu}{I} B_0 \cos \omega t \sin \phi, \quad (6.53)$$

where ϕ is the angle of the needle with respect to a fixed axis along the field, μ is the magnetic moment of the needle, I its moment of inertia, and B_0 and ω are the amplitude and the angular frequency of the magnetic field. Choose an appropriate numerical method for solving (6.53), and plot the Poincaré map at time $t = 2\pi n/\omega$. Verify that if the parameter $\lambda = \sqrt{2B_0\mu/I}/\omega > 1$, then the motion of the needle exhibits chaotic motion. Briggs (see references) discusses how to construct the corresponding laboratory system and other nonlinear physical systems.

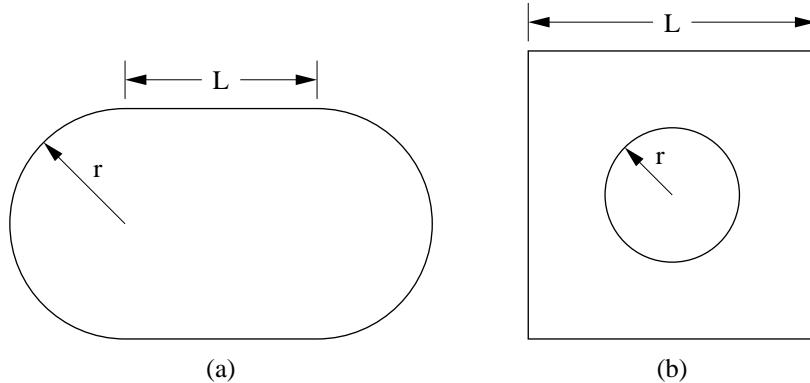


Figure 6.14: (a) Geometry of the stadium billiard model. (b) Geometry of the Sinai billiard model.

Project 6.24. Billiard models Consider a two-dimensional planar geometry in which a particle moves with constant velocity along straight line orbits until it elastically reflects off the boundary. This straight line motion occurs in various “billiard” systems. A simple example of such a system is a particle moving with fixed speed within a circle. For this geometry the angle between the particle’s momentum and the tangent to the boundary at a reflection is the same for all points.

Suppose that we divide the circle into two equal parts and connect them by straight lines of length L as shown in Figure 6.14a. This geometry is called a stadium billiard. How does the motion of a particle in the stadium compare to the motion in the circle? In both cases we can find the trajectory of the particle by geometrical considerations. The stadium billiard model and a similar geometry known as the Sinai billiard model (see Figure 6.14b) have been used as model systems for exploring the foundations of statistical mechanics. There also is much interest in relating the behavior of a classical particle in various billiard models to the solution of Schrödinger’s equation for the same geometries.

1. Write a program to simulate the stadium billiard model. Use the radius r of the semicircles as the unit of length. The algorithm for determining the path of the particle is as follows:
 - (a) Begin with an initial position (x_0, y_0) and momentum (p_{x0}, p_{y0}) of the particle such that $|\mathbf{p}_0| = 1$.
 - (b) Determine which of the four sides the particle will hit. The possibilities are the top and bottom line segments and the right and left semicircles.

- (c) Determine the next position of the particle from the intersection of the straight line defined by the current position and momentum, and the equation for the segment where the next reflection occurs.
- (d) Determine the new momentum, (p'_x, p'_y) , of the particle after reflection such that the angle of incidence equals the angle of reflection. For reflection off the line segments we have $(p'_x, p'_y) = (p_x, -p_y)$. For reflection off a circle we have

$$p'_x = (y^2 - (x - x_c)^2)p_x - 2(x - x_c)yp_y \quad (6.54a)$$

$$p'_y = -2(x - x_c)yp_x + ((x - x_c)^2 - y^2)p_y, \quad (6.54b)$$

where $(x_c, 0)$ is the center of the circle.

- (e) Repeat steps (2)–(4).
- 2. Determine if the particle dynamics is chaotic by estimating the largest Lyapunov exponent. One way to do so is to start two particles with almost identical positions and/or momenta (varying by say 10^{-5}). Compute the difference Δs of the two phase space trajectories as a function of the number of reflections n , where Δs is defined by

$$\Delta s = \sqrt{|\mathbf{r}_1 - \mathbf{r}_2|^2 + |\mathbf{p}_1 - \mathbf{p}_2|^2}. \quad (6.55)$$

Do the calculation for $L = 1$. The Lyapunov exponent can be found from a semilog plot of Δs versus n . Why does the exponential growth in Δs stop for sufficiently large n ? Repeat your calculation for different initial conditions and average your values of Δs before plotting. Repeat the calculation for $L = 0.1, 0.5$, and 2.0 and determine if your results depend on L .

- 3. Another test for the existence of chaos is the reversibility of the motion. Reverse the momentum after the particle has made n reflections, and let the drawing color equal the background color so that the path can be erased. What limitation does roundoff error place on your results? Repeat this simulation for $L = 1$ and $L = 0$.
- 4. Place a small hole of diameter d in one of the circular sections of the stadium so that the particle can escape. Choose $L = 1$ and set $d = 0.02$. Give the particle a random position and momentum, and record the time when the particle escapes through the hole. Repeat for at least 10^4 particles and compute the fraction of particles $S(n)$ remaining after a given number of reflections n . The function $S(n)$ will decay with n . Determine the functional dependence of S on n , and calculate the characteristic decay time if $S(n)$ decays exponentially. Repeat for $L = 0.1, 0.5$, and 2.0 . Is the decay time a function of L ? Does $S(n)$ decays exponentially for the circular billiard model ($L = 0$) (see Bauer and Bertsch)?
- 5. Choose an arbitrary initial position for the particle in a stadium with $L = 1$, and a small hole as in part (d). Choose at least 5000 values of the initial value p_{x0} uniformly distributed between 0 and 1. Choose p_{y0} so that $|\mathbf{p}| = 1$. Plot the escape time versus p_{x0} , and describe the visual pattern of the trajectories. Then choose 5000 values of p_{x0} in a smaller interval centered about the value of p_{x0} for which the escape time was greatest. Plot these values of the escape time versus p_{x0} . Do you see any evidence of self-similarity?
- 6. Repeat steps (a)–(e) for the Sinai billiard geometry.

Project 6.25. Mode locking and the circle map

The driven, damped pendulum can be approximated by a one-dimensional difference equation for a range of amplitudes and frequencies of the driving force. This difference equation is known as the *circle map* and is given by

$$\theta_{n+1} = \left(\theta_n + \Omega - \frac{K}{2\pi} \sin 2\pi\theta_n \right) \text{ modulo } 1 \quad (6.56)$$

The variable θ represents an angle, and Ω represents a frequency ratio, the ratio of the natural frequency of the pendulum to the frequency of the periodic driving force. The parameter K is a measure of the strength of the nonlinear coupling of the pendulum to the external force. An important quantity is the winding number which is defined as

$$W = \lim_{m \rightarrow \infty} \frac{1}{m} \sum_{n=0}^{m-1} \Delta\theta_n, \quad (6.57)$$

where $\Delta\theta_n = \Omega - (K/2\pi) \sin 2\pi\theta_n$.

1. Consider the linear case, $K = 0$. Choose $\Omega = 0.4$ and $\theta_0 = 0.2$ and determine W . Verify that if Ω is a ratio of two integers, then $W = \Omega$ and the trajectory is periodic. What is the value of W if $\Omega = \sqrt{2}/2$, an irrational number? Verify that $W = \Omega$ and that the trajectory comes arbitrarily close to any particular value of θ . Does θ_n ever return exactly to its initial value? This type of behavior of the trajectory is termed *quasiperiodic*.
2. For $K > 0$, we will find that $W \neq \Omega$ and “locks” into rational frequency ratios for a range of values of K and Ω . This type of behavior is called *mode locking*. For $K < 1$, the trajectory is either periodic or quasiperiodic. Estimate the value of W for $K = 1/2$ and values of Ω in the range $0 < \Omega \leq 1$. The widths in Ω of the various mode-locked regions where W is fixed increase with K . If time permits, consider other values of K , and draw a diagram in the K - Ω plane ($0 \leq K, \Omega \leq 1$) so that those areas corresponding to frequency locking are shaded. These shaded regions are called *Arnold tongues*.
3. For $K = 1$, all trajectories are frequency-locked periodic trajectories. Fix K at $K = 1$ and determine the dependence of W on Ω . The plot of the W versus Ω for $K = 1$ is called the *Devil’s staircase*.

Project 6.26. Nonlinear ring laser cavity Consider the ring laser in Figure 6.15 consisting of four mirrors located at the corners of a square and oriented at 45° . A nonlinear dielectric medium (for which the index of refraction depends on the intensity of the light) is placed between two of the mirrors. Light is sent in from point A and is detected at point B . Mirrors 1 and 2 are partially reflective, and mirrors 3 and 4 are totally reflective.

The electric field can be represented by a plane wave modulated by a sinusoidal function of space and time. The electric field amplitude at B arises from two sources – the light that arrives directly from A and the light that is reflected around the ring. The two sources do not have the same amplitude or phase, and the reflected field vector is rotated and diminished in amplitude

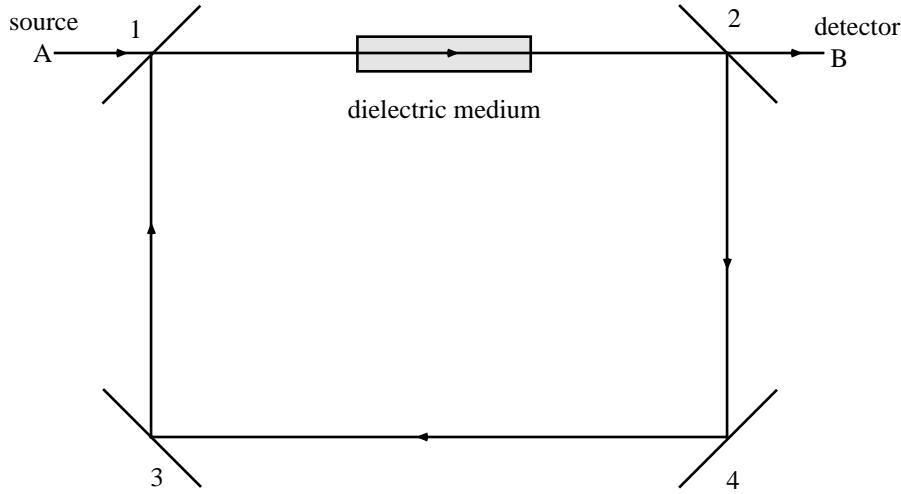


Figure 6.15: The ring laser discussed in Project 6.26.

relative to the input field. It can be shown (see Ikeda et al.) that the field at B at time t_{n+1} is related to the input field at A at time t_n by:

$$E_{n+1} = a + bE_n e^{ik - ip/(1+|E_n|^2)}. \quad (6.58)$$

We have written the electric field using complex notation, $\mathbf{E} = E_x + iE_y$, with $i = \sqrt{-1}$. The quantity a is the input amplitude of the laser at point A , b measures the loss due to the partially reflecting mirrors, and p measures the detuning due to the nonlinear dielectric medium; k would be the phase change if there were no dielectric medium present. Detuning refers to the relative rotation of the electric field due to the dielectric medium. The difference $[t_{n+1} - t_n]$ is the time for light to go once around the ring. Equation (6.58) can be written as a coupled two-dimensional map:

$$E_{x,n+1} = a + b(E_{x,n} \cos \theta_n - E_{y,n} \sin \theta_n) \quad (6.59a)$$

$$E_{y,n+1} = b(E_{x,n} \sin \theta_n + E_{y,n} \cos \theta_n), \quad (6.59b)$$

where $\theta_n = k - p/(1 + |E_n|^2)$, and $e^{i\theta} = \cos \theta + i \sin \theta$.

1. Iterate the map (6.59) for $a = 0.8$, $b = 0.9$, and $k = 0.4$. Choose the initial condition, $(E_x, E_y) = (0, 0)$. Plot E_x versus E_y for different values of p in the range $0 < p \leq 6$. Approximately determine the values of p for which the period of the system changes.
2. List the values of (E_x, E_y) and estimate as accurately as possible the value of p at which each period-doubling occurs. Use these values to estimate the Feigenbaum number δ . Compare your estimate to its value for the logistic map. How accurate is your calculation of δ ? Does the map (6.59) exhibit behavior similar to the logistic map?

3. Is there a window of period three?

Project 6.27. Chaotic scattering In Chapter 4 we discussed the classical scattering of particles off a fixed target, and found that the differential cross section for a variety of interactions is a smoothly varying function of the scattering angle. That is, a small change in the impact parameter b leads to a small change in the scattering angle θ . There is much current interest in cases where a small change in b leads to large changes in θ . Such a phenomenon is called *chaotic scattering*, because of the sensitivity to initial conditions that is characteristic of chaos. The study of chaotic scattering is relevant to the design of electronic nanostructures, because many experimental structures exhibit this type of scattering. In addition, the comparison of classical and quantum scattering is an active area of research in quantum chaos.

A typical scattering model consists of a target composed of a group of fixed hard disks and a scatterer consisting of a point particle. We trace the path of the scatterer as it bounces off the disks, and measure θ and the time of flight as a function of the impact parameter b . If a particle bounces inside the target region before leaving, the time of flight can be very long. There are even some trajectories for which the particle never leaves the target region.

Because it is difficult to monitor a trajectory that bounces back and forth between the hard disks, we instead consider a two-dimensional map that contains the key features of chaotic scattering (see Yalcinkaya and Lai for further discussion). The map is given by:

$$x_{n+1} = a[x_n - \frac{1}{4}(x_n + y_n)^2], \quad (6.60a)$$

and

$$y_{n+1} = \frac{1}{a}[y_n + \frac{1}{4}(x_n + y_n)^2], \quad (6.60b)$$

where a is a parameter. The target region is centered at the origin. In an actual scattering experiment, the relation between (x_{n+1}, y_{n+1}) and (x_n, y_n) would be much more complicated, but the map (6.60) captures most of the important features of realistic chaotic scattering experiments. The iteration number n is analogous to the number of collisions of the scattered particle off the disks. When x_n or y_n is significantly different from zero, the scatterer has left the target region.

1. Write a program to iterate the map (6.60). Let $a = 8.0$ and $y_0 = -0.3$. Choose 10^4 initial values of x_0 uniformly distributed on the interval $0 < x_0 < 0.1$. Determine the time $T(x_0)$, the number of iterations for which $x_n \leq -5.0$. After this time, x_n rapidly moves to $-\infty$. Plot $T(x_0)$ versus x_0 . Then choose 10^4 initial values in a smaller interval centered about a value of x_0 for which $T(x_0) > 7$. Plot these values of $T(x_0)$ versus x_0 . Do you see any evidence of self-similarity?
2. A trajectory is said to be *uncertain* if a small change ϵ in x_0 leads to a change in $T(x_0)$. We expect that the number of uncertain trajectories, N , will depend on a power of ϵ , that is, $N \sim \epsilon^\alpha$. Determine $N(\epsilon)$ for $\epsilon = 10^{-p}$ with $p = 2$ to 7 using the values of x_0 in part (a). Then determine the uncertainty dimension $1 - \alpha$ from a log-log plot of N versus ϵ . Repeat these measurements for other values of a . Does α depend on a ?
3. Choose 4×10^4 initial conditions in the same interval as in part (a) and determine the number of trajectories, $S(n)$, that have not yet reached $x_n = -5$ as a function of the number

of iterations n . Plot $\ln S(n)$ versus n and determine if the decay is exponential. It is possible to obtain algebraic decay for values of a less than approximately 6.5.

4. Let $a = 4.1$ and choose 100 initial conditions uniformly distributed in the region $1.0 < x_0 < 1.05$ and $0.60 < y_0 < 0.65$. Are there any trajectories that are periodic and hence have infinite escape times? Due to the accumulation of roundoff error, it is possible to find only finite, but very long escape times. These periodic trajectories form closed curves, and the regions enclosed by them are called KAM surfaces.

Project 6.28. Chemical reactions

In Project 5.8 we discussed how chemical oscillations can occur when the reactants are continuously replenished. In this project we introduce a set of chemical reactions that exhibits the period doubling route to chaos. Consider the following reactions (see Peng et al.):



Each of the above reactions has an associated rate constant. The time dependence of the concentrations of A , B , and C is given by:

$$\frac{dA}{dt} = k_1 P + k_2 PC - k_3 A - k_4 AB^2 \quad (6.62\text{a})$$

$$\frac{dB}{dt} = k_3 A + k_4 AB^2 - k_5 B \quad (6.62\text{b})$$

$$\frac{dC}{dt} = k_4 B - k_5 C. \quad (6.62\text{c})$$

We assume that P is held constant by replenishment from an external source. We also assume the chemicals are well mixed so that there is no spatial dependence. In Section 12.4 we discuss the effects of spatial inhomogeneities due to molecular diffusion. Equations (6.61) can be written in a dimensionless form as

$$\frac{dX}{d\tau} = c_1 + c_2 Z - X - XY^2 \quad (6.63\text{a})$$

$$c_3 \frac{dY}{d\tau} = X + XY^2 - Y \quad (6.63\text{b})$$

$$c_4 \frac{dZ}{d\tau} = Y - Z, \quad (6.63\text{c})$$

where the c_i are constants, $\tau = k_3 t$, and X , Y , and Z are proportional to A , B , and C , respectively.

1. Write a program to solve the coupled differential equations in (6.63). We suggest using a fourth-order Runge-Kutta algorithm with an adaptive step size. (See Project 4.20 for how to implement an adaptive step size algorithm.) Plot $\ln Y$ versus the time τ .

2. Set $c_1 = 10$, $c_3 = 0.005$, and $c_4 = 0.02$. The constant c_2 is the control parameter. Consider $c_2 = 0.10$ to 0.16 in steps of 0.005 . What is the period of $\ln Y$ for each value of c_2 ?
3. Determine the values of c_2 at which the period doublings occur for as many period doublings as you can determine. Compute the constant δ (see (6.10)) and compare its value to the value of δ for the logistic map.
4. Make a bifurcation diagram by taking the values of $\ln Y$ from the Poincaré plot at $X = Z$, and plotting them versus the control parameter c_2 . Do you see a sequence of period doublings?
5. If you have three-dimensional graphics capability, plot the trajectory of (6.63) with $\ln X$, $\ln Y$, and $\ln Z$ as the three axes. Describe the attractors for some of the cases considered in part (b).

Appendix 6A: Stability of the Fixed Points of the Logistic Map

In the following, we derive analytical expressions for the fixed points of the logistic map. The fixed-point condition is given by

$$x^* = f(x^*). \quad (6.64)$$

From (6.5) this condition yields the two fixed points

$$x^* = 0 \quad \text{and} \quad x^* = 1 - \frac{1}{4r}. \quad (6.65)$$

Because x is restricted to be positive, the only fixed point for $r < 1/4$ is $x = 0$. To determine the stability of x^* , we let

$$x_n = x^* + \epsilon_n \quad (6.66a)$$

and

$$x_{n+1} = x^* + \epsilon_{n+1}. \quad (6.66b)$$

Because $|\epsilon_n| \ll 1$, we have

$$\begin{aligned} x_{n+1} &= f(x^* + \epsilon_n) \approx f(x^*) + \epsilon_n f'(x^*) \\ &= x^* + \epsilon_n f'(x^*). \end{aligned} \quad (6.67)$$

If we compare (6.66b) and (6.67), we obtain

$$\epsilon_{n+1}/\epsilon_n = f'(x^*). \quad (6.68)$$

If $|f'(x^*)| > 1$, the trajectory will diverge from x^* since $|\epsilon_{n+1}| > |\epsilon_n|$. The opposite is true for $|f'(x^*)| < 1$. Hence, the local stability criteria for a fixed point x^* are

1. $|f'(x^*)| < 1$, x^* is stable;

2. $|f'(x^*)| = 1$, x^* is marginally stable;
3. $|f'(x^*)| > 1$, x^* is unstable.

If x^* is marginally stable, the second derivative $f''(x)$ must be considered, and the trajectory approaches x^* with deviations from x^* inversely proportional to the square root of the number of iterations.

For the logistic map the derivatives at the fixed points are respectively

$$f'(x=0) = \frac{d}{dx}[4rx(1-x)]\Big|_{x=0} = 4r \quad (6.69)$$

and

$$f'(x=x^*) = \frac{d}{dx}[4rx(1-x)]\Big|_{x=1-1/4r} = 2 - 4r. \quad (6.70)$$

It is straightforward to use (6.69) and (6.70) to find the range of r for which $x^* = 0$ and $x^* = 1 - 1/4r$ are stable.

If a trajectory has period two, then $f^{(2)}(x) = f(f(x))$ has two fixed points. If you are interested, you can solve for these fixed points analytically. As we found in Problem 6.2, these two fixed points become unstable at the same value of r . We can derive this property of the fixed points using the chain rule of differentiation:

$$\frac{d}{dx}f^{(2)}(x)\Big|_{x=x_0} = \frac{d}{dx}f(f(x))\Big|_{x=x_0} = f'(f(x_0))f'(x)\Big|_{x=x_0}. \quad (6.71)$$

If we substitute $x_1 = f(x_0)$, we can write

$$\frac{d}{dx}f(f(x))\Big|_{x=x_0} = f'(x_1)f'(x_0). \quad (6.72)$$

In the same way, we can show that

$$\frac{d}{dx}f^{(2)}(x)\Big|_{x=x_1} = f'(x_0)f'(x_1). \quad (6.73)$$

We see that if x_0 becomes unstable, then $|f^{(2)'}(x_0)| > 1$ as does $|f^{(2)'}(x_1)|$. Hence, x_1 also is unstable at the same value of r , and we conclude that both fixed points of $f^{(2)}(x)$ bifurcate at the same value of r , leading to a trajectory of period 4.

From (6.70) we see that $f'(x=x^*) = 0$ when $r = \frac{1}{2}$ and $x^* = \frac{1}{2}$. Such a fixed point is said to be *superstable*, because as we found in Problem 6.4, convergence to the fixed point is relatively rapid. In general, superstable trajectories occur whenever one of the fixed points is at $x^* = \frac{1}{2}$.

References and Suggestions for Further Reading

Books

Ralph H. Abraham and Christopher D. Shaw, *Dynamics—The Geometry of Behavior*, Addison-Wesley (1984). The authors use an abundance of visual representations.

Hao Bai-Lin, *Chaos II*, World Scientific (1990). A collection of reprints on chaotic phenomena. The following papers were cited in the text. James P. Crutchfield, J. Doyne Farmer, Norman H. Packard, and Robert S. Shaw, “Chaos,” *Sci. Amer.* **255**(6), 46–57 (1986); Mitchell J. Feigenbaum, “Quantitative universality for a class of nonlinear transformations,” *J. Stat. Phys.* **19**, 25 (1978); M. Hénon, “A two-dimensional mapping with a strange attractor,” *Commun. Math. Phys.* **50**, 50 (1976); Robert M. May, “Simple mathematical models with very complicated dynamics,” *Nature* **261**, 459 (1976); Robert Van Buskirk and Carson Jeffries, “Observation of chaotic dynamics of coupled nonlinear oscillators,” *Phys. Rev. A* **31**, 3332 (1985).

G. L. Baker and J. P. Gollub, *Chaotic Dynamics: An Introduction*, Cambridge University Press (1990). A good introduction to chaos with special emphasis on the forced damped nonlinear harmonic oscillator. Several programs are given in True BASIC.

Pedrag Cvitanovic, *Universality in Chaos*, second edition, Adam-Hilger (1989). A collection of reprints on chaotic phenomena including the articles by Hénon and May also reprinted in the Bai-Lin collection and the chaos classic, Mitchell J. Feigenbaum, “Universal behavior in nonlinear systems,” *Los Alamos Sci.* **1**, 4 (1980).

Robert Devaney, *A First Course in Chaotic Dynamical Systems*, Addison-Wesley (1992). This text is a good introduction to the more mathematical ideas behind chaos and related topics.

Jan Frøyland, *Introduction to Chaos and Coherence*, Institute of Physics Publishing (1992). See Chapter 7 for a simple model of Saturn’s rings.

Martin C. Gutzwiller, *Chaos in Classical and Quantum Mechanics*, Springer-Verlag (1990). A good introduction to problems in quantum chaos for the more advanced student.

Robert C. Hilborn, *Chaos and Nonlinear Dynamics*, Oxford University Press (1994). An excellent pedagogically oriented text.

Douglas R. Hofstadter, *Metamagical Themas*, Basic Books (1985). A shorter version is given in his article, “Metamagical themas,” *Sci. Amer.* **245**(11), 22–43 (1981).

E. Atlee Jackson, *Perspectives of Nonlinear Dynamics*, Vols. 1 and 2., Cambridge University Press (1989, 1991). An advanced text that is a joy to read.

R. V. Jensen, “Chaotic scattering, unstable periodic orbits, and fluctuations in quantum transport,” *Chaos* **1**, 101 (1991). This paper discusses the quantum version of systems similar to those discussed in Projects 6.27 and 6.24.

Francis C. Moon, *Chaotic and Fractal Dynamics, An Introduction for Applied Scientists and Engineers*, Wiley (1992). An engineering oriented text with a section on how to build devices that demonstrate chaotic dynamics.

Edward Ott, *Chaos in Dynamical Systems*, Cambridge University Press (1993). An excellent textbook on chaos at the upper undergraduate to graduate level. See also E. Ott, “Strange attractors and chaotic motions of dynamical systems,” *Rev. Mod. Phys.* **53**, 655 (1981).

Edward Ott, Tim Sauer, James A. Yorke, editors, *Coping with Chaos*, John Wiley & Sons (1994). A reprint volume emphasizing the analysis of experimental time series from chaotic systems.

Heinz-Otto Peitgen, Hartmut Jürgens, and Dietmar Saupe, *Fractals for the Classroom*, Part II, Springer-Verlag (1992). A delightful book with many beautiful illustrations. Chapter 11 discusses the nature of the bifurcation diagram of the logistic map.

Ian Percival and Derek Richards, *Introduction to Dynamics*, Cambridge University Press (1982). An advanced undergraduate text that introduces phase trajectories and the theory of stability. A derivation of the Hamiltonian for the driven damped pendulum considered in Section 6.4 is given in Chapter 5, example 5.7.

Ivars Peterson, *Newton’s Clock: Chaos in the Solar System*, W. H. Freeman (1993). An historical survey of our understanding of the motion of bodies within the solar system with a focus on chaotic motion.

Stuart L. Pimm, *The Balance of Nature*, The University of Chicago Press (1991). An introductory treatment of ecology with a chapter on applications of chaos to real biological systems. The author contends that much of the difficulty in assessing the importance of chaos is that ecological studies are too short.

Robert Shaw, *The Dripping Faucet as a Model Chaotic System*, Aerial Press, Santa Cruz, CA (1984).

Steven Strogatz, *Nonlinear Dynamics and Chaos with Applications to Physics, Biology, Chemistry and Engineering*, Addison-Wesley (1994). Another outstanding text.

Anastasios A. Tsonis, *Chaos: From Theory to Applications*, Plenum Press (1992). Of particular interest is the discussion of applications nonlinear time series forecasting.

Nicholas B. Tufillaro, Tyler Abbott, and Jeremiah Reilly, *Nonlinear Dynamics and Chaos*, Addison-Wesley (1992). See also, N. B. Tufillaro and A. M. Albano, “Chaotic dynamics of a bouncing ball,” *Amer. J. Phys.* **54**, 939 (1986). The authors describe an undergraduate level experiment of a bouncing ball subject to repeated impacts with a vibrating table.

Articles

W. Bauer and G. F. Bertsch, “Decay of ordered and chaotic systems,” *Phys. Rev. Lett.* **65**, 2213 (1990). See also the comment by Olivier Legrand and Didier Sornette, “First return, transient chaos, and decay in chaotic systems,” *Phys. Rev. Lett.* **66**, 2172 (1991), and the reply by Wolfgang Bauer and George F. Bertsch on the following page. The authors of these papers point out that the dependence of decay laws on chaotic behavior is very general and has been considered in various contexts including room acoustics and the chaotic scattering of microwaves in an “elbow” cavity. They also mention that chaotic behavior is a sufficient, but not necessary condition for exponential decay.

- Keith Briggs, “Simple experiments in chaotic dynamics,” *Amer. J. Phys.* **55**, 1083 (1987).
- J. P. Crutchfield, J. D. Farmer, and B. A. Huberman, “Fluctuations and simple chaotic dynamics,” *Phys. Repts.* **92**, 45 (1982).
- William L. Ditto and Louis M. Pecora, “Mastering chaos,” *Sci. Amer.* **262**(8), 78 (1993).
- J. C. Earnshaw and D. Haughey, “Lyapunov exponents for pedestrians,” *Amer. J. Phys.* **61**, 401 (1993).
- K. Ikeda, H. Daido, and O. Akimoto, “Optical turbulence: chaotic behavior of transmitted light from a ring cavity,” *Phys. Rev. Lett.* **45**, 709 (1980). See also S. M. Hammel, C. K. R. T. Jones, and J. V. Maloney, “Global dynamical behavior of the optical field in a ring cavity,” *J. Opt. Soc. Am. B* **2**, 552 (1985).
- Ying-Cheng Lai, “Controlling chaos,” *Computers in Physics* **8**, 62 (1994). Section 6.6 is based on this article.
- J. B. McLaughlin, “Period-doubling bifurcations and chaotic motion for a parametrically forced pendulum,” *J. Stat. Phys.* **24**, 375 (1981).
- Bo Peng, Stephen K. Scott, and Kenneth Showalter, “Period doubling and chaos in a three-variable autocatalator,” *J. Phys. Chem.* **94**, 5243 (1990).
- Niraj Srivastava, Charles Kaufman, and Gerhard Müller, “Hamiltonian chaos,” *Computers in Physics* **4**, 549 (1990); *ibid.* **5**, 239 (1991); *ibid.* **6**, 84 (1992).
- Jan Tobochnik and Harvey Gould, “Quantifying chaos,” *Computers in Physics* **3**(6), 86 (1989). Note that there is a typographical error in the equations for step (3) of the algorithm for computing the Lyapunov spectrum. The correct equations are given in Project 6.22.
- Tolga Yalcinkaya and Ying-Cheng Lai, “Chaotic scattering,” *Computers in Physics*, to be published (1995). Project 6.27 is based on a draft of this article. The map (6.60) is discussed in more detail in Yun-Tung Lau, John M. Finn, and Edward Ott, “Fractal dimension in nonhyperbolic chaotic scattering,” *Phys. Rev. Lett.* **66**, 978 (1991).

Chapter 6

Oscillations and the Simulation Interface

©2001 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
21 February 2002

We introduce threads in the context of exploring the behavior of oscillatory systems and introduce the Simulation interface.

6.1 Simple Harmonic Motion

There are many physical systems that undergo regular, repeating motion. Motion that repeats itself at definite intervals, for example, the motion of the earth about the sun, is said to be *periodic*. If an object undergoes periodic motion between two limits over the same path, we call the motion *oscillatory*. Examples of oscillatory motion that are familiar to us from our everyday experience include a plucked guitar string and the pendulum in a grandfather clock. Less obvious examples are microscopic phenomena such as the oscillations of the atoms in crystalline solids.

To illustrate the important concepts associated with oscillatory phenomena, consider an object of mass m connected to the free end of a spring. The object slides on a frictionless, horizontal surface (see Figure 6.1). We specify the position of the object by x and take $x = 0$ to be the *equilibrium* position of the object, that is, the position when the spring is relaxed. If the object is moved from $x = 0$ and then released, the object oscillates along a horizontal line. If the spring is not compressed or stretched too far from $x = 0$, the force on the object at position x is linearly related to x :

$$F = -kx. \quad (6.1)$$

The *force constant* k is a measure of the stiffness of the spring. The negative sign in (6.1) implies that the force acts to restore the object to its equilibrium position. Newton's equation of motion

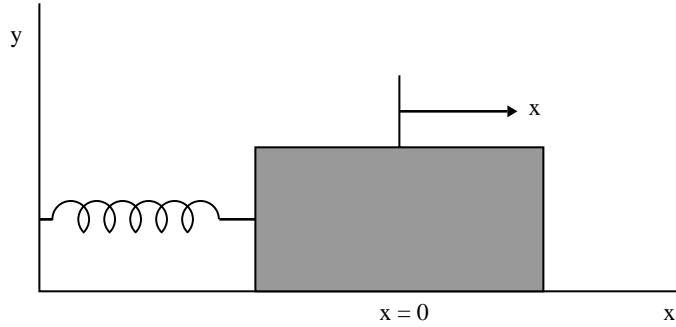


Figure 6.1: A one-dimensional harmonic oscillator. The block slides horizontally on the frictionless surface.

of the object can be written as

$$\frac{d^2x}{dt^2} = -\omega_0^2 x, \quad (6.2)$$

where the angular frequency ω_0 is defined by

$$\omega_0^2 = \frac{k}{m}. \quad (6.3)$$

The dynamical behavior described by (6.2) is called *simple harmonic motion* and can be solved analytically in terms of sine and cosine functions. Because the form of the solution will help us introduce some of the terminology needed to discuss oscillatory motion, we include the solution here. One form of the solution is

$$x(t) = A \cos(\omega_0 t + \delta), \quad (6.4)$$

where A and δ are constants and the argument of the cosine is in radians. It is straightforward to check by substitution that (6.4) is a solution of (6.2). The constants A and δ are called the *amplitude* and the *phase* respectively, and can be determined by the initial conditions for x and the velocity $v = dx/dt$.

Because the cosine is a periodic function with period 2π , we know that $x(t)$ in (6.4) also is periodic. We define the *period* T as the smallest time for which the motion repeats itself, that is,

$$x(t + T) = x(t). \quad (6.5)$$

Because $\omega_0 T$ corresponds to one *cycle*, we have

$$T = \frac{2\pi}{\omega_0} = \frac{2\pi}{\sqrt{k/m}}. \quad (6.6)$$

The *frequency* ν of the motion is the number of cycles per second and is given by $\nu = 1/T$. Note that T depends on the ratio k/m and not on A and δ . Hence the period of simple harmonic motion is independent of the amplitude of the motion.

Although the position and velocity of the oscillator are continuously changing, the total energy E remains constant and is given by

$$E = \frac{1}{2}mv^2 + \frac{1}{2}kx^2 = \frac{1}{2}kA^2. \quad (6.7)$$

The two terms in (6.7) are the kinetic and potential energies, respectively.

6.2 A Simple Program

We will find it convenient to write Newton's equation of motion as a rate equation with the state variables being the position, velocity, and time. Then Newton's equation of motion can be rewritten as a system of *three* coupled first-order equations:

$$\dot{x} = v \quad (6.8a)$$

$$\dot{v} = \omega_0^2 x \quad (6.8b)$$

$$\dot{t} = 1. \quad (6.8c)$$

Note that the time is being treated in the same way as the other variables. For reasons that will become clearer in Chapter 7, we will represent these elements in the array `state`.

We will follow in the same spirit as the programs in Chapter 5 and first introduce the class `SHO` to represent the mass on the spring:

Listing 6.1: Implementation of the Euler algorithm for the simple harmonic oscillator.

```
package org.opensourcephysics.sip.ch6;

public class SHO {
    double omega2 = 1.0;           // omega2 = k/m
    double[] state = new double[3]; // current state , [ position , velocity , time]

    public void move(double dt) {
        double x = state[0];
        state[0] = state[0] + state[1] * dt;    // position
        state[1] = state[1] - omega2 * x * dt;   // velocity
        state[2] = state[2] + dt;                // time
    }
}
```

The main difference is that we have represented the position, velocity, and time by a one-dimensional array.

The target application, `SHOApp`, implements the `Calculation` interface in the usual way. Note the similarities with Listing 5.4.

Listing 6.2: Implementation of the Calculation interface for the simple harmonic oscillator.

```
package org.opensourcephysics.sip.ch6;
import java.awt.Color;
```

```

import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;

public class SHOApp implements Calculation {
    Control myControl;
    DrawingFrame plottingFrame;
    PlottingPanel plottingPanel;
    DatasetCollection odeData;
    SHO sho;
    double dt;
    int numberofSteps;
    int datasetIndex = -1;

    public SHOApp() {
        plottingPanel = new PlottingPanel("Time", "position", "Position vs. Time");
        plottingFrame = new DrawingFrame(plottingPanel);
        odeData = new DatasetCollection();
        plottingPanel.addDrawable(odeData);
        sho = new SHO();
    }

    public void setControl(Control control) {
        myControl = control;
        resetCalculation ();
    }

    public void calculate() {
        sho.state [0] = myControl.getDouble("x");
        sho.state [1] = myControl.getDouble("vx");
        numberofSteps = myControl.getInt("number of steps");
        dt = myControl.getDouble("dt");
        datasetIndex++;
        for(int i = 0; i < numberofSteps; i++) {
            odeData.append(datasetIndex, sho.state[2], sho.state [0]);
            sho.move(dt);
        }
        plottingPanel.repaint ();
    }

    public void resetCalculation() {
        odeData.clear();
        plottingPanel.repaint ();
        datasetIndex = -1;
        sho.state [0] = 1.0;           // x
        sho.state [1] = 0;             // vx
        sho.state [2] = 0;             // time
        dt = 0.05;
        myControl.setValue("x", sho.state [0]);
        myControl.setValue("vx", sho.state [1]);
        myControl.setValue("dt", dt);
    }
}

```

```

        myControl.setValue("number of steps", 100);
    }

public static void main(String[] args) {
    Calculation app = new SHOApp();
    Control c = new CalculationControl(app);
    app.setControl(c);
}
}

```

In Exercise 6.1 we test various algorithms for finding the dynamical motion for the simple harmonic oscillator.

Problem 6.1. Verification of SHO

- Test `SHOApp` for various values of the time step `dt`. Is there anything wrong with the numerical solution?
- Define a new class, `SHOexact`, that calculates the analytical solution (6.4) for the simple harmonic oscillator. Instantiate this class in your program and add a plot of the analytical solution to the plotting panel. Describe the qualitative differences between the analytical and numerical solutions. Can the Euler algorithm be rescued by going to smaller step sizes?
- Change the `move` method in `SHO` as follows:

```

public void calculate() {
    state[1] = state[1] - omega2*state[0]*dt; // updated velocity
    state[0] = state[0] + state[1]*dt;         // updated position
    state[2] = state[2] + dt;                  // time
}

```

Note that `move` no longer implements Euler's algorithm because the velocity is updated first and this new velocity is used to update the position. In contrast, the Euler algorithm requires that all kinematical quantities be updated using the values at the beginning of the interval. As discussed in Chapter 5, the above listing is an implementation of the *Euler-Cromer* algorithm. Does this algorithm have any advantage over the Euler algorithm for this dynamical system?

- Modify the program so that ω_0 is an input variable. Determine the period T for different values of ω_0 , and confirm that $T = 2\pi/\omega$.

The results of Problem 6.1 should help us be aware that numerical methods have a *global truncation error*. This global error accumulates not only because each iteration has its own error, but because errors introduced in one iteration can grow in succeeding iteration. In fact, errors in the Euler method often grow exponentially and so merely choosing a smaller time step might not lead to a better solution. In Problem 6.2 we use conservation of energy to determine which algorithm does the best job of solving Newton's equations of motion of the simple harmonic oscillator using a reasonable choice of Δt .

Problem 6.2. Energy conservation

- a. Modify your program so that E_n , the total energy per unit mass, is computed at time $t_n = t_0 + n\Delta t$. Use Euler's algorithm and plot the difference $\Delta E_n = E_n - E_0$ as a function of t_n for several cycles for a given value of Δt . (E_0 is the initial total energy.) Choose $x(t=0) = 1$, $v(t=0) = 0$ and $\omega_0^2 = k/m = 9$ and start with $\Delta t = 0.05$. Is the difference ΔE_n uniformly small throughout the cycle? Does ΔE_n drift, that is, become bigger with time? What is the optimum choice of Δt ?
- b. Use the Euler-Cromer algorithm to answer the same questions as in part (a).
- c. Modify the program so that the Euler-Richardson algorithm is used and answer the same questions as in part (a).
- d. Compute ΔE_n and describe the qualitative difference between the time dependence of ΔE_n using the three algorithms. Which method is most consistent with the requirement of conservation of energy? For fixed Δt , which algorithm yields better results for the position in comparison to the analytical solution (6.4)? Is the requirement of conservation of energy consistent with the relative accuracy of the computed positions? For which algorithm does the total energy drift the least?
- e. Choose the best algorithm based on the criteria of your choice and determine the values of Δt that are needed to conserve the total energy to within 0.1% over one cycle for $\omega_0 = 3$ and for $\omega_0 = 6$. Can you use the same value of Δt for both values of ω_0 ? If not, how do the values of Δt correspond to the relative values of the period in the two cases?
- f. It might occur to you that adding a correction for the acceleration to the Euler method would produce more accurate results. The Verlet algorithm is based on this idea. One form of this algorithm is given by:

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2, \quad (6.9a)$$

and

$$v_{n+1} = v_n + \frac{1}{2} (a_{n+1} + a_n) \Delta t. \quad (6.9b)$$

Modify the program so that the Verlet algorithm is used.

You might have noticed that modifying your program each time so that a different algorithm can be tested is not very object oriented. In Chapter 7 we will introduce the ODE interface to allow changes in the algorithm to be made easily.

6.3 Threads

In Chapter 5 and Section 6.2 we simulated the motion of a particle by solving Newton's equations of motion using several simple numerical methods. The solutions that we obtained were plotted, but we did not attempt to visualize the motion as it occurred, and all our plots were generated *after* the calculations were completed. You probably also noticed that our programs ran for a number of time steps that was determined by a while statement or predetermined by the user.

Hence, we could not intervene to stop the simulation or change a parameter. In the following we will find that in order to visualize the motion as it occurs or to intervene during the simulation to change a parameter, it is necessary to use *threads*.

The `Calculation` interface has an important limitation — it is unable to respond to the click of a button while the calculation is being performed. This lack of response is not a problem if the calculation is short. However, if the calculation takes a long time, the user might assume that the program has crashed or is an infinite loop.

As a simple example of a program that has a similar structure, Listing 6.3 has a Stop button and an associated `ActionListener` whose sole function is to terminate the loop when the button is clicked. The boolean variable, `running`, becomes `true` when the Go button is clicked and should become `false` when the Stop button is clicked. Does this program work as advertised?

Listing 6.3: Example of a program that needs a thread.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

// program will run forever – need a thread to start and stop
public class NeedThreadApp extends JFrame implements ActionListener {
    JButton button;
    boolean running = false;

    public NeedThreadApp() {
        button = new JButton("Go");
        button.addActionListener(this);
        JTextArea outputArea = new JTextArea(20,20);
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        c.add(button);
        setSize(200,200);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("Go")) {
            button.setText("Stop");
            int i = 0;
            running = true;
            while (running) {
                System.out.println("i = " + i);
                i++;
            }
        }
        if (e.getActionCommand().equals("Stop")) {
            button.setText("Go");
            running = false;
        }
    }
}
```

```

    }

    public static void main(String[] args) {
        new NeedThreadApp();
    }
}

```

Exercise 6.3. Event queue

Compile the program in Listing 6.3 and run it. What happens if you click on the Go button? How can you stop the program? Your answer will be operating system dependent.

The problem with the program in Listing 6.3 is that once the program enters the while loop, it never leaves the loop to check on the value of `running`. There is no way the program can “listen” for the `stop` button click. What is the value of `running` after the Go button is clicked? In this case the program is trying to do two tasks at once, and there is no way for the program to break out of the while loop. It clearly is poor programming practice not to be able to interrupt a program.

The elegant way of overcoming this problem is to use *threads*. A thread is a single sequential flow of control within a program. Every program has at least one (default) thread. This thread begins at the first statement of `main()`. Traditional procedural programs have one thread. That is, the program starts by executing a line of code, then another line, and then another. The program may jump from one block of code to another, but the program always moves from one statement to the next and never enters a state where statements are executed independently. Java does not have to be like that.

If a computer has only one processor, simultaneous execution of multiple applications is an illusion created by the computer’s operating system. Although the operating system of single processor computer may be able to run multiple applications, it does so by allocating a fraction of a second to one application and then to the another so that it appears that these applications are running simultaneously. In this way, we can scroll pages in our editor and print at apparently the same time. One of the advantages of Java is that it makes simultaneous execution relatively easy.

Multiple threads are used to isolate tasks and allow the program to execute these tasks at the same time. Having multiple threads is similar to having multiple applications insofar as the operating system causes one thread to run and then another. But threads are different from multiple applications; threads are *independent* calculations (not simultaneous calculations) that access the same data. Java may run one thread to produce a graph and another thread to calculate the solution of a differential equation. In fact, the graph may be plotting the numerical solution to the equation. Because threads can share the same data, we have to make sure that these calculations work together in such a way that neither calculation changes the data that is being used by the other. One of the advantages of Java is that multiple threads are automatically distributed over multiple processors.

A thread can execute statements within an object that implements the `Runnable` interface. This interface consists of a single method, the `run` method, and the thread executes the code within this method. The `run` method cannot be invoked directly. When the `run` method returns, the calculation thread stops executing and is said to die. Another thread must be created if we wish to invoke the `run` method a second time. Listing 6.4 shows how threads can be used to start and stop a calculation.

Listing 6.4: Simple example of the use of threads.

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

// uses thread to start and stop
public class FirstThreadApp extends JFrame implements ActionListener, Runnable
{
    JButton button;
    boolean running = false;
    Thread thread;

    public FirstThreadApp() {
        button = new JButton("Go");
        button.addActionListener(this);
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        c.add(button);
        setSize(200,200);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("Go")) {
            button.setText("Stop");
            running = true;
            thread = new Thread(this);
            thread.start();
        }
        if (e.getActionCommand().equals("Stop")) {
            button.setText("Go");
            running = false;
        }
    }

    public void run() {
        int i = 0;
        while (running) {
            System.out.println("i = " + i);
            i++;
        }
    }

    public static void main(String[] args) {
        new FirstThreadApp();
    }
}

```

Note that the program is a `Runnable` object. The thread is created inside the `ActionPerformed`

method by invoking the thread's constructor, `thread = new Thread(this)`, and passing a reference to the `Runnable` program. The thread will start running after the `main` method is executed.

Writing a proper `run` method is not difficult, but requires attention to detail. The most important consideration is how to end the loop, thereby stopping the calculation and killing the thread. The correct way to stop the run method is to set the calculation thread to null. It is important that a thread not monopolize the computer's processor(s) without giving other threads a chance to run. We can do so by invoking the `sleep(int delay)` method from within the run method. The argument that is passed to sleep is the number of milliseconds that the thread should wait before continuing the calculation. A good value for computational speed is 10 milliseconds. Because the eye cannot respond to rapid screen redraws, a good value for smooth animation is 100 milliseconds or 1/10 of a second. Note that Java requires that the sleep method be enclosed in a `try` block to properly handle interrupt exceptions.

If possible, we should avoid having two threads accessing the same data. If a thread changes an object's data, we must insure that this change does not effect another concurrent use of the same data. Our programs take the easy way out by creating a new thread only if the previous calculation thread has died.

6.4 Animation Interface

In order to start and stop threads and hence be able to do animations, we have designed a more flexible graphical user interface, the `Animation` interface, to complement the `Calculation` interface that we introduced in Chapter 4. The `AnimationControl` class implements the `Control` interface and is similar to the `Control` interface that we have been using. The `AnimationControl` class has six buttons that invoke five methods. These methods are defined in another class that implements the `Animation` interface.

Listing 6.5: The `Animation` interface.

```
package org.opensourcephysics.controls;

public interface Animation extends Runnable {

    public void setControl(Control control);
    public void startAnimation();
    public void stopAnimation();
    public void initializeAnimation();
    public void resetAnimation();
    public void stepAnimation();
}
```

The methods in the `Animation` interface initialize, start, stop, step, and reset a calculation.

Exercise 6.4. Animation control test

The Chapter 6 directory has a test program, `AnimationTestApp`. Copy this program into a working directory, edit the package declaration and the import statements, and then compile and run this application. Describe the sequence of actions that the user must perform to run the calculation. What happens to the parameter input area in the control when the calculation is running?

The most important function of the `AnimationControl` class is to handle input/output by invoking the `initializeAnimation` and `resetAnimation` methods. Note that starting an animation is a two-step process. When an animation is created, it is in input mode. Input mode is used to edit parameter values, whereas run mode is used to run and single step a calculation. Consequently, the parameters to be changed in run mode. It is possible to return to input mode at any time by clicking the `edit` button. This paradigm may seem constraining at first, but it insures that input parameters are clearly stated and that these parameter are not changed arbitrarily while the program is running.

To gain familiarity with the animation user interface, we recast the simple harmonic oscillator program into this framework

Listing 6.6: Simple harmonic oscillator using the Animation interface.

```
package org.opensourcephysics.sip.ch6;
import java.awt.Color;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;

public class SHOThreadApp implements Animation {
    Control myControl;
    DrawingFrame plottingFrame;
    PlottingPanel plottingPanel;
    DatasetCollection odeData;
    SHO sho;
    double dt;
    Thread animationThread;

    public SHOThreadApp() {
        plottingPanel = new PlottingPanel("time", "position", "position vs. time");
        plottingFrame = new DrawingFrame(plottingPanel);
        odeData = new DatasetCollection();
        plottingPanel.addDrawable(odeData);
        sho = new SHO();
    }

    public void setControl(Control control) {
        myControl = control;
        resetAnimation();
    }

    public void initializeAnimation() {
        sho.state[0] = myControl.getDouble("x");
        sho.state[1] = myControl.getDouble("vx");
        dt = myControl.getDouble("dt");
    }

    public void startAnimation() {
        animationThread = new Thread(this);
        animationThread.start();
    }
}
```

```

public void stopAnimation() {
    animationThread = null;
    updateControlValues();
}

public void run() {
    while (animationThread == Thread.currentThread()) {
        odeData.append(0, sho.state[2], sho.state[0]); // first dataset
        sho.move(dt);
        plottingPanel.paintImmediately(plottingPanel.getBounds()); // paint entire panel
        try {
            Thread.sleep(10);
        } catch(InterruptedException e) {}
    }
}

public void stepAnimation() {
    odeData.append(0, sho.state[2], sho.state[0]);
    sho.move(dt);
    plottingPanel.repaint();
    updateControlValues();
}

public void resetAnimation() {
    odeData.clear();
    plottingPanel.repaint();
    sho.state[0] = 1.0;
    sho.state[1] = 0;
    sho.state[2] = 0;
    dt = 0.2;
    myControl.setValue("dt", dt);
    updateControlValues();
}

public void updateControlValues() {
    myControl.setValue("x", sho.state[0]);
    myControl.setValue("vx", sho.state[1]);
}

public static void main(String[] args) {
    Animation app = new SHOThreadApp();
    Control myControl = new AnimationControl(app);
    app.setControl(myControl);
}
}

```

Running the program and clicking the `init` button changes the behavior of the control. The first button now changes its label from `Init` to `start`. Clicking the `start` button creates the animation thread. The heart of the animation is the `run` method because it performs the physics

and transfers the resulting data to the display objects. The `sho` object encapsulates the oscillator's state variables and a solution method. The `sho.step(dt)` statement advances the state by Δt .

You may want to examine the current values as you run an animation. One way to do this is to update the control when the calculation is stopped. These values can then be edited. Another way is to write current values using the `System.out.println` or `myControl.println` methods. The `resetAnimation` method should either stop a calculation if it is currently in run mode or reset the simulation to a known initial state if it is not. The `stepAnimation` method is designed to allow us to single step an animation. Because we often want to examine how the state changes in a single step, we write the current values or position, velocity, and time to the control as we did in the `stopAnimation` method. We now have all the pieces needed to construct fairly sophisticated simulation program.

Problem 6.5. Analysis of simple harmonic motion

1. Modify the simple harmonic oscillator program so that the position and velocity of the oscillator are plotted as a function of the time t . Choose one of the four numerical algorithms and the optimum value of Δt that you determined in Problem 6.2. Describe the qualitative behavior of the position and velocity.
2. Plot the time dependence of the potential energy and the kinetic energy through one complete cycle. Where in the cycle is the kinetic energy a maximum?
3. Compute the average value of the kinetic energy and the potential energy during a complete cycle. Is there a relation between the two averages?
4. Compute $x(t)$ for different values of A and show that the shape of $x(t)$ is independent of A , that is, show that $x(t)/A$ is a *universal* function of t for a fixed value of k/m . In what units should the time be measured so that the ratio $x(t)/A$ is independent of A and k/m ?
5. The state of motion of the one-dimensional oscillator is completely specified as a function of time by $x(t)$ and $p(t)$, where p is the momentum of the oscillator. These quantities may be interpreted as the coordinates of a point in a two-dimensional space known as *phase space*. As time increases, the point $(x(t), p(t))$ moves along a *trajectory in phase space*. Modify your program so that the momentum p is plotted as a function of x , that is, choose p and x as the vertical and horizontal axes respectively. Set $\omega_0 = 3$ and compute the phase space trajectories for the initial condition $x(t = 0) = 1, v(t = 0) = 0$. What is the shape of the trajectory in phase space? What is the shape for the initial conditions, $x(t = 0) = 0, v(t = 0) = 1$ and $x(t = 0) = 4, v(t = 0) = 0$? Do you find a different phase trajectory for each initial condition? What physical quantity distinguishes the phase trajectories? Is the motion of a representative point (x, p) always in the clockwise or counterclockwise direction?

Threads can be used in any situation where the number of repetitions is not predetermined and the user is expected to interact with a visualization, such as in Problems 6.6 and 6.7.

Problem 6.6. Lissajous figures

A computer display can be used to simulate the output seen on an oscilloscope. Imagine that the vertical and horizontal inputs to an oscilloscope are sinusoidal in time, that is, $x = A \sin(\omega_x t + \phi_x)$ and $y = B \sin(\omega_y t + \phi_y)$. If the curve that is drawn repeats itself, such a curve is called a *Lissajous*

figure. For what values of the angular frequencies ω_x and ω_y do you obtain a Lissajous figure? How do the phase factors ϕ_x and ϕ_y and the amplitudes A and B affect the curves? First choose $A = B = 1$, $\omega_x = 2$, $\omega_y = 3$, $\phi_x = \pi/6$, and $\phi_y = \pi/4$. Write a program to plot y versus x , as t advances from $t = 0$.

Traveling waves are ubiquitous in nature and give rise to important phenomena such as beats and standing waves. We investigate their behavior in Problems 6.7.

Problem 6.7. Superposition of waves

- Write a program to plot $A \sin(kx + \omega t)$ from $x = x_{\min}$ to $x = x_{\max}$ for various values of t as t advances from $t = 0$ to $2\pi/\omega$. For simplicity, take $A = 1$, $\omega = 2\pi$, and $\lambda = 2\pi/k = 2$.
- Use your program to demonstrate a standing wave with wavelength $\lambda = 2$ and frequency of unity. What function should you plot?
- Use your program to demonstrate beats by plotting $(y_1 + y_2)^2$ as functions of time in the range $x_{\min} = -10$ and $x_{\max} = 10$. The quantity $y_1 + y_2$ corresponds to the superpositions of two waves. What is the beat frequency for each of the superpositions?

$$y_1(x, t) = \sin[8.4(x - 1.1t)] \quad (6.10a)$$

$$y_2(x, t) = \sin[8.0(x - 1.1t)] \quad (6.10b)$$

and

$$y_1(x, t) = \sin[8.4(x - 1.2t)] \quad (6.11a)$$

$$y_2(x, t) = \sin[8.0(x - 1.0t)] \quad (6.11b)$$

and

$$y_1(x, t) = \sin[8.4(x - 1.0t)] \quad (6.12a)$$

$$y_2(x, t) = \sin[8.0(x - 1.2t)]. \quad (6.12b)$$

What difference do you observe between these superpositions?

6.5 Visualization of the Motion of a Pendulum

A common example of a mechanical system that exhibits oscillatory motion is the simple pendulum (see Figure 6.2). A simple pendulum is an idealized system consisting of a particle or bob of mass m attached to the lower end of a rigid rod of length L and negligible mass; the upper end of the rod pivots without friction. If the bob is pulled to one side from its equilibrium position and released, the pendulum swings in a vertical plane.

Because the bob is constrained to move along the arc of a circle of radius L about the center O , the bob's position is specified by the arc length or by the angle θ (see Figure 6.2). The linear velocity and acceleration of the bob as measured along the arc are given by

$$v = L \frac{d\theta}{dt} \quad (6.13)$$

$$a = L \frac{d^2\theta}{dt^2}. \quad (6.14)$$

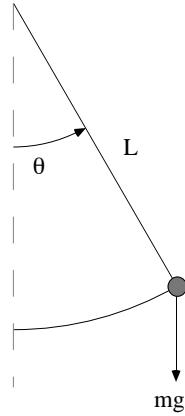


Figure 6.2: Force diagram for a simple pendulum. The angle θ is measured from the vertical direction and is positive if the mass is to the right of the vertical and negative if it is to the left.

In the absence of friction, two forces act on the object: the force mg vertically downward and the force of the rod which is directed inward to the center if $|\theta| < \pi/2$. Note that the effect of the rigid rod is to constrain the motion of the bob along the arc. From Figure 6.2, we can see that the component of mg along the arc is $mg \sin \theta$ in the direction of decreasing θ . Hence, the equation of motion can be written as

$$mL \frac{d^2\theta}{dt^2} = -mg \sin \theta \quad (6.15)$$

or

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L} \sin \theta. \quad (6.16)$$

Equation (6.16) is an example of a nonlinear equation because $\sin \theta$ rather than θ appears. Most nonlinear equations do not have analytical solutions in terms of well-known functions, and (6.16) is no exception. However, if the pendulum undergoes oscillations of sufficiently small amplitude, then $\sin \theta \approx \theta$, and (6.16) reduces to

$$\frac{d^2\theta}{dt^2} \approx -\frac{g}{L} \theta. \quad (\theta \ll 1) \quad (6.17)$$

Remember that θ is measured in radians.

Part of the fun of studying physics comes from realizing that equations that appear in different areas (and different fields) are often identical. An example of this “crossover” effect can be seen from a comparison of (6.2) and (6.17). If we associate x with θ , we see that the two equations are identical in form, and we can immediately conclude that for $\theta \ll 1$, the period of a pendulum is given by

$$T = 2\pi\sqrt{L/g}. \quad (6.18)$$

One way to understand the motion of a pendulum with large oscillations is to solve (6.16) numerically. Because we know that a numerical solution must be consistent with conservation of total energy, we derive its form here. The potential energy can be found from the following considerations. If the rod is deflected by the angle θ , then the bob is raised by the distance $h = L - L \cos \theta$ (see Figure 6.2). Hence, the potential energy of the bob in the gravitational field of the earth can be expressed as

$$U = mgh = mgL(1 - \cos \theta), \quad (6.19)$$

where the zero of the potential energy corresponds to $\theta = 0$. Because the kinetic energy of the pendulum is $\frac{1}{2}mv^2 = \frac{1}{2}mL^2(d\theta/dt)^2$, the total energy E is

$$E = \frac{1}{2}mL^2\left(\frac{d\theta}{dt}\right)^2 + mgL(1 - \cos \theta). \quad (6.20)$$

In the following, we adopt the notation $\omega = d\theta/dt$ for the angular velocity.

We use two classes to solve simulate and visualize the motion of a pendulum problem, `Pendulum` and `PendulumApp`. The `Pendulum` class, a drawable class that solves the equation of motion, is coded as follows:

Listing 6.7: A Drawable pendulum.

```
package org.opensourcephysics.sip.ch6;
import java.awt.*;
import org.opensourcephysics.display.*;

public class Pendulum implements Drawable {
    protected double omega2;           // omega^2 = g/length
    protected double[] state = new double[3]; // current state
    Color color = Color.red;
    int pixRadius = 6;

    public Pendulum() { // create Pendulum with angular frequency 1
        this(1);
    }

    public Pendulum(double _omega2) { // create Pendulum with specified angular frequency
        omega2 = _omega2;
        state [0] = 0;           // theta
        state [1] = 0;           // angular velocity
        state [2] = 0;           // time
    }

    public void setState(double theta, double omega, double t) {
        state [0] = theta;
        state [1] = omega;
        state [2] = t;
    }

    public double[] getState() {
```

```

    return state;
}

public void step(double dt) {
    // use Verlet algorithm
    double accel = -omega2*Math.sin(state[0]);
    state[0] += state[1]*dt + 0.5*accel*dt*dt;           // angle
    state[1] += 0.5*(accel - omega2*Math.sin(state[0]))*dt; // angular velocity
    state[2] += dt;                                     // time
}

public void draw(DrawingPanel drawingPanel, Graphics g) {
    int xpivot = drawingPanel.xToPix(0);
    int ypivot = drawingPanel.yToPix(0);
    int xpix = drawingPanel.xToPix(Math.sin(state[0]));
    int ypix = drawingPanel.yToPix(-Math.cos(state[0]));
    g.setColor(Color.black);
    g.drawLine(xpivot, ypivot, xpix, ypix);
    g.setColor(color);
    g.fillOval(xpix - pixRadius, ypix - pixRadius, 2*pixRadius, 2*pixRadius); // draw bob
}
}

```

Note that the step method implements the Verlet algorithm.

Encapsulating drawing within the `Pendulum` class simplifies the `PendulumApp` code because it is unnecessary for the run method to pass data to a separate drawing object. Stepping the ODE automatically updates the drawing's position.

Listing 6.8: Visualization of the motion of a pendulum.

```

package org.opensourcephysics.sip.ch6;
import java.awt.Color;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;

public class PendulumApp implements Animation {

    Control myControl;
    // contains the graph
    PlottingPanel plottingPanel = new PlottingPanel("Time", "Theta", "angle vs time");
    DrawingFrame plottingFrame = new DrawingFrame(plottingPanel);
    // contains the animation
    DrawingPanel drawingPanel = new DrawingPanel();
    DrawingFrame drawingFrame = new DrawingFrame(drawingPanel);
    Thread animationThread;
    DatasetCollection dataset;
    Pendulum pendulum;
    double dt;

    public PendulumApp() {

```

```
pendulum = new Pendulum();
drawingPanel.setPreferredMinMax(-1.2, 1.2, -1.2, 1.2);
drawingPanel.addDrawable(pendulum);
dataset = new DatasetCollection();
dataset.setXYColumnNames(0, "time", "theta");
plottingPanel.addDrawable(dataset);
}

public void setControl(Control control) {
    myControl = control;
    resetAnimation();
}

public void startAnimation() {
    animationThread = new Thread(this);
    animationThread.start(); // start the animation
}

public void initializeAnimation() {
    dt = myControl.getDouble("dt"); // time step
    double theta = myControl.getDouble("theta");
    double omega = myControl.getDouble("omega");
    double[] state = pendulum.getState();
    pendulum.setState(theta, omega, state[2]);
    plottingPanel.repaint();
    drawingPanel.repaint();
}

public void stopAnimation() {
    Thread runningThread = animationThread;
    animationThread = null; // stop the animation
    try { // wait for the thread to die
        if (runningThread != null) {
            runningThread.join();
        }
    } catch(InterruptedException e) {}
    updateControlValues();
}

public void updateControlValues() {
    double[] state = pendulum.getState();
    myControl.setValue("theta", state[0]);
    myControl.setValue("omega", state[1]);
}

public void stepAnimation() {
    stepPendulum();
    plottingPanel.repaint();
    drawingPanel.repaint();
    updateControlValues();
}
```

```

}

public void stepPendulum() {
    double[] state = pendulum.getState(); // get the state
    dataset.append(0, state[2], state[0]); // add data to the curve
    pendulum.step(dt); // advance the SHO state by dt
}

public void resetAnimation() {
    dataset.clear();
    plottingPanel.repaint();
    drawingPanel.repaint();
    double[] state = pendulum.getState();
    double theta = 1;
    double omega = 1;
    double t = 0;
    pendulum.setState(theta, omega, t);
    updateControlValues();
    myControl.setValue("dt", 0.1);
}

public void run() {
    while(animationThread == Thread.currentThread()) {
        stepPendulum();
        plottingPanel.render(); // paint the plot
        drawingPanel.render(); // paint the animation
        try {
            animationThread.sleep(100);
        } catch(InterruptedException ie) {}
    }
}

public static void main(String[] args) {
    Animation app = new PendulumApp();
    AnimationControl control = new AnimationControl(app);
    app.setControl(control);
}
}

```

Problem 6.8. Oscillations of a pendulum

1. Modify the **Pendulum** class so that the acceleration is specified in its own method. That is, modify the rate equations in the step method so that they use the following method:

```

private double getAcceleration(){
    return -omega2*Math.sin(state[0]);
}

```

This change will make it easier to add friction and an external driving force in subsequent problems.

2. Make the necessary changes so that the analytical solution for small angles is also plotted.
3. Test the program at sufficiently small amplitude so that $\sin \theta \approx \theta$. Choose $g/L = 9$ and the initial condition $\theta(t = 0) = 0.1$, $\omega(t = 0) = 0$ and determine the period. Estimate the error due to the small angle approximation for these initial conditions.
4. Simulate large amplitude oscillations of a pendulum. Set $g/L = 9$ and choose Δt so the numerical algorithm generates a stable solution. Check the stability of the solution by monitoring the total energy and ensuring that it does not drift from its initial value.
5. Set $\omega(t = 0) = 0$ and make plots of $\theta(t)$ and $\omega(t)$ for the initial conditions $\theta(t = 0) = 0.1, 0.2, 0.4, 0.8$, and 1.0 . Remember that θ is measured in radians. Describe the qualitative behavior of θ and ω . What is the period T and the amplitude θ_m in each case? Plot T versus θ_m and discuss the qualitative dependence of the period on the amplitude. How do your results for T compare in the linear and nonlinear cases, or example, which period is larger? Explain the relative values of T in terms of the relative magnitudes of the restoring force in the two cases.

6.6 Damped Harmonic Oscillator

We know from experience that most oscillatory motion in nature gradually decreases until the displacement becomes zero; such motion is said to be *damped* and the system is said to be *dissipative* rather than conservative. As an example of a damped harmonic oscillator, consider the motion of the block in Figure 6.1 when a horizontal drag force is included. For small velocities, it is a reasonable approximation to assume that the drag force is proportional to the first power of the velocity. In this case the equation of motion can be written as

$$\frac{d^2x}{dt^2} = -\omega_0^2 x - \gamma \frac{dx}{dt}. \quad (6.21)$$

The *damping coefficient* γ is a measure of the magnitude of the drag term. Note that the drag force in (6.21) opposes the motion. We simulate the behavior of the damped linear oscillator in Problem 6.9.

Problem 6.9. Damped linear oscillator

1. Incorporate the effects of damping into your harmonic oscillator animation and plot the time dependence of the position and the velocity. Describe the qualitative behavior of $x(t)$ and $v(t)$ for $\omega_0 = 3$ and $\gamma = 0.5$ with $x(t = 0) = 1, v(t = 0) = 0$.
2. The period of the motion is the time between successive maxima of $x(t)$. Compute the period and corresponding angular frequency and compare their values to the undamped case. Is the period longer or shorter? Make additional runs for $\gamma = 1, 2$, and 3 . Does the period increase or decrease with greater damping? Why?
3. The amplitude is the maximum value of x during one cycle. Compute the *relaxation time* τ , the time it takes for the amplitude of an oscillation to decrease by $1/e \approx 0.37$ from its maximum value. Is the value of τ constant throughout the motion? Compute τ for the values of γ considered in part (b) and discuss the qualitative dependence of τ on γ .

4. Plot the total energy as a function of time for the values of γ considered in part (b). If the decrease in energy is not monotonic, explain.
5. Compute the time dependence of $x(t)$ and $v(t)$ for $\gamma = 4, 5, 6, 7$, and 8 . Is the motion oscillatory for all γ ? How can you characterize the decay? For fixed ω_0 , the oscillator is said to be *critically damped* at the smallest value of γ for which the decay to equilibrium is monotonic. For what value of γ does critical damping occur for $\omega_0 = 4$ and $\omega_0 = 2$? For each value of ω_0 , compute the value of γ for which the system approaches equilibrium most quickly.
6. Compute the phase space diagram for $\omega_0 = 3$ and $\gamma = 0.5, 2, 4, 6$, and 8 . Why does the phase space trajectory converge to the origin, $(x = 0, v = 0)$? This point is called an *attractor*. Convince yourself that points near the origin move toward the attractor; this attractor is said to be *stable*. Are these qualitative features of the phase space plot independent of γ ?

Problem 6.10. Damped nonlinear pendulum

Consider a damped pendulum with $g/L = 9$ and $\gamma = 1$ and the initial condition $\theta(t = 0) = 0.2, \omega(t = 0) = 0$. In what ways is the motion of the damped nonlinear pendulum similar to the damped linear oscillator? In what ways is it different? What is the shape of the phase space trajectory for the initial condition $\theta(t = 0) = 1, \omega(t = 0) = 0$? Do you find a different phase trajectory for other initial conditions? Remember that θ is restricted to be between $-\pi$ and $+\pi$.

6.7 Response to External Forces

How can we determine the period of a pendulum that is not already in motion? The obvious way is to disturb the system, for example, to displace the bob and observe its motion. We will find that the nature of the *response* of the system to the perturbation tells us something about the nature of the system in the absence of the perturbation.

Consider the *driven* damped linear oscillator with an external force $F(t)$ in addition to the linear restoring force and linear damping force. The equation of motion can be written as

$$\frac{d^2x}{dt^2} = -\omega_0^2 x - \gamma v + \frac{1}{m}F(t). \quad (6.22)$$

It is customary to interpret the response of the system in terms of the displacement x rather than the velocity v .

The time dependence of $F(t)$ in (6.22) is arbitrary. Because many forces are periodic, we first consider the form

$$\frac{1}{m}F(t) = A_0 \cos \omega t, \quad (6.23)$$

where ω is the angular frequency of the driving force. In Problem 6.11 we consider the response of the damped linear oscillator to (6.23).

Problem 6.11. Motion of a driven damped linear oscillator

1. Modify the harmonic oscillator animation so that an external force of the form (6.23) is included. Add this force in the class that encapsulates the equations of motion without changing the target class. The angular frequency of the driving force, w should be added as an input parameter.
2. Set $\omega_0 = 3$, $\gamma = 0.5$, $\omega = 2$ and the amplitude of the external force $A_0 = 1$ for all runs unless otherwise stated. We know that for these values of ω_0 and γ , the dynamical behavior in the absence of an external force corresponds to a underdamped oscillator. Plot $x(t)$ versus t in the presence of the external force with the initial condition, $x(t = 0) = 1, v(t = 0) = 0$. How does the qualitative behavior of $x(t)$ differ from the nonperturbed case? What is the period and angular frequency of $x(t)$ after several oscillations have occurred? Repeat the same observations for $x(t)$ with $x(t = 0) = 0, v(t = 0) = 1$. Does $x(t)$ approach a limiting behavior independently of the initial conditions? Does the short time behavior of $x(t)$ depend on the initial conditions? Identify a *transient* part of $x(t)$ that depends on the initial conditions and decays in time, and a *steady state* part that dominates at longer times and is independent of the initial conditions.
3. Compute $x(t)$ for several combinations of ω_0 and ω . What is the period and angular frequency of the steady state motion in each case? What parameters determine the frequency of the steady state behavior?
4. One measure of the long-term behavior of the driven harmonic oscillator is the amplitude of the steady state displacement $A(\omega)$. The amplitude function $A(\omega)$ can be computed easily if we set ω and run the system until steady state has been achieved. We then test the position after every time step to see if a new maximum has been reached.

```

if (x > Math.abs(max)) {
    max = Math.abs(x);
    System.out.println("new max =" + max);
}

```

5. Verify that the steady state behavior of $x(t)$ is given by

$$x(t) = A(\omega) \cos(\omega t + \delta). \quad (6.24)$$

The quantity δ is the phase difference between the applied force and the steady state motion. Compute $A(\omega)$ and $\delta(\omega)$ for $\omega_0 = 3$, $\gamma = 0.5$, and $\omega = 0, 1.0, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0, 3.2$, and 3.4 . Choose the initial condition, $x(t = 0) = 0, v(t = 0) = 0$. Repeat the simulation for $\gamma = 3.0$, and plot $A(\omega)$ and $\delta(\omega)$ versus ω for the two values of γ . Discuss the qualitative behavior of $A(\omega)$ and $\delta(\omega)$ for the two values of γ . If $A(\omega)$ has a maximum, determine the angular frequency ω_m at which the maximum of A occurs. Is the value of ω_m close to the natural angular frequency ω_0 ? Compare ω_m to ω_0 and to the frequency of the damped linear oscillator in the absence of an external force.

6. Compute $x(t)$ and $A(\omega)$ for a damped linear oscillator with the amplitude of the external force $A_0 = 4$. How do the steady state results for $x(t)$ and $A(\omega)$ compare to the case $A_0 = 1$? Does the transient behavior of $x(t)$ satisfy the same relation as the steady state behavior?

7. What is the shape of the phase space trajectory for the initial condition $x(t = 0) = 1, v(t = 0) = 0$? Do you find a different phase trajectory for other initial conditions?
8. Describe the qualitative behavior of the steady state amplitude $A(\omega)$ near $\omega = 0$ and $\omega \gg \omega_0$. Why is $A(\omega = 0) < A(\omega)$ for small ω ? Why does $A(\omega) \rightarrow 0$ for $\omega \gg \omega_0$?
9. Does the mean kinetic energy resonate at the same frequency as does the amplitude? Compute the mean kinetic energy over one cycle once steady state conditions have been reached. Choose $\omega_0 = 3$ and $\gamma = 0.5$.

In Problem 6.11 we found that the response of the damped harmonic oscillator to an external driving force is linear. For example, if the magnitude of the external force is doubled, then the magnitude of the steady state motion also is doubled. This behavior is a consequence of the linear nature of the equation of motion. When a particle is subject to nonlinear forces, the response can be much more complicated (see Section ?? [in Chaos chapter]).

For many problems, the sinusoidal driving force in (6.23) is not realistic. Another example of an external force can be found by observing someone pushing a child on a swing. Because the force is nonzero only for short intervals of time, this type of force is impulsive. In the following problem, we consider the response of a damped linear oscillator to an impulsive force.

**Problem 6.12.* Response of a damped linear oscillator to nonsinusoidal external forces

1. Assume a swing can be modelled by a linear restoring force and a linear damping term. The effect of an impulse is to change the velocity. For simplicity, let the duration of the push equal the time step Δt . Introduce an integer variable for the number of time steps and use the `mod` function to ensure that the impulse is nonzero only at the time interval associated with the period of the external impulse.
2. Determine the steady state amplitude $A(\omega)$ for $\omega = 1.0, 1.3, 1.4, 1.5, 1.6, 2.5, 3.0$, and 3.5 . The corresponding period of the impulse is given by $T = 2\pi/\omega$. Choose $\omega_0 = 3$ and $\gamma = 0.5$. Are your results consistent with your experience in pushing a swing and with the comparable results of Problem 6.11?
3. Consider the response to the half-wave external force consisting of the positive part of a cosine function (see Figure 6.3). Compute $A(\omega)$ for $\omega_0 = 3$ and $\gamma = 0.5$. At what values of ω does $A(\omega)$ have a relative maxima? Is the half-wave cosine driving force equivalent to a sum of cosine functions of different frequencies? For example, does $A(\omega)$ have more than one resonance?
4. Compute the steady state response $x(t)$ to the external force

$$\frac{1}{m}F(t) = \frac{1}{\pi} + \frac{1}{2}\cos t + \frac{2}{3\pi}\cos 2t - \frac{2}{15\pi}\cos 4t. \quad (6.25)$$

How does a plot of $F(t)$ versus t compare to the half-wave cosine function? Use your results to conjecture a principle of superposition for the solutions to linear equations.

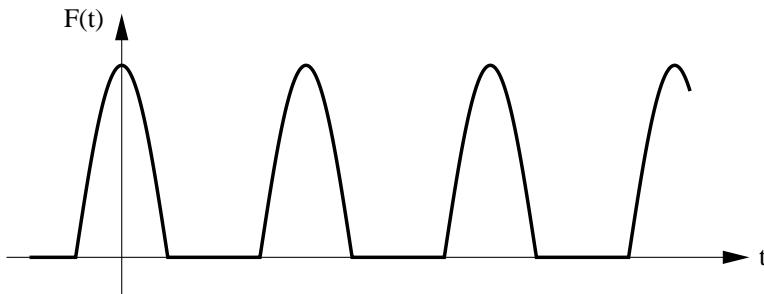


Figure 6.3: A half-wave driving force corresponding to the positive part of a cosine function.

element	voltage drop	units
resistor	$V_R = IR$	resistance R , ohms (Ω)
capacitor	$V_C = Q/C$	capacitance C , farads (F)
inductor	$V_L = L dI/dt$	inductance L , henries (H)

Table 6.1: The voltage drops across the basic electrical circuit elements. Q is the charge (coulombs) on one plate of the capacitor, and I is the current (amperes).

6.8 Electrical Circuit Oscillations

In this section we discuss several electrical analogues of the mechanical systems we have considered. Although the equations of motion are identical in form, it is convenient to consider electrical circuits separately, because the nature of the questions is somewhat different.

The starting point for electrical circuit theory is Kirchhoff's loop rule, which states that the sum of the voltage drops around a closed path of an electrical circuit is zero. This law is a consequence of conservation of energy, because a voltage drop represents the amount of energy that is lost or gained when a unit charge passes through a circuit element. The relationships for the voltage drops across each circuit element are summarized in Table 6.1.

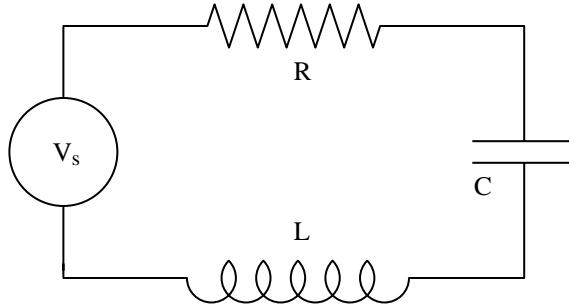
Imagine an electrical circuit with an alternating voltage source $V_s(t)$ attached in series to a resistor, inductor, and capacitor (see Figure 6.4). The corresponding loop equation is

$$V_L + V_R + V_C = V_s(t). \quad (6.26)$$

The voltage source term V_s in (6.26) is the *emf* and is measured in units of volts. If we substitute the relationships shown in Table 6.1, we find

$$L \frac{d^2Q}{dt^2} + R \frac{dQ}{dt} + \frac{Q}{C} = V_s(t), \quad (6.27)$$

where we have used the definition of current $I = dQ/dt$. We see that (6.27) for the series RLC circuit is identical in form to the damped harmonic oscillator (6.22). The analogies between ideal electrical circuits and mechanical systems are summarized in Table 6.2.

Figure 6.4: A simple series RLC circuit with a voltage source V_s .

Electric circuit	Mechanical system
charge Q	displacement x
current $I = dQ/dt$	velocity $v = dx/dt$
voltage drop	force
inductance L	mass m
inverse capacitance $1/C$	spring constant k
resistance R	damping γ

Table 6.2: Analogies between electrical parameters and mechanical parameters.

Although we are already familiar with (6.27), we first consider the dynamical behavior of an RC circuit described by

$$R \frac{dQ}{dt} = RI(t) = V_s(t) - \frac{Q}{C}. \quad (6.28)$$

Two RC circuits corresponding to (6.28) are shown in Figure 6.5. Although the loop equation (6.28) is identical regardless of the order of placement of the capacitor and resistor in Figure 6.5, the output voltage measured by the oscilloscope in Figure 6.5 is different. We will see in Problem 6.13 that these circuits act as *filters* that pass voltage components of certain frequencies while rejecting others.

An advantage of a computer simulation of an electrical circuit is that the measurement of a voltage drop across a circuit element does not affect the properties of the circuit. In fact, digital computers often are used to optimize the design of circuits for special applications. The RCApp program simulates an RC circuit with an alternating current (ac) voltage source of the form $V_s(t) = \cos \omega t$ and shows the time dependencies of the voltage across the capacitor in a plotting panel. You are asked to modify this program in Problem 6.13.

Problem 6.13. Simple filter circuits

1. Copy the RCApp and RC classes from the chapter 6 package to a working directory. Modify the RCApp program to plot the voltage across the resistor, V_R , and the voltage across the source, V_s , in addition to the voltage across the capacitor, V_C . Run this program with $R = 1000 \Omega$

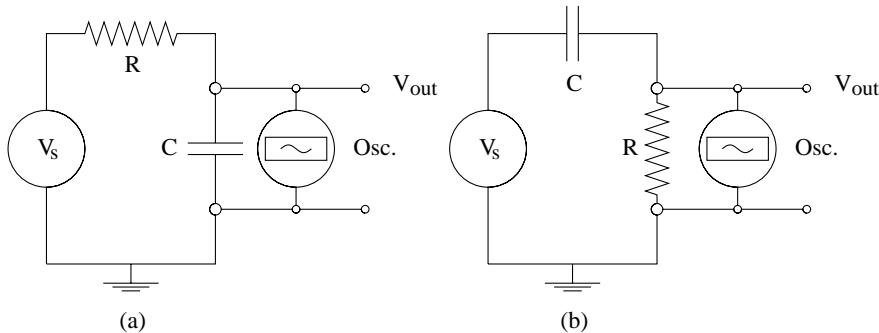


Figure 6.5: Examples of RC circuits used as low and high pass filters. Which circuit is which?

and $C = 1.0 \mu F$ (10^{-6} farads). Find the steady state amplitude of the voltage drops across the resistor and across the capacitor as a function of the angular frequency ω of the source voltage $V_s = \cos \omega t$. Consider the frequencies $f = 10, 50, 100, 160, 200, 500, 1000, 5000$, and 10000 Hz. (Remember that $\omega = 2\pi f$.) Choose Δt to be no more than 0.0001 s for $f = 10$ Hz. What is a reasonable value of Δt for $f = 10000$ Hz?

- The output voltage depends on where the digital oscilloscope is connected. What is the output voltage of the oscilloscope in Figure 6.5a? Plot the ratio of the amplitude of the output voltage to the amplitude of the input voltage as a function of ω . Use a logarithmic scale for ω . What range of frequencies is passed? Does this circuit act as a high pass or a low pass filter? Answer the same questions for the oscilloscope in Figure 6.5b. Use your results to explain the operation of a high and low pass filter. Compute the value of the cutoff frequency for which the amplitude of the output voltage drops to $1/\sqrt{2}$ (half-power) of the input value. How is the cutoff frequency related to RC ?
 - Plot the voltage drops across the capacitor and resistor as a function of time. The phase difference ϕ between each voltage drop and the source voltage can be found by finding the time t_m between the corresponding maxima of the voltages. Because ϕ is usually expressed in radians, we have the relation $\phi/2\pi = t_m/T$, where T is the period of the oscillation. What is the phase difference ϕ_C between the capacitor and the voltage source and the phase difference ϕ_R between the resistor and the voltage source? Do these phase differences depend on ω ? Does the current lead or lag the voltage, that is, does the maxima of $V_R(t)$ come before or after the maxima of $V_s(t)$? What is the phase difference between the capacitor and the resistor? Does the latter difference depend on ω ?
 - Modify the RCApp program to find the steady state response of an LR circuit with a source voltage $V_s(t) = \cos \omega t$. Let $R = 100 \Omega$ and $L = 2 \times 10^{-3} \text{ H}$. Because $L/R = 2 \times 10^{-5} \text{ s}$, it is convenient to measure the time and frequency in units of $T_0 = L/R$. We write $t^* = t/T_0$, $\omega^* = \omega T_0$, and rewrite the equation for an LR circuit as

$$I(t^*) + \frac{dI(t^*)}{dt^*} = \frac{1}{R} \cos \omega^* t^*. \quad (6.29)$$

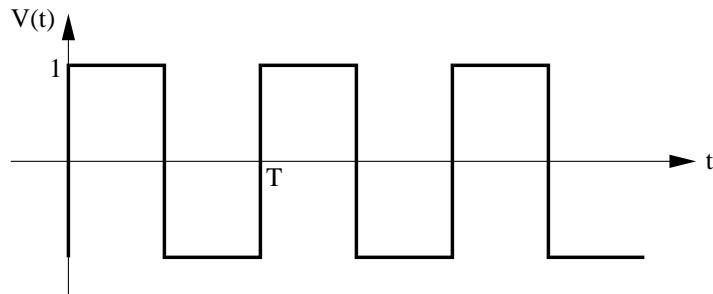


Figure 6.6: Square wave voltage with period T and unit amplitude.

Because it will be clear from the context, we now simply write t and ω rather than t^* and ω^* . What is a reasonable value of the step size Δt ? Compute the steady state amplitude of the voltage drops across the inductor and the resistor for the input frequencies $f = 10, 20, 30, 35, 50, 100,$ and 200 Hz . Use these results to explain how an LR circuit can be used as a low pass or a high pass filter. Plot the voltage drops across the inductor and resistor as a function of time and determine the phase differences ϕ_R and ϕ_L between the resistor and the voltage source and the inductor and the voltage source. Do these phase differences depend on ω ? Does the current lead or lag the voltage? What is the phase difference between the inductor and the resistor? Does the latter difference depend on ω ?

Problem 6.14. Square wave response of an RC circuit

Modify the **RCApp** program so that the voltage source is a periodic square wave as shown in Figure 6.6. Use a $1.0\ \mu\text{F}$ capacitor and a $3000\ \Omega$ resistor. Plot the computed voltage drop across the capacitor as a function of time. Make sure the period of the square wave is long enough so that the capacitor is fully charged during one half-cycle. What is the approximate time dependence of $V_C(t)$ while the capacitor is charging (discharging)?

We now consider the steady state behavior of the series RLC circuit shown in Figure 6.4 and represented by (6.27). The response of an electrical circuit is the current rather than the charge on the capacitor. Because we have simulated the analogous mechanical system, we already know much about the behavior of driven RLC circuits. Nonetheless, we will find several interesting features of ac electrical circuits in the following two problems.

Problem 6.15. Response of an RLC circuit

1. Consider an RLC series circuit with $R = 100\ \Omega$, $C = 3.0\ \mu\text{F}$, and $L = 2\ \text{mH}$. Modify the simple harmonic oscillator program or the RC filter program to simulate an RLC circuit and compute the voltage drops across the three circuit elements. Assume an ac voltage source of the form $V(t) = V_0 \cos \omega t$. Plot the current I as a function of time and determine the maximum steady state current I_m for different values of ω . Obtain the *resonance curve* by plotting $I_m(\omega)$ as a function of ω and compute the value of ω at which the resonance curve is a maximum. This value of ω is the *resonant frequency*.

2. The sharpness of the resonance curve of an ac circuit is related to the quality factor or Q value. (Q should not be confused with the charge on the capacitor.) The sharper the resonance, the larger the Q . Circuits with high Q (and hence a sharp resonance) are useful for tuning circuits in a radio so that only one station is heard at a time. We define $Q = \omega_0/\Delta\omega$, where the width $\Delta\omega$ is the frequency interval between points on the resonance curve $I_m(\omega)$ that are $\sqrt{2}/2$ of I_m at its maximum. Compute Q for the values of R , L , and C given in part (a). Change the value of R by 10% and compute the corresponding percentage change in Q . What is the corresponding change in Q if L or C is changed by 10%?
3. Compute the time dependence of the voltage drops across each circuit element for approximately fifteen frequencies ranging from 1/10 to 10 times the resonant frequency. Plot the time dependence of the voltage drops.
4. The ratio of the amplitude of the sinusoidal source voltage to the amplitude of the current is called the *impedance* Z of the circuit, that is, $Z = V_m/I_m$. This definition of Z is a generalization of the resistance that is defined by the relation $V = IR$ for direct current circuits. Use the plots of part (c) to determine I_m and V_m for different frequencies and verify that the impedance is given by

$$Z(\omega) = \sqrt{R^2 + (\omega L - 1/\omega C)^2}. \quad (6.30)$$

For what value of ω is Z a minimum? Note that the relation $V = IZ$ holds only for the maximum values of I and V and not for I and V at any time.

5. Compute the phase difference ϕ_R between the voltage drop across the resistor and the voltage source. Consider $\omega \ll \omega_0$, $\omega = \omega_0$, and $\omega \gg \omega_0$. Does the current lead or lag the voltage in each case, that is, does the current reach a maxima before or after the voltage? Also compute the phase differences ϕ_L and ϕ_C and describe their dependence on ω . Do the relative phase differences between V_C , V_R , and V_L depend on ω ?
6. Compute the amplitude of the voltage drops across the inductor and the capacitor at the resonant frequency. How do these voltage drops compare to the voltage drop across the resistor and to the source voltage? Also compare the relative phases of V_C and V_L at resonance. Explain how an RLC circuit can be used to amplify the input voltage.

6.9 Projects

Project 6.16. Chemical oscillations

The kinetics of chemical reactions can be modeled by a system of coupled first-order differential equations. As an example, consider the following reaction



where A , B , and C represent the concentrations of three different types of molecules. The corresponding rate equations for this reaction are

$$\frac{dA}{dt} = -kAB^2 \quad (6.32a)$$

$$\frac{dB}{dt} = kAB^2 \quad (6.32b)$$

$$\frac{dC}{dt} = kAB^2. \quad (6.32c)$$

The rate at which the reaction proceeds is determined by the reaction constant k . The terms on the right-hand side of (6.32) are positive if the concentration of the molecule increases in (6.31) as it does for B and C , and negative if the concentration decreases as it does for A . Note that the term $2B$ in the reaction (6.31) appears as B^2 in the rate equation (6.32).

Most chemical reactions proceed to equilibrium, where the mean concentrations of all molecules are constant. However, if the concentrations of some molecules are replenished, it is possible to observe other kinds of behavior, such as oscillations (see below) and chaotic behavior (see Project ??). In (6.32) we have assumed that the reactants are well stirred, so that there are no spatial inhomogeneities. In Section ?? we will discuss the effects of spatial inhomogeneities due to molecular diffusion.

To obtain chemical oscillations, it is essential to have a series of chemical reactions such that the products of some reactions are the reactants of others. In the following, we consider a simple set of reactions that can lead to oscillations under certain conditions (see Lefever and Nicolis):



If we assume that the reverse reactions are negligible and A and B are held constant by an external source, the corresponding rate equations are

$$\frac{dX}{dt} = A - (B + 1)X + X^2Y \quad (6.34a)$$

$$\frac{dY}{dt} = BX - X^2Y. \quad (6.34b)$$

For simplicity, we have chosen the rate constants to be unity.

1. The steady state solution of (6.34) can be found by setting the left-hand side equal to zero. Show that the steady state values for (X, Y) are $(A, B/A)$.
2. Write a program to solve numerically the rate equations given by (6.34). A simple Euler algorithm is sufficient. Your program should input the initial values of X and Y and the fixed concentrations A and B , and plot X versus Y as the reactions evolve.
3. Systematically vary the initial values of X and Y for given values of A and B . Are their steady state behaviors independent of the initial conditions?

4. Let the initial value of (X, Y) equal $(A + 0.001, B/A)$ for several different values of A and B , that is, choose initial values close to the steady state values. Classify which initial values result in steady state behavior (stable) and which ones show periodic behavior (unstable). Find the relation between A and B that separates the two types of behavior.

Project 6.17. Comparison of algorithms

1. Consider a particle of unit mass moving in the Morse potential:

$$V(x) = e^{-2x} - 2e^{-x}. \quad (6.35)$$

The total energy $E = \frac{1}{2}p^2 + V(x) < 0$, and the force on the particle is given by

$$F(x) = -\frac{dV}{dx} = 2e^{-x}(e^{-x} - 1). \quad (6.36)$$

Plot $V(x)$ and $F(x)$ versus x . What is their qualitative dependence on x ? What is the total energy for the initial condition $x_0 = 2$ and $v_0 = 0$? What type of motion do you expect?

2. Compare the Euler, Euler-Richardson, and Verlet algorithms by computing $x(t)$, $v(t)$, and E_n , where E_n is the total energy after the n th step. One measure of the error associated with the algorithm is ΔE_{\max} , the maximum value of the difference $|E_n - E_0|$ over the time interval $t_n - t_0$. Compute this measure of the error at a given value of t_n for decreasing values of Δt with the initial condition $(x_0 = 2, v_0 = 0)$. If an algorithm is second-order, the error in the total energy should decrease as $(\Delta t)^2$. The absence of such a decrease might indicate that there is an error in your program or that roundoff errors are important. Which one of the above algorithms is the best trade-off between speed, accuracy, and simplicity?
3. Compare your results for $x(t)$ to the analytical result

$$x(t) = \ln(\alpha \cos \omega t + \beta \sin \omega t - E_0^{-1}) \quad (6.37)$$

with

$$\alpha = e^{x_0} + E_0^{-1} \quad (6.38)$$

$$\beta = \omega^{-1} v_0 e^{x_0} \quad (6.39)$$

$$\omega = (2|E_0|)^{1/2}. \quad (6.40)$$

4. Repeat the computations of part (a) with $x_0 = 3, v_0 = 1$. Is the motion periodic? Which algorithm is most suitable in this case?

References and Suggestions for Further Reading

F. S. Acton, *Numerical Methods That Work*, The Mathematical Association of America (1990), Chapter 5.

- G. L. Baker and J. P. Gollub, *Chaotic Dynamics: An Introduction*, Cambridge University Press (1990). A good introduction to the notion of phase space.
- A. Douglas Davis, *Classical Mechanics*, Academic Press (1986). The author gives a “crash” course in conversational BASIC and Pascal and simple numerical solutions of Newton’s equations of motion. Much emphasis is given to the harmonic oscillator problem.
- S. Eubank, W. Miner, T. Tajima, and J. Wiley, “Interactive computer simulation and analysis of Newtonian dynamics,” *Amer. J. Phys.* **57**, 457 (1989).
- Richard P. Feynman, Robert B. Leighton, and Matthew Sands, *The Feynman Lectures on Physics*, Vol. 1, Addison-Wesley (1963). Chapters 21, 23, 24, and 25 are devoted to various aspects of harmonic motion.
- Charles Kittel, Walter D. Knight, and Malvin A. Ruderman, *Mechanics*, second edition, revised by A. Carl Helmholtz and Burton J. Moyer, McGraw-Hill (1973).
- R. Lefever and G. Nicolis, “Chemical instabilities and sustained oscillations,” *J. Theor. Biol.* **30**, 267 (1971).
- Jerry B. Marion and Stephen T. Thornton, *Classical Dynamics*, fourth edition, Academic Press (1995). Excellent discussion of linear and nonlinear oscillators.
- M. F. McInerney, “Computer-aided experiments with the damped harmonic oscillator,” *Am. J. Phys.* **53**, 991 (1985).
- William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes*, second edition, Cambridge University Press (1992). Chapter 16 discusses the integration of ordinary differential equations.
- J. C. Sprott, *Introduction to Modern Electronics*, John Wiley & Sons (1981). The first five chapters treat the topics discussed in Section 6.8.
- S. C. Zilio, “Measurement and analysis of large-angle pendulum motion,” *Am. J. Phys.* **50**, 450 (1982).

There are many good books on Java drawing and Java threads. We list a few of our favorites:

- David M. Geary, *Graphic Java: Mastering the JFC*, Vol. 1 AWT and Vol. 2 Swing, Prentice Hall (1999).
- Jonathan Knudsen, *Java 2D Graphics*, O’Reilly (1999).
- Scott Oaks and Henry Wong, *Java Threads*, second edition, O’Reilly (1999).

Chapter 7

Random Processes

©2001 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
4 April 2001

Random processes are introduced in the context of several simple physical systems.

7.1 Order to Disorder

In Chapter 6 we saw several examples of how under certain conditions, the behavior of nonlinear deterministic systems can be so complicated that it can be described as random. In this chapter we will see some examples of how chance can generate statistically predictable outcomes. For example, we know that if we bet often enough on the outcome of a game for which the outcome is determined by chance, we will lose money eventually if the probability of winning is less than 50%.

We first give an example that illustrates the tendency of many particle systems to evolve toward a well defined state. Imagine a closed box that is divided into two parts of equal volume (see Figure 7.1). The left half contains a gas of N identical particles and the right half is empty. We then make a small hole in the partition between the two halves. What happens? We know that after some time, the system reaches equilibrium, and the average number of particles in each half of the box is $N/2$.

How can we simulate this process? One way is to give each particle an initial velocity and position and adopt a simple deterministic model of the motion of the particles. We could assume that each particle moves in a straight line until it hits a wall of the box or another particle and undergoes an elastic collision. We will consider similar deterministic models in Chapter 8. Instead, we consider a simpler approach based on the simulation of a *random process*.

The basic assumption underlying the probabilistic model that we will consider is that the motion of the particles is such that their trajectory is random. For simplicity, we assume that the particles do not interact with one another so that the probability per unit time that a particle goes through the hole is the same for all particles regardless of the number of particles in either half.

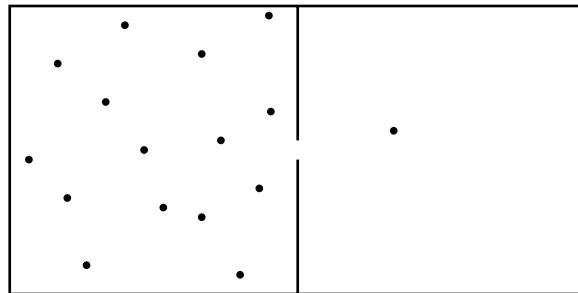


Figure 7.1: A box is divided into two equal halves by a partition. After a small hole is opened in the partition, one particle can pass through the hole per unit time.

We also assume that the size of the hole is such that only one particle can pass through it in one unit of time.

Assume that our model consists of N noninteracting particles, all of which are initially on the left-hand side. One way to implement move a particle is to choose a particle at random and move it to the other side. For example, we could use an array to specify the position of each particle and simulate the random process by generating at random an integer i between 1 and N and changing the array appropriately. The tool we need to do this simulation and other simulations of probabilistic systems is a random number generator.

It might seem strange that we can use a deterministic computer to generate sequences of random numbers. In Chapter 12 we discuss some of the methods for computing a set of numbers that appear statistically random, but are in fact generated by a deterministic algorithm. These algorithms are sometimes called pseudorandom number generators to distinguish their output from intrinsically random physical processes such as the time between clicks in a Geiger counter near a radioactive sample.

For the present we will be content to use the random number generator supplied with various programming languages, although these random number generators vary greatly in quality. In Java the method `Math.random()` produces a random number r that is uniformly distributed in the interval $0 \leq r < 1$. To generate a random integer i between 1 and N we can write:

```
int i = (int)(N*Math.random()) + 1;
```

Because the effect of the `int` function is to round the output of `rnd` to its nearest integer, it is necessary to add 1.

The procedure we have specified is needlessly cumbersome, because our only interest is the number of particles on each side. That is, we need to know only n , the number of particles on the left side; the number on the right side is $n' = N - n$. Because each particle has the same chance to go through the hole, the probability per unit time that a particle moves from left to right equals the number of particles on the left side divided by the total number of particles, that is, the probability of a move from left to right is n/N . The algorithm for simulating the evolution of the model can be summarized by the following steps:

1. Generate a random number r from a uniformly distributed set of random numbers in the interval $0 \leq r < 1$.
2. Compare r to the current value of the fraction of particles n/N on the left side of the box.
3. If $r \leq n/N$, move a particle from left to right, that is, let $n \rightarrow n - 1$; otherwise, move a particle from right to left.
4. Increase the “time” by unity.

Note that the above definition of time is arbitrary. Class `Box` implements this algorithm and plots the evolution of n .

```
// 4/5/01, 10:20 am
package edu.clarku.sip.chapter7;
import edu.clarku.sip.plot.*;
import edu.clarku.sip.templates.*;

// simulation of the particles in a box problem
public class Box implements AnimationModel, Runnable
{
    // N should be lower case according to Java convention for variables
    private int N;          // total number of particles
    private int nleft;       // number on left
    private int time;
    private Control myControl = new SAnimationControl(this);
    private Plot plot;
    private Thread thread;
    private int dataSet = -1;

    public Box()
    {
        plot = new Plot("time", "nleft", "Approach to equilibrium");
        plot.setYMinimum(0);
    }

    public void step()
    {
        plot.addPoint(dataSet, time, nleft);
        plot.render();
        moveParticle();
    }

    public void reset()
    {
        myControl.setValue("N", 64);
    }
}
```

```
public void stopCalculation()
{
    thread = null;
}

public void continueCalculation()
{
    thread = new Thread(this);
    thread.start();
}

public void clear(){plot.deleteAllPoints(); dataSet = -1; }

public void startCalculation()
{
    dataSet++;
    N = (int) myControl.getValue("N");
    time = 0;
    nleft = N;          // all particles initially on left side
    thread = new Thread(this);
    thread.start();
}

public void run()
{
    while (thread == Thread.currentThread())
        step();
}

// move particle through hole
public void moveParticle()
{
    // generate random number and move particle
    double r = Math.random();
    double ratio = (double) nleft/N;
    if (r <= ratio)
        nleft--;
    else
        nleft++;
    time++;
}

public static void main(String[] args)
{
    Box b = new Box();
```

```

    b.reset();
}
}
}
```

How long does it take for the system to reach equilibrium? How does this time depend on the number of particles? After the system reaches equilibrium, what is the magnitude of the fluctuations? How do the fluctuations depend on the number of particles? Problems 7.2 and 7.3 address such questions.

Exercise 7.1. Some simple tests

- It is frequently quicker to write a short program to test the properties of a function than to look it up in a manual. For example, what are the values of `(int) 3/2` and `(int) -3/2`.
- Write a little program to test the nature of `Math.round(arg)`, `Math.ceil(arg)`, and `Math.rint`.
- Write a short program to test whether the same sequence of random numbers appears each time the program is run if we use the method class `Math.random()` to generate the sequence. In Chapter 12 we will learn about ways of generating random numbers and ways that we can set the seed in Java so that we can generate the same sequence if we wish. The ability to do the latter is essential for testing your programs.

Problem 7.2. Approach to equilibrium

- Run the program and describe the time evolution of n , the number of particles on the left side of the box. Choose the total number of particles N to be $N = 8, 16, 64, 400, 800$, and 3600. Estimate the time for the system to reach equilibrium from the plots. What is your qualitative criterion for equilibrium. How does this time depend on N ? What criterion did you use for equilibrium? Does n change when the system is in equilibrium?
- For sufficiently large N , does the time dependence of n appear to be deterministic? Based on the shape of your plots of $n(t)$, what is the qualitative behavior of $n(t)$ before equilibrium is reached?

Problem 7.3. Equilibrium fluctuations

- Modify the program so that averages are taken after equilibrium has been reached. What is the maximum deviation of $n(t)$ from $N/2$ for $N = 64, 400, 800$, and 3600? Run for a time that is long enough to yield meaningful results. How do you know if your results are meaningful? How do your results for the maximum deviation depend on N ?
- A measure of the equilibrium fluctuations is the variance σ^2 defined as

$$\sigma^2 = \overline{(n - \bar{n})^2} = \overline{n^2} - \bar{n}^2. \quad (7.1)$$

The bar denotes a time average taken after the system has reached equilibrium. The relative magnitude of the fluctuations is σ/\bar{n} . Compute the variance of n for the same values of N considered in part (a). How do the relative fluctuations, σ/\bar{n} , depend on N ?

From Problem 7.2 we see that $n(t)$ decreases in time from its initial value to its equilibrium value in an almost deterministic manner if $N \gg 1$. It is instructive to derive the time dependence of $n(t)$ to show explicitly how chance can generate deterministic behavior. If there are $n(t)$ particles on the left side after t moves, then the change in $n(t)$ in the time interval Δt is given by

$$\Delta n = \left[\frac{-n(t)}{N} + \frac{N - n(t)}{N} \right] \Delta t. \quad (7.2)$$

(Recall that we defined the time so that the time interval $\Delta t = 1$ in our simulations.) What is the meaning of the two terms in (7.2)? If we treat n and t as continuous variables and take the limit $\Delta t \rightarrow 0$, we have

$$\frac{dn}{dt} = 1 - \frac{2n(t)}{N}. \quad (7.3)$$

The solution of the differential equation (7.3) is

$$n(t) = \frac{N}{2} [1 + e^{-2t/N}], \quad (7.4)$$

where we have used the initial condition $n(t = 0) = N$. How does the exponential form (7.4) compare to your Monte Carlo results for various values of N ? We can define a *relaxation time* τ as the time it takes $n(t)$ to decrease to $1/e$ of its initial value. How does τ depend on N ? Does this prediction for τ agree with your results from Problem 7.2?

**Problem 7.4.* A simple modification

Modify your program so that each side of the box is chosen with equal probability. A particle is then moved from the side chosen to the other side. If the side chosen does not have a particle in it, then no particle is moved during this time interval. Do you expect that the system behaves in the same way as before? Do the simulation starting with all the particles on the left side of the box and choose $N = 800$. Compare the behavior of $n(t)$ with your predicted behavior, and with the behavior of $n(t)$ found in Problem 7.3. How do the values of \bar{n} and σ^2 compare? Is this variation of the model realistic?

The above probabilistic method for simulating the approach to equilibrium is an example of a *Monte Carlo* method, that is, the random sampling of the most probable outcomes. An alternative method is to use *exact enumeration* and to determine all the possibilities at each time interval. For example, suppose that at $t = 0$, $n = 8$, and $n' = 0$. At $t = 1$, the only possibility is $n = 7$ and $n' = 1$. Hence, $P(n = 7, t = 1) = 1$ and all other probabilities are zero. At $t = 2$, one of the seven particles on the left can move to the right, or the one particle on the right can move to the left. Because the first possibility can occur in seven different ways, we have the nonzero probabilities, $P(n = 6, t = 2) = 7/8$ and $P(n = 8, t = 2) = 1/8$. Hence at $t = 2$, the average number of particles on the left side of the box is

$$\langle n(t = 2) \rangle = 6P(6, 2) + 8P(8, 2) = \frac{1}{8}[6 \times 7 + 8 \times 1] = 6.25.$$

We have denoted the average by the brackets $\langle \dots \rangle$ to distinguish it from the time average that was computed in Problem 7.3. Is this exact result consistent with what you found in Problem 7.2? In this example N is small, and we can continue the enumeration of all the possibilities indefinitely.

However for larger N , the number of possibilities becomes very large after a few time intervals, and we are forced to use Monte Carlo methods.

So far we have run each simulation only once. Clearly, running a Monte Carlo simulation only once cannot reproduce the exact enumeration results, because in general, each run gives somewhat different outcomes if we use a different sequence of random numbers. In general, we need to do a Monte Carlo simulation many times and average over the results to obtain meaningful averages. Each run is called a *trial* or a *sample*. How do you know how many trials to use? The answer usually can be obtained empirically by averaging over more and more trials until the average results do not change within the desired level of accuracy.

7.2 Introduction to Random Walks

In Section 7.1 we considered the random motion of many particles in a box, but we did not care about their trajectories—all we needed to know was the number of particles on each side. Now suppose that we want to characterize the motion of a dust particle in a glass of water. We know that as a given dust particle collides with the water molecules, it changes its direction frequently. Its motion appears so erratic that we cannot predict its trajectory after even a small number of collisions. These considerations suggest that a simple model for the trajectory of a dust particle is that it moves in any direction with equal probability. Such a model is an example of a *random walk*.

The original statement of a random walk was formulated in the context of a “drunken sailor.” If a drunkard begins at a lamp post and takes N steps of equal length in random directions, how far will the drunkard be from the lamp post? We will find that the mean square displacement of a random walker, for example, a molecule or a drunkard, grows linearly with time. This result and its relation to diffusion leads to many applications that might seem to be unrelated to the original drunken sailor problem.

We first consider an idealized one-dimensional example of a random walker that can move only along a line. Suppose that the walker begins at $x = 0$ and that each step is of equal length ℓ . At each interval of time the walker has a probability p of a step to the right and a probability $q = 1 - p$ of a step to the left. The direction of each step is independent of the preceding one. After N steps the displacement x of a walker is given by

$$x_N = \sum_{i=1}^N s_i, \quad (7.5)$$

and the displacement squared x^2 is

$$x_N^2 = \left(\sum_{i=1}^N s_i \right)^2, \quad (7.6)$$

where $s_i = \pm \ell$. We can generate one walk of N steps by flipping a coin N times and increasing x by ℓ each time the coin is heads and decreasing x by ℓ each time the coin is tails. For simplicity, we first assume $p = q = \frac{1}{2}$. We expect that if we average over a sufficient number of walks of N

steps, then the average of x_N , denoted by $\langle x_N \rangle$, would be zero. That is, we expect to find as many steps in one direction as in the opposite direction.

To find $\langle x_N^2 \rangle$ analytically, we write (7.6) as the sum of two terms:

$$x_N^2 = \sum_{i=1}^N s_i^2 + \sum_{i \neq j=1}^N s_i s_j. \quad (7.7)$$

The first sum in (7.7) includes terms for which $i = j$; the second sum is over i and j such that $i \neq j$. The product $s_i s_j$ for $i \neq j$ equals $+\ell^2$ and $-\ell^2$ with equal probability, and hence the average of the second term in (7.7) is zero. Because $s_i^2 = \ell^2$ independently of the sign of s_i , the first term in (7.7) equals $\ell^2 N$ for all walks, and hence equals $\ell^2 N$ on the average. We conclude that

$$\langle x_N^2 \rangle = \ell^2 N. \quad (7.8)$$

If the time interval for a step is Δt rather than unity, we should replace N in (7.8) by $N\Delta t$. The result (7.8) can be generalized to two- and three-dimensional walks where each step is a vector \mathbf{s} of constant magnitude, but in a random direction.

The above derivation of the N -dependence of $\langle x_N \rangle$ and $\langle x_N^2 \rangle$ assumes that $p = \frac{1}{2}$. For general p , it is easy to show that $\langle x_N \rangle = (p - q)\ell N$. What is the meaning of this linear dependence on N ? For $p \neq \frac{1}{2}$, it is convenient to consider the dispersion $\langle \Delta x_N^2 \rangle$ defined as

$$\langle \Delta x_N^2 \rangle \equiv \langle (x_N - \langle x_N \rangle)^2 \rangle = \langle x_N^2 \rangle - \langle x_N \rangle^2. \quad (7.9)$$

It is straightforward to show that for any value of p , the N dependence of $\langle \Delta x_N^2 \rangle$ is given by

$$\langle \Delta x_N^2 \rangle = 4pq\ell^2 N. \quad (7.10)$$

What is the N dependence of $\langle x_N^2 \rangle$ for $p \neq q$?

We can gain more insight into the nature of random walks by doing a Monte Carlo simulation, that is, by using a computer to “flip coins” and averaging over many trials. The implementation of the random walk algorithm is simple, for example,

```
if (p < Math.random())
    x++;
else
    x--;
```

The more difficult parts of the program are associated with bookkeeping. The walker takes a total of N steps in each trial and the average values of x_N and x_N^2 are computed.

```
// 3/29/01, 8:30 pm
package edu.clarku.sip.chapter7;
import edu.clarku.sip.graphics.*;
import edu.clarku.sip.templates.*;
import java.awt.*;
import javax.swing.*;
```

```
import java.text.NumberFormat;
// A random walk in 1D
public class OneDimensionalWalk implements AnimationModel, Runnable
{
    private int N;                      // maximum number of steps in one trial
    private double p = 0.5;              // probability of right step
    private int trials;                 // number of trials
    private int steps;                  // steps made by walker
    private int x;                      // position of walker
    private double xcum;
    private double x2cum;
    private double xbar;
    private double x2bar;
    private Control myControl = new SAnimationControl(this);
    private Thread animationThread;
    private Histogram histogram = new Histogram(); // part of graphics package
    private World2D world = new World2D();
    // inner class to display variable trials, xbar, and x2bar
    private VariablesDisplay variablesDisplay = new VariablesDisplay();
    private TickMarks tickMarks = new TickMarks(); // part of graphics package
    private NumberFormat numberFormat;

    public OneDimensionalWalk()
    {
        numberFormat = NumberFormat.getInstance();
        numberFormat.setMaximumFractionDigits(2);
        world.setXOffset(60);           // argument in pixels
        world.setYOffset(60);
        world.addDrawable(histogram);
        world.addDrawable(tickMarks);
        world.addDrawable(variablesDisplay);
    }

    public void startCalculation()
    {
        N = (int) myControl.getValue("N"); // number of steps in one trial
        p = myControl.getValue("p");      // probability of right step
        steps = 0;
        trials = 0;
        x = 0;                         // initial position of walker
        xcum = 0;                       // accumulate values of x after N steps
        x2cum = 0;
        histogram.reset();
        animationThread = new Thread(this);
        animationThread.start();
    }
}
```

```
public void stopCalculation()
{
    animationThread = null;
}

public void continueCalculation()
{
    animationThread = new Thread(this);
    animationThread.start();
}

public void clear()
{
    histogram.reset();
    world.repaint();
}

public void step()
{
    // move walker one step until number of steps equals N
    while (!moveOneStep()) // keep calling method moveOneStep while it returns false
        ; // do nothing, semicolon needed to make while loop valid

    // set range of world in real coordinates
    double xmin = histogram.getXMin();
    double xmax = histogram.getXMax();
    int ymin = histogram.getYMin();
    int ymax = histogram.getYMax();
    world.setXYMinMax(xmin, xmax, ymin, ymax);
    world.render();
}

// move walker one step, return true if walker has completed N steps
private boolean moveOneStep()
{
    if (p < Math.random())
        x++;
    else
        x--;
    steps++;
    if (steps == N)
    {
        trials++;
        histogram.addPoint(x);
        xcum = xcum + x;
    }
}
```

```
x2cum = x2cum + x*x;
xbar = xcum/trials;
x2bar = x2cum/trials;
x = 0;
steps = 0;
return true;
}
return false;
}

public void run()
{
    while (animationThread == Thread.currentThread())
    {
        step();
        try
        {
            Thread.sleep(100);
        }
        catch(InterruptedException e){}
    }
}

public void reset()
{
    myControl.setValue("N", 16);
    myControl.setValue("p", 0.5);
}

private class VariablesDisplay implements Drawable
{
    public void draw(World2D w, Graphics g)
    {
        g.setColor(Color.black);
        int yOffset = (int) w.getYOffset();
        int xOffset = (int) w.getXOffset();
        int width = w.getSize().width;
        g.drawString("trials = " + trials, xOffset, yOffset/2);
        g.drawString("<x> = " + numberFormat.format(xbar), xOffset + width/4,yOffset/2);
        g.drawString("<x^2> = " + numberFormat.format(x2bar), xOffset + width/2,yOffset/2);
    }
}

public static void main(String[] args)
{
    OneDimensionalWalk odw = new OneDimensionalWalk();
```

```

        odw.reset();
    }
}

```

Problem 7.5. Random walks in one dimension

- In class `OneDimensionalWalk` the steps are of unit length so that $\ell = 1$. Use the program to estimate the number of trials needed to obtain $\langle x_N^2 \rangle$ for $N = 10$ steps with an accuracy of approximately 5%. Compare your result to the exact answer (7.8). Approximately how many trials do you need to obtain the same relative accuracy for $N = 40$?
- Is $\langle x_N \rangle$ exactly zero in your simulations? Explain the difference between the analytical result and the results of your simulations.
- How do your results for $\langle x_N \rangle$ and $\langle \Delta x_N^2 \rangle$ change for $p \neq q$? Choose $p = 0.7$ and determine the N dependence of $\langle x_N \rangle$ and $\langle \Delta x_N^2 \rangle$.
- Determine $\langle x_N^2 \rangle$ for $N = 1$ to $N = 5$ by enumerating all the possible walks. For $N = 1$, there are two possible walks: one step to the right and one step to the left. In both cases $x^2 = 1$ and hence $\langle x_1^2 \rangle = 1$ (for $p = \frac{1}{2}$). For $N = 2$ there are four possible walks with the same probability: (i) two steps to the right, (ii) two steps to the left, (iii) first step to the right and second step to the left, and (iv) first step to the left and second step to the right. The value of x_2^2 for these walks is 4, 4, 0, and 0 respectively, and hence $\langle x_2^2 \rangle = (4 + 4 + 0 + 0)/4 = 2$. Write a program that enumerates all the possible walks of a given N and x and compute the various averages exactly.

Class `OneDimensionalWalk` also computes and displays the distribution of values of the displacement x after N steps by using the `Histogram` class listed in Appendix 7A. One way of determining the number of times that the variable x has a certain value is to define a one-dimensional array and let

```
histogram(x) = histogram(x) + 1;
```

In this case because x takes only integer values, the array index of `histogram` is the same as x itself. However, the above statement does not work in Java because x can be negative as well as positive. What we need is a way of mapping values of the bin number to the number of occurrences corresponding to the bin. This map is easy to implement in this case, but Java defines a general `Hashtable` class that maps bin numbers (keys) to occurrences (values). The `Histogram` class also draws itself.

Problem 7.6. Probability distribution

- Compute $P_N(x)$, the probability that the displacement of the walker from the origin is x after N steps. What is the difference between the histogram, that is, the number of occurrences, and the probability? Consider $N = 10$ and $N = 40$ and at least 1000 trials. Does the qualitative form of $P_N(x)$ change as the number of trials increases? What is the approximate width of $P_N(x)$ and the value of $P_N(x)$ at its maximum for each value of N ?

- b. Is $P_N(x)$ a continuous function of x ? Can you fit the envelope of $P_N(x)$ to a continuous function such as

$$A e^{-(x-\langle x_N \rangle)^2/(2\langle \Delta x_N^2 \rangle)}, \quad (7.11)$$

where A is a normalization constant related to $\langle \Delta x_N^2 \rangle$. Compare your computed values for $P_N(x)$ to the form (7.11) using $\langle x_N \rangle$ and $\langle \Delta x_N^2 \rangle$ as input.

- c. Determine $\langle x_N^2 \rangle$ for $N = 1$ to $N = 5$ by enumerating all the possible walks. For $N = 1$, there are two possible walks: one step to the right and one step to the left. In both cases $x^2 = 1$ and hence $\langle x_1^2 \rangle = 1$ (for $p = \frac{1}{2}$). For $N = 2$ there are four possible walks with the same probability: (i) two steps to the right, (ii) two steps to the left, (iii) first step to the right and second step to the left, and (iv) first step to the left and second step to the right. The value of x_2^2 for these walks is 4, 4, 0, and 0 respectively, and hence $\langle x_2^2 \rangle = (4 + 4 + 0 + 0)/4 = 2$. Write a program that enumerates all the possible walks of a given N and x and compute the various averages exactly.

One reason that random walks are very useful in simulating many physical processes and modeling many differential equations of physical interest is that their behavior is closely related to the solutions of the *diffusion* equation. The one-dimensional diffusion equation can be written as

$$\frac{\partial P(x,t)}{\partial t} = D \frac{\partial^2 P(x,t)}{\partial x^2}, \quad (7.12)$$

where D is the self-diffusion coefficient and $P(x,t) dx$ is the probability of a particle being in the interval between x and $x + dx$ at time t . In a typical application $P(x,t)$ might represent the concentration of ink molecules diffusing in a fluid. In three dimensions the second derivative $\partial^2/\partial x^2$ is replaced by the Laplacian ∇^2 .

In Appendix 7B we show that the solution to the diffusion equation with the boundary condition $P(x = \pm\infty, t) = 0$ yields

$$\langle x_N \rangle = 0 \quad (7.13)$$

and

$$\langle x_N^2 \rangle = 2Dt. \quad (7.14)$$

If we compare the form of (7.8) with (7.14), we see that the random walk and the diffusion equation give the same time dependence if we identify t with $N\Delta t$ and $2D$ with $\ell^2/\Delta t$.

The relation of discrete random walks to the diffusion equation implies that we can approach many problems in two ways. The traditional way is to formulate the problem as a partial differential equation as in (7.12) and solve the equation by various numerical methods. One difficulty with this approach is the treatment of complicated boundary conditions. An alternative approach is to formulate the problem as a random walk. In Chapter 12 we will consider random walks in many texts and find that it is straightforward to handle various boundary conditions. Think of other situations that can be treated as random walks (see for example, Section 10.2 and Chapter ??.)

7.3 The Poisson Distribution and Nuclear Decay

As we have seen, we often can change the names of several variables and do a seemingly different physical problem. Our goal in this section is to discuss the decay of unstable nuclei, but we first discuss a conceptually easier problem related to throwing darts. Related physical problems are the distribution of stars in the sky and the distribution of photons on a photographic plate.

Suppose we randomly throw $N = 100$ darts at a board that has been divided into $L = 1000$ equal size regions. The probability that a dart hits a given region in any one throw is $p = 1/1000$. If we count the number of darts in the different regions, we would find that most regions are empty, some regions have one dart, and other regions have more than one dart. What is the probability $P(n)$ that a given region has a particular number of darts?

Problem 7.7. Throwing darts

Write a program that simulates the throwing of N darts at random into L regions in a dart board. Throwing a dart at random at the board is equivalent to choosing an integer at random between 1 and L . Determine $H(n)$, the number of regions with n darts. Note that if $H(n) = H_1(n)$ for the first trial, and $H(n) = H_2(n)$ for the second trial, then $H(n) = H_1(n) + H_2(n)$ for both trials. Obtain $H(n)$ for many trials, and then compute the distribution

$$P(n) = \frac{H(n)}{\sum_{n=0}^N H(n)}. \quad (7.15)$$

As an example, choose $N = 50$ and $L = 1000$. Choose the number of trials to be sufficiently large so that you can determine the qualitative form of $P(n)$.

If N is much greater than unity, then p , the probability that a dart strikes a given region, is much less than unity. The conditions $N \gg 1$ and $p \ll 1$ and the independence of the events (the landing of a dart in a particular region) satisfy the requirements for a *Poisson distribution*. The Poisson distribution, $P(n)$, is given by

$$P(n) = \frac{\langle n \rangle^n}{n!} e^{-\langle n \rangle}, \quad (7.16)$$

where n is the number of darts in a given region and $\langle n \rangle$ is the mean number, $\langle n \rangle = \sum_{n=0}^{\infty} n P(n)$. The upper limit of the sum should be N , but since $N \gg 1$, we can take the upper limit to be ∞ when it is convenient.

Problem 7.8. Darts and the Poisson distribution

- Write a program to compute $\sum_{n=0}^N P(n)$, $\sum_{n=0}^N n P(n)$, and $\sum_{n=0}^N n^2 P(n)$ using the form (7.16) for $P(n)$. Choose a reasonable value for $\langle n \rangle$. Verify that $P(n)$ in (7.16) is normalized. What is the value of σ for the Poisson distribution?
- Modify the program that you developed for Problem 7.7 to compute $\langle n \rangle$. Choose $N = 50$ and $L = 1000$ and use your measured value of $\langle n \rangle$ as input to compare the Poisson distribution (7.16) to your computed values of $P(n)$. If time permits, use larger values of N and L .
- Choose $L = 100$ and $N = 50$ and redo part (b). Are your results consistent with a Poisson distribution? What happens if $L = N = 50$?

Now that we are more familiar with the Poisson distribution, we consider the decay of radioactive nuclei. We know that a collection of radioactive nuclei will decay into other nuclei, and that there is no way to know a priori which nucleus will decay next. If all nuclei of a particular type are identical, why do they not all decay at the same time? The answer is based on the fundamental uncertainty inherent in the quantum description of matter at the microscopic level. In the following, we will see that a simple model of the decay process leads to an exponential decay law. This approach complements the continuum approach discussed in Section 2.8.

Because each nucleus is identical, we assume that during any time interval Δt , each nucleus has the same probability p of decaying. The basic algorithm is simple — choose an unstable nucleus and generate a random number r uniformly distributed in the unit interval $0 \leq r < 1$. If $r \leq p$, the unstable nucleus decays; otherwise, it does not. Every unstable nucleus is tested during each time interval. Note that for a system of unstable nuclei, there are many events that can happen during each time interval, for example, $0, 1, 2, \dots, n$ nuclei can decay. In contrast, for the particles in the box problem, there is a probability of unity that a particle is moved from one side to the other side at each time interval. Remember that once a nucleus decays, it is no longer in the group of unstable nuclei that is tested at each time interval. Program `nuclear_decay`, listed below, implements the nuclear decay algorithm.

```

PROGRAM nuclear_decay
! simulation of decay of unstable nuclei
DIM ncum(0 to 1000)
CALL initial(N0,p,ntrial,tmax,ncum())
CALL decay(N0,p,ntrial,tmax,ncum())
CALL output(ntrial,tmax,ncum())
END

SUB initial(N0,p,ntrial,tmax,ncum())
  RANDOMIZE
  INPUT prompt "initial number of unstable nuclei = ": N0
  INPUT prompt "decay probability for unstable nucleus = ": p
  INPUT prompt "number of time intervals per trial = ": tmax
  INPUT prompt "number of trials = ": ntrial
  FOR t = 1 to tmax
    LET ncum(t) = 0      ! accumulate number of unstable nuclei
  NEXT t
END SUB

SUB decay(N0,p,ntrial,tmax,ncum())
  DIM N(5000)
  FOR itrial = 1 to ntrial
    FOR i = 1 to N0
      LET N(i) = 1      ! nucleus = 1 if unstable
    NEXT i
    LET ncum(0) = ncum(0) + N0
    LET n_unstable = N0 ! # of unstable nuclei
    FOR t = 1 to tmax

```

```

FOR i = 1 to NO
    IF N(i) = 1 then
        IF rnd <= p then
            LET N(i) = 0      ! nucleus decays
            LET n_unstable = n_unstable - 1
        END IF
    END IF
NEXT i
LET ncum(t) = ncum(t) + n_unstable ! accumulate data
NEXT t
NEXT itrial
END SUB

SUB output(ntrial,tmax,ncum())
    ! print data to file to be read by separate program
    OPEN #1: name "decay.dat", create new, access output
    PRINT #1: "time", "mean number of unstable nuclei"
    FOR t = 0 to tmax
        PRINT #1: t, ncum(t)/ntrial
    NEXT t
    CLOSE #1
END SUB

```

Problem 7.9. Monte Carlo simulation of nuclear decay

- Program `nuclear_decay` does the simulation `ntrial` times and averages the results. Assume that the time interval is one second. Choose `NO` = 100 (the initial number of unstable nuclei), `p` = 0.01, `tmax` = 100, and `ntrial` = 20. Is your result for $N(t)$, the mean number of unstable nuclei at time t , consistent with the expected behavior, $N(t) = N(0) e^{-\lambda t}$ found in Section 2.8? What is the value of λ for this value of p ?
- There are a very large number of unstable nuclei in a typical radioactive source. We also know that over any reasonable time interval, only a relatively small number decay. Because $N \gg 1$ and $p \ll 1$, we expect that $P(n)$, the probability that n nuclei decay during a specified time interval, is a Poisson distribution. Modify Program `nuclear_decay` so that it outputs the probability that n unstable nuclei decay during the first time interval. Choose `NO` = 1000, `p` = 0.001, `tmax` = 1, and `ntrial` = 1000. What is the mean number $\langle n \rangle$ of nuclei that decay during this interval? What is the associated variance? Plot $P(n)$ versus n and compare your results to the Poisson distribution (7.16) with your measured value of $\langle n \rangle$ as input. Then consider p = 0.02.
- Modify Program `nuclear_decay` so that it outputs the probability that n unstable nuclei decay during two time intervals. Choose `NO` = 1000, `p` = 0.001, `tmax` = 2, and `ntrial` = 1000. Compare the probability you obtain with your results from part (b). How do your results change as the time interval becomes larger?
- Increase p for fixed N = 1000 and determine $P(n)$ for a given time interval. Estimate the values of p and n for which the Poisson distribution is no longer applicable.

- e. Modify your program so that it flashes a small circle on the screen or makes a sound (like that of a Geiger counter) when a nucleus decays. Choose the location of the small circle at random. Do a single run and describe the qualitative differences between the visual and/or audio patterns for the situations in parts (a)–(d)? Choose $N \geq 5000$. Such a visualization might be somewhat misleading on a serial computer because only one nuclei can be considered at a time. In contrast, for a real system, nuclei can decay simultaneously. How can you improve the visualization to better approximate a real system?

7.4 Problems in Probability

Because most of the questions in this introductory chapter on random processes can be answered by analytical methods, why bother to simulate these processes? One reason is that it is simpler to introduce new methods in a familiar context. Another reason is that if we change the nature of the random processes slightly, it often happens that it is difficult or impossible to obtain the answers by familiar methods. Still another reason is that writing a program and doing a simulation can aid your intuitive understanding of the subtle concept of probability. Probability is an elusive concept in part because it cannot be measured at one time. To reinforce the importance of thinking about how to solve a problem on a computer, we suggest some problems in probability in the following. Does thinking about these problems in this way help lead you to a pencil and paper solution?

Problem 7.10. The three boxes: stick or switch? Suppose that there are three identical boxes, each with a lid. When you leave the room, a friend places a \$10 bill in one of the boxes and closes the lid of each box. The friend knows the location of the \$10 bill, but you do not. You then reenter the room and guess which one of the boxes has the \$10 bill. As soon as you do, your friend opens the lid of another box that is empty. So if you have chosen an empty box, your friend will open the lid of the other empty box. If you have chosen the right box, your friend will open at random the lid of one of the two empty boxes. You now have the opportunity to stay with your original choice, or to switch to the other unopened box. Suppose that you play this game many times and that each time you guess correctly, you keep the money. To maximize your winnings, should you maintain your initial choice or should you switch? Which strategy is better? Write a program to simulate this game and output the probability of winning for switching and for not switching. It is likely that before you finish your program, the correct strategy will become clear. To make your program more useful, consider four or five boxes.

Problem 7.11. Conditional probability

Suppose that many people in a community are tested at random for HIV. The accuracy of the test is 87% and the incidence of the disease in the general population, independent of any test, is 1%. If a person tests positive for HIV, what is the probability that this person really has HIV? Write a program to compute the probability. The answer can be found by using Bayes' theorem (cf. Bernardo and Smith). The answer is much less than 87%.

Problem 7.12. The roll of the dice

- a. Write a program to compute the probability of obtaining at least one double six in twenty-four throws of a pair of die.

Team	Won	Lost	Percentage
Oakland	99	63	0.611
Kansas City	92	70	0.568
California	91	71	0.562
Texas	83	79	0.512
Minnesota	80	82	0.494
Seattle	73	89	0.451
Chicago	69	92	0.429

Table 7.1: The American League West standings for 1989.

- b. A player rolls two dice. If the sum of the two dice is 7 or 11, the player wins immediately. If the sum is 2, 3, or 12, the player loses immediately. If the game is neither won nor lost on the first throw, the initial number is either 4, 5, 6, 8, 9, or 10. The player rolls the dice again until she either wins by repeating her initial number or she loses by rolling a 7. Write a program to determine the probability that the player wins this game (a variation of the game of craps).
- c. Suppose that two gamblers each begin with \$100 in capital and on each throw of a coin, one gambler must win \$1 and the other must lose \$1. How long can they play on the average until the capital of the loser is exhausted? How long can they play if they each begin with \$1000? Neither gambler is allowed to go into debt.

Problem 7.13. The Boys of Summer

Luck plays a large role in the outcome of any baseball season. The American League West standings for 1989 are given in Table 7.1. Suppose that the teams remain unchanged and their probability of winning a particular game was the same as in 1989. Do a simulation to determine the probability that Oakland would lead the division for another season. For simplicity, assume that the teams play only each other.

Much of the present day motivation for the development of probability comes from science rather than from gambling. The next problem has much to do with statistical physics even though this application is not apparent.

Problem 7.14. Money exchange

Consider a two-dimensional plane that has been subdivided into cells, for example, a checkerboard. There can be an indefinite number of coins stacked on each cell. For simplicity, we initially assign one coin to each cell. The game proceeds as follows. Select two cells at random. If there is at least one coin on the first cell, move one coin to the second cell. If the first cell is empty, then do nothing. After many coin exchanges, what does a typical state look like? Are the coins uniformly distributed as in the initial state or are many cells empty? Does the system approach equilibrium? Write a Monte Carlo program to simulate this game and show the state of the cells visually. Consider a system with at least 16×16 cells. Plot the histogram $H(n)$ versus n , where $H(n)$ is the number of cells with n coins. Do your results change if you consider bigger systems or begin with more coins on each cell?

Problem 7.15. Distribution of cooking times

An industrious physics student finds a job at a local fast food restaurant to help him pay his way through college. His task is to cook 20 hamburgers on a grill at any one time. When a hamburger is cooked, he is supposed to replace it with an uncooked hamburger. However, our physics major does not pay attention to whether the hamburger is cooked or not. His method is to choose a hamburger at random and replace it by a uncooked one and not bother to check whether the hamburger that he removes from the grill is cooked or not. What is the distribution of cooking times of the hamburgers that he removes? To simplify the problem, assume that he replaces a random hamburger at regular intervals of one minute and that there is an indefinite supply of uncooked hamburgers. Does the qualitative nature of the distribution change if he cooks 40 hamburgers at any one time?

7.5 Method of Least Squares

There is a long way to go from obtaining data to determining the relation that best describes it. For example, in Problem 7.5 we computed the mean square displacement $\langle x_N^2 \rangle$ as a function of N for a simple random walk. In Problem 7.9 we did a simulation of $N(t)$, the number of unstable nuclei at time t . Given the finite accuracy of our data, how do we know if our simulation results are consistent with the exponential relation between N and t ? Are our simulation results consistent with the theoretical prediction that $\langle x_N^2 \rangle$ is proportional to N for $p = \frac{1}{2}$?

The approach that we have been using is to plot the computed values of $\langle x_N^2 \rangle$ as a function of N and to rely on our eye to help us draw the curve that best fits the data points. Of course, this graphical approach works best when the curve is a straight line, that is, when the relation is linear. The advantages of this approach are that it is straightforward and allows us to see what we are doing. For example, if a data point is far from the curve, or if there is a gap in the data, we will notice it easily. If the true analytical relation is not linear, it is likely that we will notice that the data points do not fit a simple straight line, but instead show curvature. If we blindly let a computer fit the data to a straight line, we might not notice that the fit is not very good unless we already have had much experience fitting data by hand. Finally, the visceral experience of using a transparent ruler and fitting the data gives us some feeling for the nature of the data that might otherwise be missed. It usually is a good idea to plot some data in this way even though a computer can do it much faster.

Although the graphical approach is simple, it does not yield precise fits and we need to use analytical methods also. The most common method for finding the best straight line fit to a series of measured points is called *linear regression* or *least squares*. Suppose we have n pairs of measurements $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ and that the errors are entirely in the values of y . For convenience, we also assume that the uncertainties in y all have the same magnitude. Our goal is to obtain the best fit to the linear function

$$y = mx + b. \quad (7.17)$$

The problem is to calculate the values of the parameters m and b for the best straight line through the n data points. The difference

$$d_i = y_i - mx_i - b \quad (7.18)$$

is a measure of the discrepancy in y_i . It is reasonable to assume that the best set of values of m and b are those that minimize the quantity

$$S = \sum_{i=1}^n (y_i - mx_i - b)^2. \quad (7.19)$$

Why should we minimize the sum of the squared differences between the experimental values, y_i , and the analytical values, $mx_i + b$, and not some other function of the differences? The justification is based on the assumption that if we did many simulations, then the values of d_i would be distributed according to the Gaussian distribution (see Problems 7.5 and 12.8). Based on this assumption, it can be shown that the values of m and b that minimize S yield a set of values of $mx_i + b$ that are the *most probable* set of measurements that we would find based on the available information.

To minimize S , we take the derivative of S with respect to b and m :

$$\frac{\partial S}{\partial m} = -2 \sum_{i=1}^n x_i(y_i - mx_i - b) = 0, \quad (7.20a)$$

$$\frac{\partial S}{\partial b} = -2 \sum_{i=1}^n (y_i - mx_i - b) = 0. \quad (7.20b)$$

From (7.20) we obtain two simultaneous equations:

$$m \sum_{i=1}^n x_i^2 + b \sum_{i=1}^n x_i = \sum_{i=1}^n x_i y_i \quad (7.21a)$$

$$m \sum_{i=1}^n x_i + bn = \sum_{i=1}^n y_i. \quad (7.21b)$$

It is convenient to define the average quantities

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (7.22a)$$

$$\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i \quad (7.22b)$$

$$\bar{xy} = \frac{1}{n} \sum_{i=1}^n x_i y_i, \quad (7.22c)$$

and rewrite (7.21a) and (7.21b) as

$$m\bar{x}^2 + b\bar{x} = \bar{xy}, \quad (7.23a)$$

$$m\bar{x} + b = \bar{y}. \quad (7.23b)$$

N	$\langle x_N^2 \rangle$
8	19.43
16	37.65
32	76.98
64	160.38

Table 7.2: Computed values of the mean square displacement $\langle x_N^2 \rangle$ as a function of the total number of steps N . The results $\langle x_N^2 \rangle$ are averaged over 1000 trials. The one-dimensional random walker takes steps of length 1 or 2 with equal probability, and the direction of the step is random with $p = \frac{1}{2}$.

The solution of (7.23a) and (7.23b) can be expressed as

$$m = \frac{\bar{xy} - \bar{x}\bar{y}}{(\Delta x)^2} \quad (7.24a)$$

$$b = \bar{y} - m\bar{x}. \quad (7.24b)$$

where

$$(\Delta x)^2 = \bar{x^2} - \bar{x}^2 \quad (7.24c)$$

Equations (7.24a)–(7.24c) determine the slope m and the intercept b of the best straight line through the n data points.

As an example, consider the data shown in Table 7.2 for a one-dimensional random walk. To make the example more interesting, suppose that the walker takes steps of length 1 or 2 with equal probability. The direction of the step is random and $p = \frac{1}{2}$. As in Section 7.2, we assume that the mean square displacement $\langle x_N^2 \rangle$ obeys the general relation

$$\langle x_N^2 \rangle = aN^{2\nu} \quad (7.25)$$

with an unknown exponent ν . First we convert the nonlinear relation (7.25) to a linear relation by taking the logarithm of both sides:

$$\ln\langle x_N^2 \rangle = \ln a + 2\nu \ln N. \quad (7.26)$$

If we take the logarithm of the data in Table 7.2 and use (7.24), we find that $m = 1.02$ and $b = 0.83$. Hence, we conclude from our limited data and the relation $2\nu = m$ that $\nu \approx 0.51$, a numerical result that is consistent with the expected result $\nu = 1/2$. The values of $y = \ln\langle x_N^2 \rangle$ and $x = \ln N$ and the least squares fit are shown in Figure 7.2. Note that the original fitting problem is nonlinear, that is, $\langle x_N^2 \rangle$ depends on N^ν rather than N . Often a problem that looks nonlinear can be turned into a linear problem by a change of variables.

The least squares fitting procedure also allows us to estimate the uncertainty or the most probable error in m and b by analyzing the measurements themselves. The result of this analysis

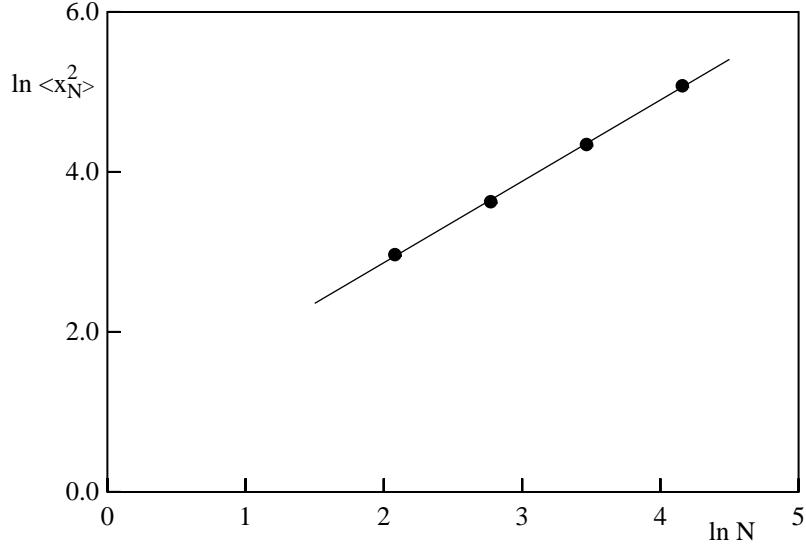


Figure 7.2: Plot of $\ln\langle x_N^2 \rangle$ versus $\ln N$ for the data listed in Table 7.2. The straight line $y = 1.02x + 0.83$ through the points is found by minimizing the sum (7.19).

is that the most probable error in m and b , σ_m and σ_b respectively, is given by

$$\sigma_m = \frac{1}{\sqrt{n}} \frac{\sigma_y}{\Delta x} \quad (7.27a)$$

$$\sigma_b = \frac{1}{\sqrt{n}} \frac{(\bar{x}^2)^{1/2}}{\Delta x} \sigma_y, \quad (7.27b)$$

where

$$\sigma_y^2 = \frac{1}{n-2} \sum_{i=1}^n d_i^2, \quad (7.27c)$$

and d_i is given by (7.18). Because there are n data points, we might have guessed that n rather than $n - 2$ would be present in the denominator of (7.27c). The reason for the factor of $n - 2$ is related to the fact that to determine σ_y , we first need to calculate *two* quantities m and b , leaving only $n - 2$ independent degrees of freedom. To see that the $n - 2$ factor is reasonable, consider the special case of $n = 2$. In this case we can find a line that passes exactly through the two data points, but we cannot deduce anything about the reliability of the set of measurements because the fit always is exact. If we use (7.27c), we see that both the numerator and denominator would be zero, and hence $\sigma_y = 0/0$, that is, σ_y is undetermined. If a factor of n appeared in (7.27c) instead, we would conclude that $\sigma_y = 0/2 = 0$, an absurd conclusion. Usually $n \gg 1$, and the difference between n and $n - 2$ is negligible.

For our example, $\sigma_y = 0.03$, $\sigma_b = 0.07$, and $\sigma_m = 0.02$. The uncertainties δm and δb are related by $2\delta b = \delta m$. Because we can associate δm with σ_m , we conclude that our best estimate

for ν is $\nu = 0.51 \pm 0.01$.

If the values of y_i have different uncertainties σ_i , then the data points are weighted by the quantity $w_i = 1/\sigma_i^2$. In this case it is reasonable to minimize the quantity

$$\chi^2 = \sum_{i=1}^n w_i(y_i - mx_i - b)^2. \quad (7.28)$$

The resulting expressions (7.24a) and (7.24b) for m and b are unchanged if we generalize the definition of the averages to be

$$\bar{f} = \frac{1}{n\bar{w}} \sum_{i=1}^n w_i f_i, \quad (7.29)$$

where

$$\bar{w} = \frac{1}{n} \sum_{i=1}^n w_i \quad (7.30)$$

In Chapter 11 we discuss how to estimate the most probable errors in $\langle x_N^2 \rangle$.

Problem 7.16. Example of least squares fit

- Write a program to find the least squares fit for a set of data. As a check on your program, compute the most probable values of m and b for the data shown in Table 7.2.
- Modify the random walk program so that steps of length 1 and 2 are taken with equal probability. Use at least 10 000 trials and do a least squares fit to $\langle x_N^2 \rangle$ as done in the text. Is your most probable estimate for ν closer to $\nu = 1/2$?

For the simple random walk problems considered in this chapter, the relation $\langle x_N^2 \rangle = aN^\nu$ holds for all N . However, in many random walk problems (see Chapter 12), a power law relation between $\langle x_N^2 \rangle$ and N holds only asymptotically for large N , and hence we should use only the larger values of N to estimate the slope. We also need to give equal weight to all intervals of the independent variable N . In the above example, we used $N = 8, 16, 32$, and 64 , so that the values of $\ln N$ are equally spaced.

7.6 A Simple Variational Monte Carlo Method

Many problems in physics can be formulated in terms of a variational principle. In the following, we consider examples of variational principles from geometrical optics and classical mechanics. We then discuss how Monte Carlo methods can be applied to obtain estimates for the maximum or minimum. A more sophisticated application of Monte Carlo methods to a variational problem in quantum mechanics is discussed in Chapter ???. Our everyday experience of light leads naturally to the concept of light rays. This description of light propagation, called *geometrical* or *ray optics*, is applicable when the wavelength of light is small compared to the linear dimensions of any obstacles or openings. The propagation of light rays can be formulated in terms of a principle due to Fermat:

A ray of light follows the path between two points (consistent with any constraints) that requires the least amount of time.

Fermat's principle of least time can be adopted as the basis of geometrical optics. For example, Fermat's principle implies that light travels from a point A to a point B in a straight line in a homogeneous medium. Because the speed of light is constant along any path within the medium, the path of shortest time is the path of shortest distance, that is, a straight line from A to B . What happens if we impose the constraint that the light must strike a mirror before reaching B ?

The speed of light in a medium can be expressed in terms of c , the speed of light in a vacuum, and the index of refraction n of the medium:

$$v = \frac{c}{n}. \quad (7.31)$$

Suppose that a light ray in a medium with index of refraction n_1 passes through a second medium with index of refraction n_2 . The two media are separated by a plane surface. We now show how we can use Fermat's principle and a Monte Carlo method to find the path of the light. The analytical solution to this problem using Fermat's principle is found in many texts (cf. Feynman et al.).

Our strategy, as implemented in **Program fermat**, is to begin with an arbitrary path and to make changes in the path at random. These changes are accepted only if they reduce the travel time of the light. Some of the features of **Program fermat** include:

- a. Light propagates from left to right through N media.
- b. The width of each region is unity and the index of refraction is uniform in each region. The index i increases from left to right. There are $N - 1$ boundaries separating the N media with index of refraction $n(i)$ and speed $v(i)$. We have chosen units such that the speed of light in a vacuum equals unity.
- c. Because the light propagates in a straight line in each medium, the path of the light is given by the coordinates $y(i)$ at each boundary.
- d. The coordinates of the light source and the detector are at $(1, y(1))$ and $(N, y(N))$ respectively, where $y(1)$ and $y(N)$ are fixed.
- e. The initial path is the connection of the set of random points at the boundary of each region.
- f. The path of the light is found by choosing the boundary i at random and generating a trial value of $y(i)$ that differs from its previous value by a random number between $-\delta$ to δ . If the trial value of $y(i)$ yields a shorter travel time, this value becomes the new value for $y(i)$.
- g. The path is redrawn whenever it is changed.

```
PROGRAM fermat
! Monte Carlo method for finding minimum optical path
DIM y(101),v(101)
CALL initial(y(),v(),N)
CALL change_path(y(),v(),N)
END
```

```

SUB initial(y(),v(),N)
  RANDOMIZE
  LET N = 10           ! number of different media (even)
  ! speed of light in vacuum equal to unity
  LET v1 = 1.0
  LET index = 1.5      ! index of refraction of medium #2
  LET v2 = 1/index
  FOR i = 1 to N/2
    LET v(i) = v1      ! speed of light in left half
  NEXT i
  FOR i = N/2 + 1 to N
    LET v(i) = v2      ! speed of light in right half
  NEXT i
  LET y(1) = 2          ! fixed source
  LET y(N) = 8          ! fixed detector
  SET WINDOW 0.5,N+1,y(1)-0.5,y(N)+0.5
  BOX LINES 1,N,y(1),y(N)
  PLOT LINES: N/2,y(1);N/2,y(N)      ! boundary
  ! choose initial path at boundary between endpoints
  FOR i = 2 to N-1
    LET y(i) = (y(N) - y(1))*rnd + y(1)
  NEXT i
  SET COLOR "red"
  FOR i = 1 to N-1
    PLOT LINES: i,y(i);i+1,y(i+1)  ! initial path
  NEXT i
  SET COLOR "red/white"
END SUB

SUB change_path(y(),v(),N)
  LET delta = 0.5        ! maximum change in y
  DO
    ! choose random x not including x = 1 or N
    LET x = int((N-2)*rnd) + 2
    LET ytrial = y(x) + (2*rnd - 1)*delta    ! new y position
    LET dy2 = (y(x) - y(x+1))^2      ! vertical distance squared
    LET dist = sqr(1 + dy2)      ! horizontal distance is unity
    LET t_original = dist/v(x+1)
    LET dy2 = (y(x) - y(x-1))^2
    LET dist = sqr(1 + dy2)
    LET t_original = t_original + dist/v(x)
    LET dy2 = (ytrial - y(x+1))^2
    LET dist = sqr(1 + dy2)
    LET t_trial = dist/v(x+1)
    LET dy2 = (ytrial - y(x-1))^2
  
```

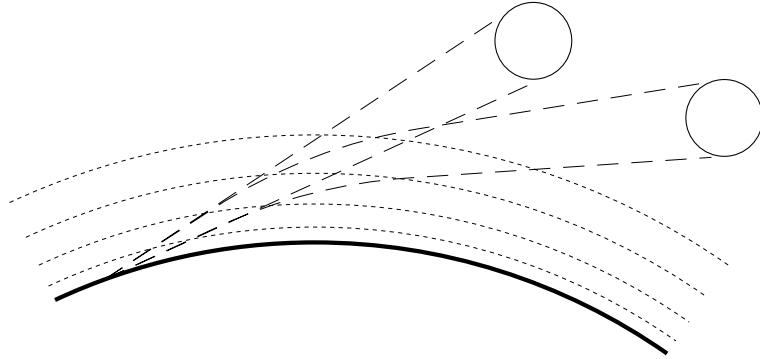


Figure 7.3: Near the horizon, the apparent (exaggerated) position of the sun is higher than the true position of the sun. Note that the light rays from the true sun are curved due to refraction.

```

LET dist = sqr(1 + dy2)
LET t_trial = t_trial + dist/v(x)
IF t_trial < t_original then      ! new position reduces time
    SET COLOR "white"
    PLOT LINES: x-1,y(x-1);x,y(x);x+1,y(x+1)
    SET COLOR "red"
    LET y(x) = ytrial
    PLOT LINES: x-1,y(x-1);x,y(x);x+1,y(x+1)
END IF
LOOP until key input
END SUB

```

Problem 7.17. The law of refraction

- Use **Program fermat** to determine the angle of incidence θ_1 and the angle of refraction θ_2 between two media with different indices of refraction. The angles θ_1 and θ_2 are measured from the normal to the boundary. Set $N = 10$ and let the first medium be air ($n_1 \approx 1$) and the second medium be glass ($n_2 \approx 1.5$). Describe the path of the light after a number of trial paths are attempted. Add some statements to the program to determine θ_1 and θ_2 , the vertical position of the intersection of the light at the boundary between the two media, and the total time for the light to go from $(1, y(1))$ and $(10, y(10))$.
- Modify the program so that the first medium represents glass ($n_1 \approx 1.5$) and the second medium represents water ($n_2 \approx 1.33$). Verify that your results in (a) and (b) are consistent with Snell's law, $n_2 \sin \theta_2 = n_1 \sin \theta_1$.

Problem 7.18. Inhomogeneous media

- The earth's atmosphere is thin at the top and dense near the earth's surface. We can model this inhomogeneous medium by dividing the atmosphere into equal width segments each of which

is homogeneous. Take the index of refraction of region i to be $n(i) = 1 + (i - 1)*dn$. Run **Program fermat** with $N = 10$ and $dn = 0.1$, and find the path of least time. Use your results to explain why when we see the sun set, the sun already is below the horizon (see Figure 7.3).

- * Use **Program fermat** to find the appropriate distribution of $n(i)$ for a fiber optic cable. In this case the i th region corresponds to a cross sectional slab through the cable. Although a real cable is three-dimensional, we consider a two-dimensional cable for simplicity. Imagine the cable to be a flat, long ribbon of width equal to N . The middle region is the center of the cable and the $i = 1$ and $i = N$ regions are at the edge of the cable. We want the cable to have the property that if a ray of light starts from one side of the cable and ends at the other, the slope dy/dx of the path should be near zero at the edges so that light does not escape from the cable.

Fermat's principle is an example of an extremum (maxima or minima) principle. An extremum means that a small change ϵ in an independent variable leads to a change in a function (more precisely, a function of functions) that is proportional to ϵ^2 or a higher power of ϵ . An important extremum principle in classical mechanics is based on the action S :

$$S = \int_{t_{\text{initial}}}^{t_{\text{final}}} L dt \quad (7.32)$$

The Lagrangian L in (7.32) is the kinetic energy minus the potential energy. The extremum principle for the action is known as *the principle of least action*. If we were to take the extremum of (7.32), we would find that the path for which S is an extremum satisfies the differential equation of motion equivalent to Newton's second law (for conservative forces). One reason for the importance of the principle of least action is that quantum mechanics can be formulated in terms of an integral over the action. This way of doing quantum mechanics is called the path integral formulation (see Section ??).

Our main motivation here is to gain more experience with extremum problems. To use (7.32) to find the motion of a single particle in one dimension, we fix the position at the initial and final times, $x(t_{\text{initial}})$ and $x(t_{\text{final}})$, and then choose the velocities and positions for other times $t_{\text{initial}} < t < t_{\text{final}}$ so as to minimize the action. One way to implement this procedure numerically is to convert the integral in (7.32) to a sum:

$$S \approx \sum_{i=1}^{N-1} L(t_i) \Delta t, \quad (7.33)$$

where $t_i = t_{\text{initial}} + i\Delta t$. (The approximation used to obtain (7.33) is known as the rectangular approximation and is discussed in Chapter 11.) For a single particle in one dimension, we can write

$$L_i \approx \frac{m}{2(\Delta t)^2} (x_{i+1} - x_i)^2 - u(x_i), \quad (7.34)$$

where m is the mass of the particle and $u(x_i)$ is the potential energy of the particle at x_i . The velocity has been approximated as the difference in position divided by the change in time Δt .

**Problem 7.19.* Principle of least action

- a. Write a program to minimize the action S given in (7.32) for the motion of a single particle in one dimension. Use the approximate form of the Lagrangian given in (7.34). One way to write the program is to modify *Program fermat* so that the vertical coordinate for the light ray becomes the position of the particle, and the horizontal region number i of width Δx becomes the discrete time interval number of duration Δt . The quantity to be minimized is different, but otherwise the algorithm is similar. It is possible to extend the principle of least action to more dimensions or particles, but it is necessary to begin with a path close to the optimum one to obtain a good approximation to the optimum path in a reasonable time.
- b. Verify your program for the case of free fall for which the potential energy is $u(x) = mgx$. Choose $x(t = 0) = 2\text{ m}$ and $x(t = 10\text{ s}) = 8\text{ m}$, and begin with $N = 20$. Allow the maximum change in the position to be 5 m. Make sure that the window coordinates are sufficiently large to see the paths. What do you expect the shape of the plot of x versus t to be?
- c. Consider the harmonic potential $u(x) = \frac{1}{2}kx^2$. What shape do you expect the path $x(t)$ to be? Increase N to approximately 50.

Appendix 7A: Listing of Histogram Class

```

package edu.clarku.sip.graphics;
import java.awt.*;
import java.util.*;

public class Histogram implements Drawable
{
    private Hashtable bins = new Hashtable(); // Hashtable maps bin number to occurrences
    private double binWidth = 1; // 0 < 1, 1 < 2, 2 < 3, etc.
    private boolean discrete = true; // false if the bins are continuous
    private double xmin, xmax;
    private int ymin, ymax;
    private Color binColor = Color.red;

    public Histogram()
    {
        resetXYMinMax();
    }

    public void setDiscrete(boolean b)
    {
        discrete = b;
    }

    public void setBinColor(Color c){binColor = c;}

    private void resetXYMinMax()

```

```

{
    xmin = Integer.MAX_VALUE;
    xmax = Integer.MIN_VALUE;
    ymax = 10;
    ymin = 0;
}

public void setBinWidth(double d)
{
    binWidth = d;
}

public double getXMin(){return xmin;}
public double getXMax(){return xmax;}
public int getYMin(){return ymin;}
public int getYMax(){return ymax;}

public void addPoint(double value)
{
    int binNumber = (int)(value/binWidth);
    // determine if there has been any occurrences for this bin
    Integer occurences = (Integer) bins.get(new Integer(binNumber));
    int count = 1;
    if (occurences == null)
    {
        bins.put(new Integer(binNumber), new Integer(count)); // binNumber has one occurrence
    }
    else
    {
        // need to put objects in hashtable, but can only add ints
        count = count + occurences.intValue(); // increase occurrences by one
        bins.put(new Integer(binNumber), new Integer(count));
    }
    ymax = Math.max(count, ymax);
    xmin = Math.min(binNumber - binWidth, xmin);
    xmax = Math.max(binNumber + binWidth, xmax);
}

public void draw(World2D p, Graphics g)
{
    if (bins.size() == 0)
        return;

    drawHistogram(p, g);
}

```

```

public void drawHistogram(World2D p, Graphics g)
{
    Hashtable temp = (Hashtable) bins.clone();
    Enumeration keys = temp.keys();
    g.setColor(Color.black);
    int xoffset = (int) p.getXOffset();
    int yoffset = (int) p.getYOffset();
    // draw an enclosing box
    g.drawRect(xoffset, yoffset, p.getSize().width - xoffset*2, p.getSize().height - 2*yoffset);
    g.setColor(binColor);
    while (keys.hasMoreElements())
    {
        Integer binNumber = (Integer) keys.nextElement();
        Integer occurrences = (Integer) bins.get(binNumber);
        drawBin(p, g, binNumber.intValue(), occurrences.intValue());
    }
}

public void drawBin(World2D p, Graphics g, int binNumber, int occurrences)
{
    int px = p.xToPix(binNumber*binWidth);
    if (discrete)
    {
        g.drawLine(px, p.yToPix(0), px, p.yToPix(occurrences));
    }
    else // continuous distribution
    {
        int py = p.yToPix(occurrences);
        int px2 = p.xToPix(binNumber*binWidth +binWidth);
        int pWidth = px2 - px;
        int pHeight = Math.abs(py - p.yToPix(0));
        g.fillRect(px, py, pWidth, pHeight);
    }
}

public void reset()
{
    bins = new Hashtable();
    resetXYMinMax();
}

public static void main(String[] args)
{
    Histogram h = new Histogram();
    h.setDiscrete(false);
    h.addPoint(1);
}

```

```

    h.addPoint(1);
    h.addPoint(3);
    h.addPoint(-3);
    World2D w = new World2D();
    w.setXYMinMax(-4,4,0,3);
    w.setXOffset(60);
    w.setYOffset(60);
    w.addDrawable(h);
    w.addDrawable(new TickMarks());
    w.repaint();
}
}

```

Appendix 7B: Random Walks and the Diffusion Equation

To gain some insight into the relation between random walks and the diffusion equation, we show that the latter implies that $\langle x(t) \rangle$ is zero and $\langle x^2(t) \rangle$ is proportional to t . We rewrite the diffusion equation (7.12) here for convenience:

$$\frac{\partial P(x,t)}{\partial t} = D \frac{\partial^2 P(x,t)}{\partial x^2}, \quad (7.35)$$

To derive the t dependence of $\langle x(t) \rangle$ and $\langle x^2(t) \rangle$ from (7.35), we write the average of any function of x as

$$\langle f(x,t) \rangle = \int_{-\infty}^{\infty} f(x) P(x,t) dx. \quad (7.36)$$

The average displacement is given by

$$\langle x(t) \rangle = \int_{-\infty}^{\infty} x P(x,t) dx. \quad (7.37)$$

To do the integral on the right hand side of (7.37), we multiply both sides of (7.35) by x and formally integrate over x :

$$\int_{-\infty}^{\infty} x \frac{\partial P(x,t)}{\partial t} dx = D \int_{-\infty}^{\infty} x \frac{\partial^2 P(x,t)}{\partial x^2} dx. \quad (7.38)$$

The left-hand side can be expressed as:

$$\int_{-\infty}^{\infty} x \frac{\partial P(x,t)}{\partial t} dx = \frac{\partial}{\partial t} \int_{-\infty}^{\infty} x P(x,t) dx = \frac{d}{dt} \langle x \rangle. \quad (7.39)$$

The right-hand side of (7.38) can be written in the desired form by doing an integration by parts:

$$D \int_{-\infty}^{\infty} x \frac{\partial^2 P(x,t)}{\partial x^2} dx = D x \frac{\partial P(x,t)}{\partial x} \Big|_{x=-\infty}^{x=\infty} - D \int_{-\infty}^{\infty} \frac{\partial P(x,t)}{\partial x} dx. \quad (7.40)$$

The first term on the right hand side of (7.40) is zero because $P(x = \pm\infty, t) = 0$ and all the spatial derivatives of P at $x = \pm\infty$ are zero. The second term also is zero because it integrates to $D[P(x = \infty, t) - P(x = -\infty, t)]$. Hence, we find that

$$\frac{d}{dt}\langle x \rangle = 0, \quad (7.41)$$

or $\langle x \rangle$ is a constant, independent of time. Because $x = 0$ at $t = 0$, we conclude that $\langle x \rangle = 0$ for all t . To calculate $\langle x^2(t) \rangle$, two integrations by parts are necessary, and we find that

$$\frac{d}{dt}\langle x^2(t) \rangle = 2D, \quad (7.42)$$

or

$$\langle x^2(t) \rangle = 2Dt. \quad (7.43)$$

We see that the random walk and the diffusion equation have the same time dependence. In d -dimensional space, $2D$ is replaced by $2dD$.

References and Suggestions for Further Reading

William R. Bennett, *Scientific and Engineering Problem-Solving with the Computer*, Prentice Hall (1976). Many random processes including the spread of disease are considered in Chapter 6.

J. M. Bernardo and A. F. M. Smith, *Bayesian Theory*, John Wiley & Sons (1994). Bayes Theorem is given concisely on page 2.

Philip R. Bevington and D. Keith Robinson, *Data Reduction and Error Analysis for the Physical Sciences*, second edition, McGraw-Hill (1992).

William S. Cleveland and Robert McGill, "Graphical perception and graphical methods for analyzing scientific data," *Science* **229**, 828 (1985). There is more to analyzing data than least squares fits.

A. K. Dewdney, "Computer Recreations (Five easy pieces for a do loop and random-number generator)," *Sci. Amer.* **252**(#4), 20 (1985).

Robert M. Eisberg, *Applied Mathematical Physics with Programmable Pocket Calculators*, McGraw-Hill (1976). Chapter 7 discusses entropy and the arrow of time.

Richard P. Feynman, Robert B. Leighton, and Matthew Sands, *The Feynman Lectures on Physics*, Addison-Wesley (1963). See Vol. 1, Chapter 26 for a discussion of the principle of least time and Vol. 2, Chapter 19, for a discussion of the principle of least action.

Peter R. Keller and Mary M. Keller, *Visual Cues*, IEEE Press (1993). A well illustrated book on data visualization techniques.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes*, second edition, Cambridge University Press (1992). See Chapter 15 for a general discussion of the modeling of data including general linear least squares and nonlinear fits.

F. Reif, *Statistical and Thermal Physics*, Berkeley Physics, Vol. 5, McGraw-Hill (1965). Chapter 2 introduces random walks.

David Ruelle, *Chance and Chaos*, Princeton Publishing Company (1993). A non-technical introduction to chaos theory that discusses the relation of chaos to randomness.

Charles Ruhla, *The Physics of Chance*, Oxford University Press (1992). A delightful book on probability in many contexts.

G. L. Squires, *Practical Physics*, third edition, Cambridge University Press (1985). An excellent text on the design of experiments and the analysis of data.

John R. Taylor, *An Introduction to Error Analysis*, University Science Books, Oxford University Press (1982).

Edward R. Tufte, *The Visual Display of Quantitative Information*, Graphics Press (1983).

Charles A. Whitney, *Random Processes in Physical Systems*, John Wiley and Sons (1990). An excellent introduction to random processes with many applications to astronomy.

Robert S. Wolff and Larry Yaeger, *Visualization of Natural Phenomena*, Springer-Verlag (1993). A CD-ROM disk is included with the book. An extensive list of references also is given.

Hugh D. Young, *Statistical Treatment of Experimental Data*, McGraw-Hill (1962).

Chapter 7

Two-Dimensional Trajectories and the ODE Interface

©2002 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
11 March 2002

We introduce two Java interfaces for the solution of systems of first-order differential equations and study motion in two dimensions.

7.1 Trajectories

You are probably familiar with two-dimensional trajectory problems in the absence of air resistance. For example, if a ball is thrown in the air with an initial velocity v_0 at an angle θ_0 with respect to the ground, how far will the ball travel in the horizontal direction, and what is its maximum height and time of flight? Suppose that a ball is released at a nonzero height h above the ground. What is the launch angle for the maximum range? Are your answers still applicable if air resistance is taken into account? We consider these and similar questions in the following.

Consider an object of mass m whose initial velocity \mathbf{v}_0 is directed at an angle θ_0 above the horizontal (see Figure 7.1a). The particle is subjected to gravitational and drag forces of magnitude mg and F_d ; the direction of the drag force is opposite to \mathbf{v} (see Figure 7.1b). Newton's equations of motion for the x and y components of the motion can be written as

$$m \frac{dv_x}{dt} = -F_d \cos \theta \quad (7.1a)$$

$$m \frac{dv_y}{dt} = -mg - F_d \sin \theta. \quad (7.1b)$$

As an example, let us consider a round steel ball of radius 4 cm. A reasonable assumption for a steel ball of this size and typical speed is that $F_d = k_2 v^2$. Because $v_x = v \cos \theta$ and $v_y = v \sin \theta$,

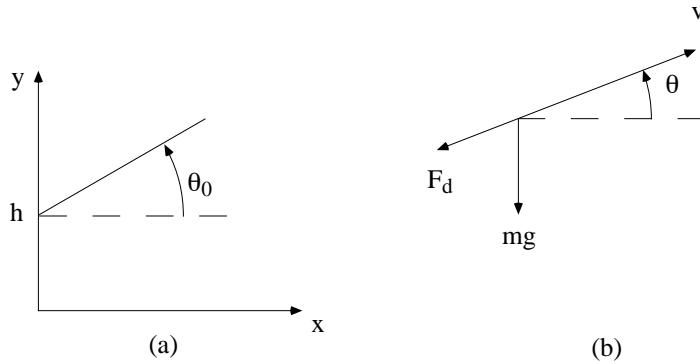


Figure 7.1: (a) A ball is thrown from a height h at an launch angle θ_0 measured with respect to the horizontal. The initial velocity is v_0 . (b) The gravitational and drag forces on a particle.

we can rewrite (7.1) as

$$\frac{dv_x}{dt} = -Cvv_x \quad (7.2a)$$

$$\frac{dv_y}{dt} = -g - Cvv_y, \quad (7.2b)$$

where $C = k_2/m$. Note that $-Cvv_x$ and $-Cvv_y$ are the x and y components of the drag force $-Cv^2$. Because (7.2) for the change in v_x and v_y involve the square of the velocity, $v^2 = v_x^2 + v_y^2$, we cannot calculate the vertical motion of the ball without reference to the horizontal motion, that is, the motion in the x and y direction is *coupled*.

The two-dimensional trajectory problem leads to a system of five first-order differential equations for the following variables: x , v_x , y , v_y , and t . We could, of course, solve these equations by incorporating a numerical algorithm directly into the class that defines the physics as we did in Chapters 5 and 6. This approach to solving equations works and may, in fact, be desirable for problems where the algorithm is unlikely to change and computational speed is the overriding factor (see Chapter ??). However, this approach is not very object-oriented. Changing the algorithm or the physics requires that we recode the class. But the physics and the algorithm are clearly separate concepts and should be treated as separate classes. To make this separation we introduce the `ODE` (ordinary differential equation) and `ODESolver` interfaces in Section 7.2.

7.2 Solving ODEs with Interfaces

Although interfaces can be used for solving individual differential equations, their strength is in dealing with systems of many equations. The key to using the OSP interfaces to solve differential equations is to use arrays for the variables and their derivatives. The `state` array contains the current value for each variable; the `rate` array is calculated on the fly whenever the rate of change of these variables is needed. These arrays are used by two interfaces, `ODE` and `ODESolver`. The `ODE`

interface defines the variables and the derivatives that the numerical method needs to compute the solution, and the `ODESolver` interface defines the numerical method.

The `ODE` interface contains two methods, `getState` and `getRate`, and is written as follows:

```
package org.opensourcephysics.numerics;
public interface ODE {
    public double[] getState();
    public void getRate(double[] state, double[] rate);
}
```

This interface allows us to encapsulate almost any physics problem that can be written in terms of ordinary differential equations.

The `ODESolver` interface defines methods that initialize and set the step size of a numerical method and is written as follows:

```
package org.opensourcephysics.numerics;
public interface ODESolver {
    public void initialize (double stepSize);
    public double step();
    public void setStepSize(double stepSize);
    public double getStepSize();
}
```

The `initialize` method sets the step size and, if necessary, allocates temporary arrays and initializes variables. It is usually invoked within the solver's constructor but should be invoked again if the number of differential equations change. The other methods are self explanatory.

To solve a system of differential equations, an object that implements the `ODE` interface is first created and then passed to a differential equation solver in the solver's constructor. For example, if we wish to solve a radioactive decay problem using the Euler algorithm, we set up the problem using the following three statements:

```
ODE decay = new Decay();
ODESolver odeSolver = new Euler(decay);
odeSolver.setStepSize(dt);
```

The differential equations in the `ODE` object are then solved by repeatedly invoking the solver's step method:

```
odeSolver.step();
```

The solver's step method advances the state by `dt`. The state array can be read after every step if the variables are to be plotted or analyzed in some fashion.

We first discuss the details using the Euler method. Subsequently, we will use other numerical methods that are predefined in the `OSP` numerics package (see Table 7.1). Of course, it is up to you to determine if the chosen numerical method is stable and accurate when applied to a given problem. The important point is that we only need to instantiate a different solver to switch numerical methods.

The `OSP` numerics package implements the Euler algorithm so that it can be used to solve any system of first-order differential equations. The Euler method takes a variable, *x* for example, and

Name	algorithm
Euler	Euler algorithm
Verlet	Verlet algorithm
RK4	Runge-Kutta fourth-order algorithm with fixed step size
RK45	Runge-Kutta fifth-order algorithm with variable step size

Table 7.1: Numerical methods included in the OSP package.

advances it to a new value, $x + \Delta x$. The change Δx is calculated using the rate of change of x with respect to some independent parameter, $\dot{x} = dx/dt$. We extend this idea to systems of first-order differential equations by labeling the $n + 1$ variables as $x_0, x_1, x_2, \dots, x_n$. The system of equations can be written in the form:

$$\dot{x}_0 = f_0(x_0, x_1, \dots, x_n) \quad (7.3a)$$

$$\dot{x}_1 = f_1(x_0, x_1, \dots, x_n) \quad (7.3b)$$

...

$$\dot{x}_n = f_n(x_0, x_1, \dots, x_n). \quad (7.3c)$$

Each equation specifies the rate of change of a variable in terms of all the other variables. The variable that is used to take the derivative can be any independent parameter, but it is often the time. By convention, we will always list this variable last.

We first apply this notation to the nuclear decay problem in Chapter 5, and write it as two differential equations:

$$\dot{x}_0 = -kx_0 \quad (7.4a)$$

$$\dot{x}_1 = 1. \quad (7.4b)$$

Equation (7.4a) is equivalent to $dN/dt = -kN$; (7.4b) is the rate of change for the time variable, $dt/dt = 1$. That is, x_0 is the number of radioactive nuclei, and x_1 is the time. Using this notation, the Euler algorithm for the i th step is

$$x_{0,i+1} = x_{0,i} - kx_{0,i} \Delta t \quad (7.5a)$$

$$x_{1,i+1} = x_{1,i} + \Delta t. \quad (7.5b)$$

The change in each variable is determined by evaluating the rate at the beginning of the i th step, $f_j(x_{0,i}, x_{1,i}, x_{2,i}, \dots, x_{n,i})$. If the rates are not constant, which they usually are not, there will be a truncation error in the new calculated values.

We next encapsulate the Euler algorithm in Listing 7.1. The `step()` method gets the current state from the ODE object and then sends this same array back to the ODE object to calculate the rates. This array passing is certainly not necessary for the Euler algorithm. However, higher order algorithms often evaluate the rate several times using intermediate values before a final state is calculated. Hence, the ODE interface requires a `getRate` method that evaluates the rate equations for arbitrary states, not just for the current state.

Listing 7.1: Implementation of the Euler algorithm to be used in conjunction with the ODE interface.

```
// Euler algorithm encapsulated as an object
package org.opensourcephysics.numerics;
public class Euler extends Object implements ODESolver {
    private double stepSize = 0.1;
    private int numberOfEquations = 0;
    private double[] rate;
    private ODE ode;

    public Euler(ODE _ode){
        ode = _ode;
        numberOfEquations = ode.getState().length;
        rate = new double[numberOfEquations];
    }

    public double step(){
        double[] state = ode.getState();
        ode.getRate(state, rate);
        for (int i = 0; i < numberOfEquations; i++) {
            state[i] = state[i] + stepSize*rate[i];
        }
        return stepSize;
    }

    public void setStepSize(double _stepSize){
        stepSize = _stepSize;
    }

    public void initialize (double stepSize){setStepSize(stepSize);}

    public double getStepSize() {return stepSize;}
}
```

Note that the state of the system is represented by an array (`state`), which in the decay example represents two variables. Corresponding to these two variables are two rates that are represented by the `rate` array. It might be tempting to choose the name “rates” for the latter array, but it is more consistent to choose a singular name. For example, we speak about the rate of change of the state of a system. The question of the most appropriate name for an array that represents many variables arises often, and it usually is better to choose the singular rather than the plural name.

The `Euler` class works in conjunction with any class that implements the `ODE` interface. Consider, for example, the radioactive decay problem that was discussed in Section 5.4. The `Decay` class encapsulates this problem in Listing 7.2. The most important method is, of course, `getRate`.

Listing 7.2: The `Decay` class.

```
package org.opensourcephysics.sip.ch7;
```

```

import org.opensourcephysics.numerics.ODE;

public class Decay implements ODE {

    double k;                                // decay constant
    double[] state = new double[2];           // [ particle number, time]

    public Decay() {
        k = 1;                                // decay constant
        state [0] = 1.0;                        // particle number
        state [1] = 0;                          // time
    }

    public double[] getState() {
        return state;
    }

    public void getRate(double[] state, double[] rate) {
        rate[0] = -k*state [0];                // dN/dt = -kN
        rate [1] = 1;                           // dt/dt = 1
    }
}

```

We now test the decay class using in `DecayApp`, which is shown in Listing 7.3.

Listing 7.3: Listing of `DecayApp`.

```

package org.opensourcephysics.sip.ch7;

import org.opensourcephysics.display.*;
import org.opensourcephysics.numerics.*;

public class DecayApp {
    public static void main(String[] args) {
        PlottingPanel plottingPanel = new PlottingPanel("Time", "Number", "Number versus Time");
        DrawingFrame plottingFrame = new DrawingFrame(plottingPanel);
        DatasetCollection datasetCollection = new DatasetCollection();
        plottingPanel.addDrawable(datasetCollection); // add dataset collection to plotting panel
        Decay decay = new Decay();                  // object that contains decay equations
        ODESolver odeSolver = new Euler(decay);    // use Euler method to solve decay equations
        odeSolver.setStepSize(0.1);
        for (int i = 0; i < 100; i++) {
            // append time and particle number
            datasetCollection.append(0, decay.state [1], decay.state [0]);
            odeSolver.step();
        }
        plottingPanel.repaint();
    }
}

```

Note that for simplicity, we have hard coded the initial number of particles. The decay object

and the ODE solver are created, but the decay object is created first so that the solver obtains a reference to this object. Also note that there is only one state array, and this array is created in the decay object. The state array is retrieved by other objects, such as the ODE solver. This array is used over and over as the solution is carried forward using the `step` method. It would be inefficient to allocate a new rate array at every step.

Another version of the exponential decay program (called `DecayAnimationApp`) that uses the Animation interface is available in the Chapter 7 package. You may wish to use this program to do Problem 7.1.

Exercise 7.1. Test of Decay.

- Compile `DecayApp` verify that its output is the same as what was obtained in Problem 5.2.
- Write a Euler-Richardson class for the ODE interface and modify `DecayApp` so that the ODE solver uses the Euler-Richardson algorithm. Is this solver more accurate than the Euler solver?
- Modify `DecayApp` so that the ODE solver is RK4 (see Table 7.1). What are the changes that you need to make? Discuss the advantages (and any disadvantages) of placing the algorithm in a separate class. Is the RK4 solver more accurate than the Euler solver? (The nature of the fourth-order Runge-Kutta algorithm (RK4) is discussed in Appendix 7A.)

7.3 Two-Dimensional Trajectories

Computing the trajectory of a particle in two dimensions is now straightforward. The equations of motion can be written as rate equations as follows:

$$\dot{x}_0 = x_1 \quad (7.6a)$$

$$\dot{x}_1 = F_x(x_0, x_1, x_2, x_3, x_4)/m \quad (7.6b)$$

$$\dot{x}_2 = x_3 \quad (7.6c)$$

$$\dot{x}_3 = F_y(x_0, x_1, x_2, x_3, x_4)/m \quad (7.6d)$$

$$\dot{x}_4 = 1. \quad (7.6e)$$

The forces F_x and F_y are determined by (7.2a) and (7.2b), respectively. In Exercise 7.2 we modify the `DecayApp` program to solve these equations.

Problem 7.2. Trajectory of a steel ball

- Create a class that encapsulates (7.6). Include a method to set the initial conditions using the angle θ_0 above the horizontal and the initial velocity of v_0 .
- Compute the two-dimensional trajectory of a ball moving in air and plot y as a function of x . Use the Euler-Richardson algorithm. As a check on your program, first neglect air resistance so that you can compare your computed results with the exact results. Assume that a ball is thrown from ground level at an angle θ_0 above the horizontal with an initial velocity of $v_0 = 15$ m/s. Vary θ_0 and show that the maximum range occurs at $\theta_0 = \theta_{\max} = 45^\circ$. What is R_{\max} , the maximum range, at this angle? Compare your numerical value to the analytical result $R_{\max} = v_0^2/g$.

- c. Suppose that a steel ball is thrown from a height h at an angle θ_0 above the horizontal with the same initial speed as in part (b). If you neglect air resistance, do you expect θ_{\max} to be larger or smaller than 45° ? Although this problem can be solved analytically, you can determine the numerical value of θ_{\max} without changing your program. What is θ_{\max} for $h = 2$ m? By what percent is the range R changed if θ is varied by 2% from θ_{\max} ?
- d. Consider the effects of air resistance on the range and optimum angle of a steel ball. For a ball of mass 7 kg and cross-sectional area 0.01 m^2 , the parameter C_2 is approximately 0.1. What are the units of C_2 ? However, it is convenient to exaggerate the effects of air resistance so that you can more easily determine the qualitative nature of the effects. Compute the optimum angle for $h = 2$ m, $v_0 = 30$ m/s, and $C = k_2/m = 0.1$, and compare your answer to the value found in part (c). Is R more or less sensitive to changes in θ_0 from θ_{\max} than in part (b)? Determine the optimum launch angle and the corresponding range for the more realistic value of $C = 0.001$. A detailed discussion of the maximum range of the ball has been given by Lichtenberg and Wills (see references).

Problem 7.3. Coupled motion

Consider the motion of two identical objects that both start from a height h . One object is dropped vertically from rest and the other is thrown with a horizontal velocity v_0 . Which object reaches the ground first?

- a. Give reasons for your answer assuming that air resistance can be neglected.
- b. Assume that air resistance cannot be neglected and that the drag force is proportional to v^2 . Give reasons for your anticipated answer for this case. Then perform numerical simulations using, for example, $C = k_2/m = 0.1$, $h = 10$ m, and $v_0 = 30$ m/s. Are your qualitative results consistent with your anticipated answer? If they are not, the source of the discrepancy might be an error in your program. Or the discrepancy might be due to your failure to anticipate the effects of the coupling between the vertical and horizontal motion.
- c. Suppose that the drag force is proportional to v rather than to v^2 . Is your anticipated answer similar to that in part (b)? Do a simulation to test your intuition.

Problem 7.4. ODE and the Pendulum

Rewrite `PendulumApp` from Chapter 6 so that it uses the ODE interface. What do you have to do to consider a damped pendulum?

7.4 Crossed E and B Fields

[xx not finished xx] Coupled equations of motion occur in electrodynamics when a charged particle moves through a magnetic field. Consider a particle of mass m and charge q traveling with velocity \mathbf{v} through an electric field \mathbf{E} and a magnetic field \mathbf{B} . The equations of motion can be written in vector form as:

$$m\dot{\mathbf{r}} = \vec{\mathbf{v}} \quad (7.7)$$

and

$$\dot{\vec{v}} = q\vec{E} + q\vec{v} \times \vec{B}. \quad (7.8)$$

If we restrict ourselves to the special case of constant perpendicular fields with the magnetic field in the \hat{z} direction and the electric field in the \hat{y} direction, then the net force acts only in the x - y plane. The rate equations for the velocity components can then be written as

$$m \frac{dv_x}{dt} = qv_y B_z \quad (7.9a)$$

$$m \frac{dv_y}{dt} = qE_y - qv_x B_z. \quad (7.9b)$$

To solve these equations and show an animation, we define a drawable charged particle class as follows:

7.5 Spin of Balls in Flight

[xx not finished xx] Of particular interest to sports fans is the curve of balls in flight due to their rotation and the effect of air resistance on the range and speed of baseballs and golf balls.

The equations of motion are

$$\dot{\vec{r}} = \vec{v} \quad (7.10)$$

and

$$\dot{\vec{v}} = \vec{g} - k_D v \vec{v} + k_L \omega \times \vec{v}, \quad (7.11)$$

where k_D and k_L are the drag and lift coefficients, respectively. Typical values for a baseball are $k_D = 0.0037$ and $k_L = 0.0029$

The rate equations for the velocity components can be written as

$$m \frac{dv_x}{dt} = -k_D v v_x + k_L (\omega_y v_z - \omega_z v_y) \quad (7.12a)$$

$$m \frac{dv_y}{dt} = -k_D v v_y + k_L (\omega_z v_x - \omega_x v_z) \quad (7.12b)$$

$$m \frac{dv_z}{dt} = -k_D v v_z + k_L (\omega_x v_y - \omega_y v_x) - g. \quad (7.12c)$$

7.6 Projects

Applications of the ideas and methods that we have discussed for trajectories are important in the physics of clouds. For example, to understand the behavior of falling water droplets, it is necessary to take into account drag resistance as well as droplet growth by condensation and other mechanisms. Because of the variety and complexity of the mechanisms, simulation plays an essential role.

Another area of interest is sports. The trajectory of various shapes through the air such as baseballs, footballs, and javelins can be very involved because the complete description of their motion requires three spatial coordinates, three angles, and six derivatives.

Project 7.5. Sports

[xx see AJP articles on the physics of sports. xx]

Project 7.6. Comparison of algorithms

- a. Appendix 7A summarizes some of the more commonly used algorithms for the numerical solution of Newton's equations of motion. Compare the Euler, Euler-Richardson, Verlet, and fourth-order Runge-Kutta algorithms by computing $x(t)$, $v(t)$, and E_n , where E_n is the total energy after the n th step for the Morse potential $V(x) = e^{-2x} - 2e^{-x}$ considered in Project 6.17. Which one of these algorithms is the best trade-off between speed, accuracy, and simplicity?
- b. Consider a particle of unit mass in the Lennard-Jones potential (see Section ??)

$$V(r) = 4\left[\left(\frac{1}{r}\right)^{12} - \left(\frac{1}{r}\right)^6\right]. \quad (7.13)$$

Consider the motion of a particle in two dimensions. Choose initial conditions such that the particle experiences the steeply repulsive part of the potential and test the various algorithms considered in part (a).

Appendix 7A: Numerical Integration of Newton's Equation of Motion

We summarize several of the common finite difference methods for the solution of Newton's equations of motion with continuous force functions. The variety of algorithms currently in use is evidence that no single method is superior under all conditions.

To simplify the notation, we consider the motion of a particle in one dimension and write Newton's equations of motion in the form

$$\frac{dv}{dt} = a(t), \quad (7.14a)$$

and

$$\frac{dx}{dt} = v(t), \quad (7.14b)$$

where $a(t) \equiv a(x(t), v(t), t)$. The goal of finite difference methods is to determine the values of x_{n+1} and v_{n+1} at time $t_{n+1} = t_n + \Delta t$. We already have seen that Δt must be chosen so that the integration method generates a stable solution. If the system is conservative, Δt must be sufficiently small so that the total energy is conserved to the desired accuracy.

The nature of many of the integration algorithms can be understood by expanding $v_{n+1} = v(t_n + \Delta t)$ and $x_{n+1} = x(t_n + \Delta t)$ in a Taylor series. We write

$$v_{n+1} = v_n + a_n \Delta t + O((\Delta t)^2), \quad (7.15a)$$

and

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 + O((\Delta t)^3). \quad (7.15b)$$

The familiar Euler method is equivalent to retaining the $O(\Delta t)$ terms in (7.15a):

$$v_{n+1} = v_n + a_n \Delta t \quad (7.16a)$$

and

$$x_{n+1} = x_n + v_n \Delta t. \quad (\text{Euler}) \quad (7.16b)$$

Because order Δt terms are retained in (7.16), the local truncation error, the error in one time step, is order $(\Delta t)^2$. The global error, the total error over the time of interest, due to the accumulation of errors from step to step is order Δt . This estimate of the global error follows from the fact that the number of steps into which the total time is divided is proportional to $1/\Delta t$. Hence, the order of the global error is reduced by a factor of $1/\Delta t$ relative to the local error. We say that a method is n th order if its global error is order $(\Delta t)^n$. The Euler method is an example of a *first-order* method.

The Euler method is asymmetrical because it advances the solution by a time step Δt , but uses information about the derivative only at the beginning of the interval. We already have found that the accuracy of the Euler method is limited and that frequently its solutions are not stable. We also found that a simple modification of (7.16) yields solutions that are stable for oscillatory systems. For completeness, we repeat the Euler-Cromer algorithm or last-point approximation here:

$$v_{n+1} = v_n + a_n \Delta t, \quad (7.17a)$$

and

$$x_{n+1} = x_n + v_{n+1} \Delta t. \quad (\text{Euler-Cromer}) \quad (7.17b)$$

An obvious way to improve the Euler method is to use the mean velocity during the interval to obtain the new position. The corresponding *midpoint* method can be written as

$$v_{n+1} = v_n + a_n \Delta t, \quad (7.18a)$$

and

$$x_{n+1} = x_n + \frac{1}{2}(v_{n+1} + v_n) \Delta t. \quad (\text{midpoint}) \quad (7.18b)$$

Note that if we substitute (7.18a) for v_{n+1} into (7.18b), we obtain

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n \Delta t^2. \quad (7.19)$$

Hence, the midpoint method yields second-order accuracy for the position and first-order accuracy for the velocity. Although the midpoint approximation yields exact results for constant acceleration, it usually does not yield much better results than the Euler method. In fact, both methods are equally poor, because the errors increase with each time step.

A higher order method whose error is bounded is the *half-step* method. In this method the average velocity during an interval is taken to be the velocity in the middle of the interval. The

half-step method can be written as

$$v_{n+\frac{1}{2}} = v_{n-\frac{1}{2}} + a_n \Delta t, \quad (7.20a)$$

and

$$x_{n+1} = x_n + v_{n+\frac{1}{2}} \Delta t. \quad (\text{half-step}) \quad (7.20b)$$

Note that the half-step method is not self-starting, that is, (7.20a) does not allow us to calculate $v_{\frac{1}{2}}$. This problem can be overcome by adopting the Euler algorithm for the first half step:

$$v_{\frac{1}{2}} = v_0 + \frac{1}{2} a_0 \Delta t. \quad (7.20c)$$

Because the half-step method is stable, it is a common textbook method.

The *Euler-Richardson method* was introduced in Chapter 5, but we list it here for completeness.

$$v_m = v_n + \frac{1}{2} a_n \Delta t \quad (7.21a)$$

$$y_m = y_n + \frac{1}{2} v_n \Delta t \quad (7.21b)$$

$$t_m = t_n + \frac{1}{2} \Delta t \quad (7.21c)$$

$$a_m = F(y_m, v_m, t_m)/m, \quad (7.21d)$$

and

$$v_{n+1} = v_n + a_m \Delta t \quad (7.22a)$$

$$y_{n+1} = y_n + v_m \Delta t. \quad (\text{Euler-Richardson}) \quad (7.22b)$$

We will see that the Euler-Richardson algorithm is equivalent to the second-order Runge-Kutta algorithm (see (7.31)).

One of the most common drift-free higher order algorithms is commonly attributed to Verlet. We write the Taylor series expansion for x_{n-1} in a form similar to (7.15b):

$$x_{n-1} = x_n - v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2. \quad (7.23)$$

If we add the forward and reverse forms, (7.15b) and (7.23) respectively, we obtain

$$x_{n+1} + x_{n-1} = 2x_n + a_n (\Delta t)^2 + O((\Delta t)^4) \quad (7.24)$$

or

$$x_{n+1} = 2x_n - x_{n-1} + a_n (\Delta t)^2. \quad (7.25a)$$

Similarly, the subtraction of the Taylor series for x_{n+1} and x_{n-1} yields

$$v_n = \frac{x_{n+1} - x_{n-1}}{2 \Delta t}. \quad (\text{original Verlet}) \quad (7.25b)$$

Note that the global error associated with the *Verlet* algorithm (7.25) is third-order for the position and second-order for the velocity. However, the velocity plays no part in the integration of the

equations of motion. In the numerical analysis literature, the Verlet algorithm is known as the “explicit central difference method.”

Because this form of the Verlet algorithm is not self-starting, another algorithm must be used to obtain the first few terms. An additional problem is that the new velocity (7.25b) is found by computing the difference between two quantities of the same order of magnitude. As we discussed in Chapter 5, such an operation results in a loss of numerical precision and may give rise to roundoff errors.

A mathematically equivalent version of the original Verlet algorithm is given by

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 \quad (7.26a)$$

and

$$v_{n+1} = v_n + \frac{1}{2} (a_{n+1} + a_n) \Delta t. \quad (\text{velocity Verlet}) \quad (7.26b)$$

We see that (7.26), known as the *velocity* form of the Verlet algorithm, is self-starting and minimizes roundoff errors. Because we will make no use of (7.25) in the text, we will refer to (7.26) as the Verlet algorithm.

We can derive (7.26) from (7.25) by the following considerations. We first solve (7.25b) for x_{n-1} and write $x_{n-1} = x_{n+1} - 2v_n \Delta t$. If we substitute this expression for x_{n-1} into (7.25a) and solve for x_{n+1} , we find the form (7.26a). Then we use (7.25b) to write v_{n+1} as:

$$v_{n+1} = \frac{x_{n+2} - x_n}{2\Delta t}, \quad (7.27)$$

and use (7.25a) to obtain $x_{n+2} = 2x_{n+1} - x_n + a_{n+1}(\Delta t)^2$. If we substitute this form for x_{n+2} into (7.27), we obtain

$$v_{n+1} = \frac{x_{n+1} - x_n}{\Delta t} + \frac{1}{2} a_{n+1} \Delta t. \quad (7.28)$$

Finally, we use (7.26a) for x_{n+1} to eliminate $x_{n+1} - x_n$ from (7.28); after some algebra we obtain the desired result (7.26b).

Another useful algorithm that avoids the roundoff errors of the original Verlet algorithm is due to Beeman and Schofield. We write the *Beeman* algorithm in the form:

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{6} (4a_n - a_{n-1})(\Delta t)^2 \quad (7.29a)$$

and

$$v_{n+1} = v_n + \frac{1}{6} (2a_{n+1} + 5a_n - a_{n-1}) \Delta t. \quad (7.29b)$$

Note that (7.29) does not calculate particle trajectories more accurately than the Verlet algorithm. Its advantage is that in general it does a better job of maintaining energy conservation. However, the Beeman algorithm is not self-starting.

The most common finite difference method for solving ordinary differential equations is the *Runge-Kutta* method. To explain the method, we first consider the solution of the first-order

differential equation

$$\frac{dx}{dt} = f(x, t). \quad (7.30)$$

The *second-order* Runge-Kutta solution of (7.30) can be written using standard notation as:

$$k_1 = f(x_n, t_n)\Delta t \quad (7.31a)$$

$$k_2 = f\left(x_n + \frac{k_1}{2}, t_n + \frac{\Delta t}{2}\right)\Delta t \quad (7.31b)$$

$$x_{n+1} = x_n + k_2 + O((\Delta t)^3). \quad (7.31c)$$

The interpretation of (7.31) is as follows. The Euler method assumes that the slope $f(x_n, t_n)$ at (x_n, t_n) can be used to extrapolate to the next step, that is, $x_{n+1} = x_n + f(x_n, t_n)\Delta t$. A plausible way of making a better estimate of the slope is to use the Euler method to extrapolate to the midpoint of the interval and then to use the midpoint slope across the full width of the interval. Hence, the Runge-Kutta estimate for the slope is $f(x^*, t_n + \frac{1}{2}\Delta t)$, where $x^* = x_n + \frac{1}{2}f(x_n, t_n)\Delta t$ (see (7.31b)).

The application of the second-order Runge-Kutta method to Newton's equation of motion (7.14) yields

$$k_{1v} = a_n(x_n, v_n, t_n)\Delta t \quad (7.32a)$$

$$k_{1x} = v_n\Delta t \quad (7.32b)$$

$$k_{2v} = a\left(x_n + \frac{k_{1x}}{2}, v_n + \frac{k_{1v}}{2}, t_n + \frac{\Delta t}{2}\right)\Delta t \quad (7.32c)$$

$$k_{2x} = (v_n + \frac{k_{1v}}{2})\Delta t, \quad (7.32d)$$

and

$$v_{n+1} = v_n + k_{2v} \quad (7.32e)$$

$$x_{n+1} = x_n + k_{2x} \quad (\text{second-order Runge Kutta}) \quad (7.32f)$$

In the *fourth-order* Runge-Kutta algorithm, the derivative is computed at the beginning of the time interval, in two different ways at the middle of the interval, and again at the end of the interval. The two estimates of the derivative at the middle of the interval are given twice the weight of the other two estimates. The algorithm for the solution of (7.30) can be written in standard notation as

$$k_1 = f(x_n, t_n)\Delta t \quad (7.33a)$$

$$k_2 = f\left(x_n + \frac{k_1}{2}, t_n + \frac{\Delta t}{2}\right)\Delta t \quad (7.33b)$$

$$k_3 = f\left(x_n + \frac{k_2}{2}, t_n + \frac{\Delta t}{2}\right)\Delta t \quad (7.33c)$$

$$k_4 = f(x_n + k_3, t_n + \Delta t)\Delta t, \quad (7.33d)$$

and

$$x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \quad (7.34)$$

The application of the fourth-order Runge-Kutta method to Newton's equation of motion (7.14) yields

$$k_{1v} = a(x_n, v_n, t_n)\Delta t \quad (7.35a)$$

$$k_{1x} = v_n\Delta t \quad (7.35b)$$

$$k_{2v} = a(x_n + \frac{k_{1x}}{2}, v_n + \frac{k_{1v}}{2}, t_n + \frac{\Delta t}{2})\Delta t \quad (7.35c)$$

$$k_{2x} = (v_n + \frac{k_{1v}}{2})\Delta t \quad (7.35d)$$

$$k_{3v} = a(x_n + \frac{k_{2x}}{2}, v_n + \frac{k_{2v}}{2}, t_n + \frac{\Delta t}{2})\Delta t \quad (7.35e)$$

$$k_{3x} = (v_n + \frac{k_{2v}}{2})\Delta t \quad (7.35f)$$

$$k_{4v} = a(x_n + k_{3x}, v_n + k_{3v}, t + \Delta t) \quad (7.35g)$$

$$k_{4x} = (v_n + k_{3x})\Delta t, \quad (7.35h)$$

and

$$v_{n+1} = v_n + \frac{1}{6}(k_{1v} + 2k_{2v} + 2k_{3v} + k_{4v}) \quad (7.36a)$$

$$x_{n+1} = x_n + \frac{1}{6}(k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x}). \quad (\text{fourth-order Runge-Kutta}) \quad (7.36b)$$

Because Runge-Kutta methods are self-starting, they are frequently used to obtain the first few iterations for an algorithm that is not self-starting.

Our last example is the *predictor-corrector* method. The idea is to first *predict* the value of the new position:

$$x_p = x_{n-1} + 2v_n\Delta t. \quad (\text{predictor}) \quad (7.37)$$

The predicted value of the position allows us to predict the acceleration a_p . Then using a_p , we obtain the *corrected* values of v_{n+1} and x_{n+1} :

$$v_{n+1} = v_n + \frac{1}{2}(a_p + a_n)\Delta t \quad (7.38a)$$

$$x_{n+1} = x_n + \frac{1}{2}(v_{n+1} + v_n)\Delta t. \quad (\text{corrected}) \quad (7.38b)$$

The corrected values of x_{n+1} and v_{n+1} are used to obtain a new predicted value of a_{n+1} , and hence a new predicted value of v_{n+1} and x_{n+1} . This process is repeated until the predicted and corrected values of x_{n+1} differ by less than the desired value. Note that the predictor-corrector method is not self-starting.

As we have emphasized, there is no single algorithm for solving Newton's equations of motion that is superior under all conditions. It is usually a good idea to start with a simple algorithm, and then to try a higher order algorithm to see if any real improvement is obtained.

References and Suggestions for Further Reading

F. S. Acton, *Numerical Methods That Work*, The Mathematical Association of America (1990), Chapter 5.

Paul L. DeVries, *A First Course in Computational Physics*, John Wiley & Sons (1994).

Tao Pang, *Computational Physics*, Cambridge University Press (1997).

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes*, second edition, Cambridge University Press (1992). Chapter 16 discusses the integration of ordinary differential equations.

Chapter 8

The Dynamics of Many Particle Systems

©2000 by Harvey Gould and Jan Tobochnik
6 December 2000

We simulate the dynamical behavior of many particle systems and observe their qualitative features. Some of the basic ideas of equilibrium statistical mechanics and kinetic theory are introduced.

8.1 Introduction

Gases, liquids, and solids are examples of systems that contain many mutually interacting particles. Given our knowledge of the laws of physics at the microscopic level, how can we understand the observed behavior of these systems and more complex systems such as polymers and proteins? As an example, consider two cups of water prepared under similar conditions. Each cup contains approximately 10^{24} molecules which, to a good approximation, move according to the laws of classical physics. Although the intermolecular forces produce a complicated trajectory for each molecule, the observable properties of the water in each cup are indistinguishable and are easy to describe. For example, the temperature of the water in each cup is independent of time even though the positions and velocities of the individual molecules are changing continually.

One way to understand the behavior of a many particle system is to begin from the known intermolecular interactions and do a computer simulation of its dynamics. This approach, known as the **molecular dynamics** method, has been applied to systems of several hundred to a million particles and has given us much insight into the behavior of gases, liquids, and solids.

A knowledge of the trajectories of 10^4 or even 10^{24} particles is not helpful unless we know the right questions to ask. What are the useful parameters needed to describe these systems? What are the essential characteristics and regularities exhibited by many particle systems? From our study of chaotic systems, we might suspect that the only meaningful quantities we can compute

are averages over the trajectories, rather than the trajectories themselves. Questions such as these are addressed by statistical mechanics and many of the ideas of statistical mechanics are discussed in this chapter. However, the only background needed for this chapter is a knowledge of Newton's laws of motion.

8.2 The Intermolecular Potential

The first step is to specify the model system we wish to simulate. For simplicity, we assume that the dynamics can be treated classically and that the molecules are spherical and chemically inert. We also assume that the force between any pair of molecules depends only on the distance between them. In this case the total potential energy U is a sum of two-particle interactions:

$$U = u(r_{12}) + u(r_{13}) + \cdots + u(r_{23}) + \cdots = \sum_{i=1}^{N-1} \sum_{j=i+1}^N u(r_{ij}), \quad (8.1)$$

where $u(r_{ij})$ depends only on the magnitude of the distance \mathbf{r}_{ij} between particles i and j . The pairwise interaction form (8.1) is appropriate for simple liquids such as liquid argon.

In principle, the form of $u(r)$ for electrically neutral molecules can be constructed by a first principles quantum mechanical calculation. Such a calculation is very difficult, and it usually is sufficient to choose a simple phenomenological form for $u(r)$. The most important features of $u(r)$ for simple liquids are a strong repulsion for small r and a weak attraction at large r . The repulsion for small r is a consequence of the Pauli exclusion principle. That is, the electron clouds of two molecules must distort to avoid overlap, causing some of the electrons to be in different quantum states. The net effect is an increase in kinetic energy and an effective repulsive force between the electrons, known as **core repulsion**. The dominant weak attraction at larger r is due to the mutual polarization of each molecule; the resultant attractive force is called the **van der Waals** force.

One of the most common phenomenological forms of $u(r)$ is the Lennard-Jones potential:

$$u(r) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right]. \quad (8.2)$$

A plot of the Lennard-Jones potential is shown in Figure 8.1. The r^{-12} form of the repulsive part of the interaction has been chosen for convenience only. The Lennard-Jones potential is parameterized by a length σ and an energy ϵ . Note that $u(r) = 0$ at $r = \sigma$, and that $u(r)$ is essentially zero for $r > 3\sigma$. The parameter ϵ is the depth of the potential at the minimum of $u(r)$; the minimum occurs at a separation $r = 2^{1/6}\sigma$. The parameters ϵ and σ of the Lennard-Jones potential which give good agreement with the experimental properties of liquid argon are $\epsilon = 1.65 \times 10^{-21} \text{ J}$ and $\sigma = 3.4 \text{ \AA}$.

Problem 8.1. Qualitative properties of the Lennard-Jones interaction

Write a short program or use a graphics package to plot the Lennard-Jones potential (8.1) and the magnitude of the corresponding force:

$$\mathbf{f}(r) = -\nabla u(r) = \frac{24\epsilon}{r} \left[2\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^6 \right] \hat{\mathbf{r}}. \quad (8.3)$$

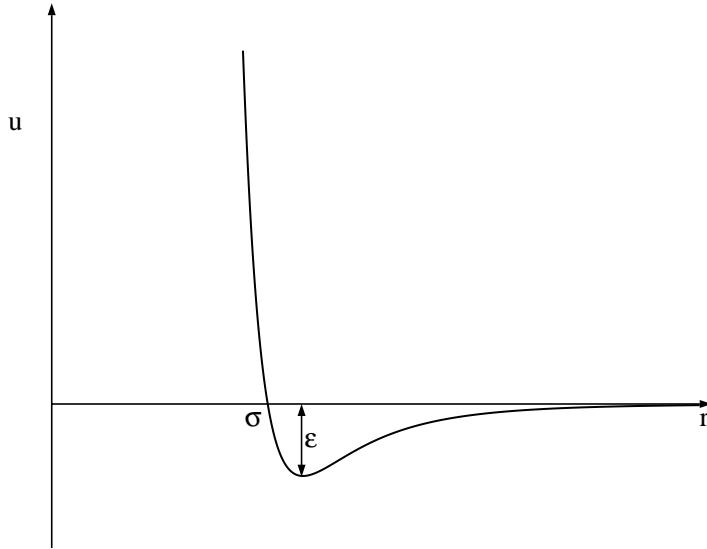


Figure 8.1: Plot of the Lennard-Jones potential $u(r)$. Note that the potential is characterized by a length σ and an energy ϵ .

What is the value of $u(r)$ for $r = 0.8\sigma$? How much does u increase if r is decreased to $r = 0.72\sigma$, a decrease of 10%? What is the value of u at $r = 2.5\sigma$? At what value of r does the force equal zero?

8.3 The Numerical Algorithm

Now that we have specified the interaction between the particles, we need to introduce a numerical integration method for computing the trajectory of each particle. As might be expected, we need to use at least a second-order algorithm to maintain conservation of energy for the times of interest in molecular dynamics simulations. We adopt the commonly used algorithm:

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 \quad (8.4a)$$

$$v_{n+1} = v_n + \frac{1}{2} (a_{n+1} + a_n) \Delta t. \quad (8.4b)$$

To simplify the notation, we have written the algorithm for only one component of the particle's motion. The new position is used to find the new acceleration a_{n+1} which is used together with a_n to obtain the new velocity v_{n+1} . The algorithm represented by (8.4) is a convenient form of the **Verlet algorithm** (see Appendix 5A).

8.4 Boundary Conditions

A useful simulation must incorporate all the relevant features of the physical system of interest. The ultimate goal of our simulations is to understand the behavior of bulk systems—systems of the order of $N \sim 10^{23} - 10^{25}$ particles. In bulk systems the fraction of particles near the walls of the container is negligibly small. However, the number of particles that can be studied in a molecular dynamics simulation is typically $10^3 - 10^5$, although as many as 10^6 particles or more, can be studied on present-day supercomputers. For these small systems the fraction of particles near the walls of the container is significant, and hence the behavior of such a system would be dominated by surface effects.

The most common way of minimizing surface effects and to simulate more closely the properties of a bulk system is to use what are known as **periodic boundary conditions**. First consider a one-dimensional “box” of N particles that are constrained to move on a line of length L . The ends of the line serve as imaginary walls. The usual application of periodic boundary conditions is equivalent to considering the line to be a circle (see Figure 8.2). The distance between the particles is measured along the arc, and hence the maximum separation between any two particles is $L/2$.

The computer code for periodic boundary conditions is straightforward. If a particle leaves the box by crossing a boundary, we add or subtract L to the coordinate. One simple way is to use an IF statement after the particles have been moved:

```
IF x > L then
    LET x = x - L
ELSE IF x < 0 then
    LET x = x + L
END IF
```

To compute the minimum distance dx between particles 1 and 2 at $x(1)$ and $x(2)$ respectively, we can write

```
LET dx = x(1) - x(2)
IF dx > 0.5*L then
    LET dx = dx - L
ELSE IF dx < -0.5*L then
    LET dx = dx + L
END IF
```

The generalization of this application of periodic boundary conditions to two dimensions is straightforward if we imagine a box with opposite edges joined so that the box becomes the surface of a torus (the shape of a doughnut and a bagel).

We now discuss the motivation for this choice of boundary conditions. Imagine a set of N particles in a two-dimensional cell. The use of periodic boundary conditions implies that this central cell is duplicated an infinite number of times to fill two-dimensional space. Each image cell contains the original particles in the same relative positions as the central cell. Figure 8.3 shows the first several image cells for $N = 2$ particles. Periodic boundary conditions yield an infinite system, although the motion of particles in the image cells is identical to the motion of the particles

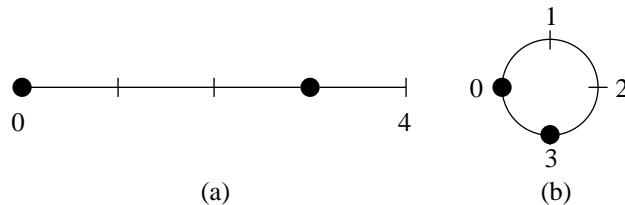


Figure 8.2: (a) Two particles at $x = 0$ and $x = 3$ on a line of length $L = 4$; the distance between the particles is 3. (b) The application of periodic boundary conditions for short range interactions is equivalent to thinking of the line as forming a circle of circumference L . In this case the minimum distance between the two particles is 1.

in the central cell. These boundary conditions also imply that every point in the cell is equivalent and that there is no surface. The shape of the central cell must be such that the cell fills space under successive translations.

As a particle moves in the original cell, its periodic images move in the image cells. Hence only the motion of the particles in the central cell needs to be followed. When a particle enters or leaves the central cell, the move is accompanied by an image of that particle leaving or entering a neighboring cell through the opposite face.

The total force on a given particle i is due to the force from every other particle j within the central cell and from the periodic images of particle j . That is, if particle i interacts with particle j in the central cell, then particle i interacts with all the periodic replicas of particle j . Hence in general, there are an infinite number of contributions to the force on any given particle. For long range interactions such as the Coulomb potential, these contributions have to be included using special methods. However, for short range interactions, we may reduce the number of contributions by adopting the **minimum image** or nearest image approximation. This approximation implies that particle i in the central cell interacts only with the nearest image of particle j ; the interaction is set equal to zero if the distance of the image from particle i is greater than $L/2$. An example of the minimum image condition is shown in Figure 8.3. Note that the minimum image approximation implies that the calculation of the total force on all N particles due to pairwise interactions involves a maximum of $N(N - 1)/2$ contributions.

8.5 Units

To reduce the possibility of roundoff error, it is useful to choose units so that the computed quantities are neither too small nor too large. Because the values of the distance and the energy associated with typical liquids are very small in SI units, we choose the Lennard-Jones parameters σ and ϵ to be the units of distance and energy, respectively. (The values of σ and ϵ for argon are given in Table 8.1.) We also choose the unit of mass to be the mass of one atom, m . We can express all other quantities in terms of σ , ϵ , and m . For example, we measure velocities in units of $(\epsilon/m)^{1/2}$, and the time in units of $\sigma(m/\epsilon)^{1/2}$. If we take $m = 6.69 \times 10^{-26}$ kg, the mass of an argon atom, then the unit of time is 2.17×10^{-12} s. The units of some of the physical quantities

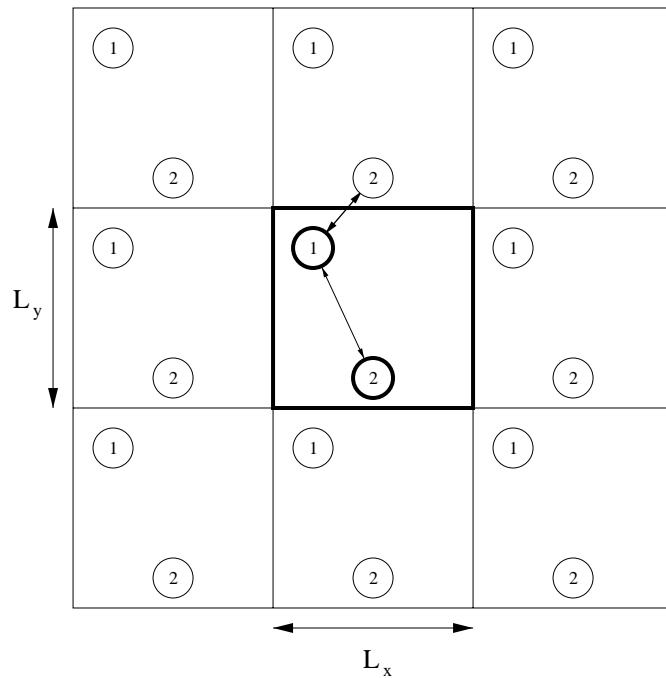


Figure 8.3: Example of the minimum image approximation in two dimensions. The minimum image distance convention implies that the separation between particles 1 and 2 is given by the shorter of the two distances shown.

of interest are shown in Table 8.1. All program variables are in reduced units, e.g., the time in our molecular dynamics program is expressed in units of $\sigma(m/\epsilon)^{1/2}$. As an example, suppose that we run our molecular dynamics program for 2000 time steps with a time step $\tau = 0.01$. The total time of our run is $2000 \times 0.01 = 20$ in reduced units or 4.34×10^{-11} s for argon (see Table 8.1). The total time of a typical molecular dynamics simulation is in the range of $10 - 10^4$ in reduced units, corresponding to a duration of approximately $10^{-11} - 10^{-9}$ s.

8.6 A Molecular Dynamics Program

In the following, we develop a molecular dynamics simulation of a two-dimensional system of particles interacting via the Lennard-Jones potential. We choose two rather than three dimensions because it is easier to visualize the results and the calculations are not as time consuming. The structure of Program `md` is given in the following:

```
PROGRAM md
PUBLIC x(36),y(36),vx(36),vy(36),ax(36),ay(36)
PUBLIC N,Lx,Ly,dt,dt2
```

quantity	unit	value for argon
length	σ	3.4×10^{-10} m
energy	ϵ	1.65×10^{-21} J
mass	m	6.69×10^{-26} kg
time	$\sigma(m/\epsilon)^{1/2}$	2.17×10^{-12} s
velocity	$(\epsilon/m)^{1/2}$	1.57×10^2 m/s
force	ϵ/σ	4.85×10^{-12} N
pressure	ϵ/σ^2	1.43×10^{-2} N · m ⁻¹
temperature	ϵ/k	120 K

Table 8.1: The system of units used in the molecular dynamics simulations of particles interacting via the Lennard-Jones potential. The numerical values of σ , ϵ , and m are for argon. The quantity k is Boltzmann's constant and has the value $k = 1.38 \times 10^{-23}$ J/K. The unit of pressure is for a two-dimensional system.

```

LIBRARY "csgraphics"
CALL initial(t,ke,kecum,pecum,vcum,area)
CALL set_up_windows(#1,#2)
CALL accel(pe,virial)
LET E = ke + pe           ! total energy
LET ncum = 0               ! number of times data accumulated
LET flag$ = ""
DO
    CALL show_positions(flag$,#2)
    CALL Verlet(t,ke,pe,virial)
    CALL show_output(t,ke,pe,virial,kecum,vcum,ncum,area,#1)
LOOP until flag$ = "stop"
CALL save_config
END

```

The x - and y -components of the positions, velocities, and accelerations are represented by arrays and are declared as public variables (cf. [Appendix 3C](#)) because they are used in almost all of the subroutines. These arrays are dimensioned in a PUBLIC statement and are declared in a DECLARE PUBLIC statement in each subroutine in which they are used. An array that is declared in a PUBLIC statement is not dimensioned in a DIM statement. The nature of the passed variables and the subroutines are discussed in the following.

Because the system is deterministic, the nature of the motion is determined by the initial conditions. An appropriate choice of the initial conditions is more difficult than might first appear. For example, how do we choose the initial configuration (a set of positions and velocities) to correspond to a fluid at a desired temperature? We postpone a discussion of such questions until Section 8.7, and instead we use initial conditions that have been computed previously.

The initial conditions can be incorporated into the program by either reading a data file or storing the information within a program using DATA and READ statements. The following program illustrates the use of the DATA and READ statements:

```

PROGRAM example_data
DIM x(6)
DATA 4.48,3.06,0.20,2.08,3.88,3.36
FOR i = 1 to 6
    READ x(i)                      ! reads input from DATA statement
NEXT i
END

```

The location of the DATA statements is unimportant, but the data must be stored in the order in which it will be read by the READ statements.

The following version of SUB initial uses the DATA and READ statements to store an initial configuration of $N = 16$ particles in a central cell of linear dimension $Lx = Ly = 6$.

```

SUB initial(t,ke,kecum,pecum,vcum,area)
DECLARE PUBLIC x(),y(),vx(),vy()
DECLARE PUBLIC N,Lx,Ly,dt,dt2
DECLARE DEF pbc
LET dt = 0.01
LET dt2 = dt*dt
LET response$ = ""
DO
    INPUT prompt "read data statements (d) or file (f)? ": start$
    IF (start$ = "d") or (start$ = "D") then
        LET response$ = "ok"
        LET N = 16
        LET Lx = 6
        LET Ly = 6
        DATA 1.09,0.98,-0.33,0.78,3.12,5.25,0.12,-1.19
        DATA 0.08,2.38,-0.08,-0.10,0.54,4.08,-1.94,-0.56
        DATA 2.52,4.39,0.75,0.34,3.03,2.94,1.70,-1.08
        DATA 4.25,3.01,0.84,0.47,0.89,3.11,-1.04,0.06
        DATA 2.76,0.31,1.64,1.36,3.14,1.91,0.38,-1.24
        DATA 0.23,5.71,-1.58,0.55,1.91,2.46,-1.55,-0.16
        DATA 4.77,0.96,-0.23,-0.83,5.10,4.63,-0.31,0.65
        DATA 4.97,5.88,1.18,1.48,3.90,0.20,0.46,-0.51
        FOR i = 1 to N
            READ x(i),y(i),vx(i),vy(i)
        NEXT i
    ELSE IF (start$ = "f") or (start$ = "F") then
        LET response$ = "ok"
        INPUT prompt "file name = ": file$
        OPEN #1: name file$, access input
        INPUT #1: N
        INPUT #1: Lx
        INPUT #1: Ly
        INPUT #1: heading$

```

```

FOR i = 1 to N
    INPUT #1: x(i),y(i)
NEXT i
INPUT #1: heading$
FOR i = 1 to N
    INPUT #1: vx(i),vy(i)
NEXT i
CLOSE #1
ELSE
    PRINT
    PRINT "d or f are the only acceptable responses."
END IF
LOOP until response$ = "ok"
CLEAR
LET ke = 0           ! kinetic energy
FOR i = 1 to N
    LET ke = ke + vx(i)*vx(i) + vy(i)*vy(i)
NEXT i
LET ke = 0.5*ke
LET area = Lx*Ly
LET t = 0           ! time
! initialize sums
LET kecum = 0
LET pecum = 0
LET vcum = 0
END SUB

```

It is good programming practice to have an appropriate response for any input. For this reason SUB `initial` checks that an appropriate response is given to the INPUT statement.

SUB `accel` finds the total force on each particle and uses Newton's third law to reduce the number of calculations by a factor of two. (In reduced units, the mass of a particle is unity and hence acceleration and force are equivalent.) FUNCTION `separation` ensures that the separation between particles is no greater than $L_x/2$ in the x direction and $L_y/2$ in the y direction. The force on a given particle is assumed to be due to all the other $N - 1$ particles. Because of the short range nature of the Lennard-Jones potential, we could truncate the force at a distance $r = r_c \approx 2.5\sigma$ and ignore the forces from particles whose distance is greater than r_c . The meaning of the quantity `virial` in SUB `accel` is discussed in Section 8.7.

```

SUB accel(pe,virial)
DECLARE PUBLIC x(),y(),ax(),ay()
DECLARE PUBLIC N,Lx,Ly
DECLARE DEF separation
FOR i = 1 to N
    LET ax(i) = 0
    LET ay(i) = 0
NEXT i

```

```

LET pe = 0
LET virial = 0
FOR i = 1 to N - 1           ! compute total force on particle i
    FOR j = i + 1 to N        ! due to particles j > i
        LET dx = separation(x(i) - x(j),Lx)
        LET dy = separation(y(i) - y(j),Ly)
        ! acceleration = force because mass = 1 in reduced units
        CALL force(dx,dy,fxij,fyij,pot)
        LET ax(i) = ax(i) + fxij
        LET ay(i) = ay(i) + fyij
        LET ax(j) = ax(j) - fxij ! Newton's third law
        LET ay(j) = ay(j) - fyij
        LET pe = pe + pot
        LET virial = virial + dx*fx + dy*fy
    NEXT j
NEXT i
END SUB

FUNCTION separation(ds,L)
    IF ds > 0.5*L then
        LET separation = ds - L
    ELSE IF ds < -0.5*L then
        LET separation = ds + L
    ELSE
        LET separation = ds
    END IF
END DEF

SUB force(dx,dy,fx,fy,pot)
    LET r2 = dx*dx + dy*dy
    LET rm2 = 1/r2
    LET rm6 = rm2*rm2*rm2
    LET f_over_r = 24*rm6*(2*rm6 - 1)*rm2
    LET fx = f_over_r*dx
    LET fy = f_over_r*dy
    LET pot = 4*(rm6*rm6 - rm6)
END SUB

```

The Verlet algorithm for the numerical solution of Newton's equations of motion is implemented in **SUB Verlet**. Note that the velocity is partially updated using the old acceleration. Then **SUB accel** is called to determine the acceleration using the new position and the velocity is updated again. **SUB Verlet** calls **FUNCTION pbc** which implements the periodic boundary conditions using **IF** statements.

```
SUB Verlet(t,ke,pe,virial)
```

```

DECLARE PUBLIC x(),y(),vx(),vy(),ax(),ay()
DECLARE PUBLIC N,Lx,Ly,dt,dt2
DECLARE DEF pbc
FOR i = 1 to N
    LET xnew = x(i) + vx(i)*dt + 0.5*ax(i)*dt2
    LET ynew = y(i) + vy(i)*dt + 0.5*ay(i)*dt2
    ! partially update velocity using old acceleration
    LET vx(i) = vx(i) + 0.5*ax(i)*dt
    LET vy(i) = vy(i) + 0.5*ay(i)*dt
    LET x(i) = pbc(xnew,Lx)
    LET y(i) = pbc(ynew,Ly)
NEXT i
CALL accel(pe,virial)           ! new acceleration
LET ke = 0
FOR i = 1 to N
    ! complete the update of the velocity using new acceleration
    LET vx(i) = vx(i) + 0.5*ax(i)*dt
    LET vy(i) = vy(i) + 0.5*ay(i)*dt
    LET ke = ke + vx(i)*vx(i) + vy(i)*vy(i)
NEXT i
LET ke = 0.5*ke
LET t = t + dt
END SUB

FUNCTION pbc(pos,L)
IF pos < 0 then
    LET pbc = pos + L
ELSE IF pos > L then
    LET pbc = pos - L
ELSE
    LET pbc = pos
END IF
END DEF

```

In SUB `set_up_windows` we divide the screen into two windows so that the trajectories are shown in a separate window. One advantage of the use of multiple windows is that the `CLEAR` statement erases the current window only, rather than the entire screen.

```

SUB set_up_windows(#1,#2)
DECLARE PUBLIC Lx,Ly
OPEN #1: screen 0,1,0.90,1.0 ! numerical output
OPEN #2: screen 0.02,1,0.02,0.90 ! particle trajectories
CALL compute_aspect_ratio(Lx,xwin,ywin)
SET WINDOW 0,xwin,0,ywin
BOX LINES 0,Lx,0,Ly
CALL headings(#1)

```

```

END SUB

SUB headings(#1)
  WINDOW #1
  SET CURSOR 1,1
  PRINT using "#####": "time steps";
  PRINT,
  PRINT using "##.##": "time";
  PRINT,
  PRINT using "--#.##": "energy";
  PRINT,
  PRINT using "#.##": "<T>";
  PRINT,
  PRINT using "#.##": "<P>"
END SUB

```

The kinetic energy is computed at each time step and its values are accumulated in **SUB show_output**. Although it is inefficient to compute the kinetic energy (and the virial) at every time step, we do so for simplicity.

```

SUB show_output(t,ke,pe,virial,kecum,vcum,ncum,area,#1)
  WINDOW #1
  DECLARE PUBLIC N,Lx,Ly
  SET CURSOR 2,1
  SET COLOR "black/white"
  LET ncum = ncum + 1
  PRINT using "#####": ncum;
  PRINT,
  PRINT using "##.##": t;      ! time
  PRINT,
  LET E = ke + pe           ! total energy
  PRINT using "--#.##": E;
  PRINT,
  LET kecum = kecum + ke
  LET vcum = vcum + virial
  LET mean_ke = kecum/ncum    ! still need to divide by N
  LET p = mean_ke + (0.5*vcum)/ncum ! mean pressure * area
  LET p = p/area
  PRINT using "#.##": mean_ke/N;   ! mean kinetic temperature
  PRINT,
  PRINT using "#.##": p;
END SUB

```

The trajectories of the individual particles are displayed in **SUB show_positions** by representing the position of a particle at each time step by a point. The trajectories eventually fill the screen, which can be cleared by pressing the letter ‘r’ (refresh). If the letter ‘n’ is pressed, the

particle positions are not shown on the screen and the program runs faster. Pressing ‘r’ restarts the screen updates. The program can be stopped by pressing the letter ‘s’ (stop).

```
SUB show_positions(flag$,#2)
DECLARE PUBLIC x(),y(),N,Lx,Ly
IF key input then
    GET KEY k
    IF k = ord("r") then
        WINDOW #2
        CLEAR
        SET COLOR "black"
        BOX LINES 0,Lx,0,Ly
        LET flag$ = ""
    ELSEIF k = ord("s") then
        LET flag$ = "stop"
    ELSEIF k = ord("n") then
        LET flag$ = "no_show"
    END IF
END IF
IF flag$ <> "no_show" then
    WINDOW #2
    SET COLOR "red"
    FOR i = 1 to N
        PLOT x(i),y(i)
    NEXT i
END IF
END SUB
```

Frequently, it is convenient to start a new run from the last configuration of a previous run. The final configuration is saved to a file in the following:

```
SUB save_config
DECLARE PUBLIC x(),y(),vx(),vy()
DECLARE PUBLIC N,Lx,Ly
INPUT prompt "file name of saved configuration = ": file$
OPEN #1: name file$,access output,create new
PRINT #1: N
PRINT #1: Lx
PRINT #1: Ly
PRINT #1: "x(i)","y(i)"
! comma added between outputs on the same line so that file
! can be read by True BASIC
FOR i = 1 to N
    PRINT #1, using "----.####, ----.#####": x(i),y(i)
NEXT i
PRINT #1: "vx(i)","vy(i)"
```

```

FOR i = 1 to N
    PRINT #1, using "----.####, ----.#####": vx(i),vy(i)
NEXT i
CLOSE #1
END SUB

```

In Problem 8.2 we use this preliminary version of Program `md` to simulate the approach of a system to equilibrium.

Problem 8.2. Approach to equilibrium

1. The initial configuration incorporated into the DATA statements in Program `md` corresponds to $N = 16$ particles interacting via the Lennard-Jones potential in a square cell of linear dimension $L = 6$. Check that the x and y coordinates of all the particles lies between 0 and 6. Set $\Delta t = 0.01$ and run the program to make sure that it is working properly. The total energy should be approximately conserved and the trajectories of all sixteen particles should be seen on the screen.
2. Suppose that at $t = 0$, the constraint that $0 \leq x \leq 6$ is removed and the particles can move freely in a rectangular cell with $L_x = 12$ and $L_y = 6$. Incorporate this change into your program and observe the trajectories of the particles. Does the system become more or less random as time increases?
3. Compute $n(t)$, the number of particles in the left half of the cell, and plot $n(t)$ as a function of t . What is the qualitative behavior of $n(t)$? Also compute the time average of $n(t)$, and plot it as a function of t . What is the mean number of particles on the left half after the system has reached equilibrium? Compare your qualitative results with the results you found in Problem 7.1. Would the approach to equilibrium be better defined if you did a molecular dynamics simulation with $N = 64$ particles?

Problem 8.3. Sensitivity to initial conditions

1. Consider the following initial condition corresponding to $N = 11$ particles moving in the same direction with the same velocity (see Figure 8.4). Choose $L_x = L_y = 10$ and $\Delta t = 0.01$.

```

FOR i = 1 to N
    LET x(i) = Lx/2
    LET y(i) = (i - 0.5)*Ly/N
    LET vx(i) = 1
    LET vy(i) = 0
NEXT i

```

Does the system eventually reach equilibrium? Why or why not?

2. Change the velocity of particle 6 so that $vx(6) = 0.99$ and $vy(6) = 0.01$. Is the behavior of the system qualitatively different than in part (a)? Does the system eventually reach equilibrium? Are the trajectories of the particles sensitive to the initial conditions? Explain why almost all initial states lead to the same qualitative behavior.

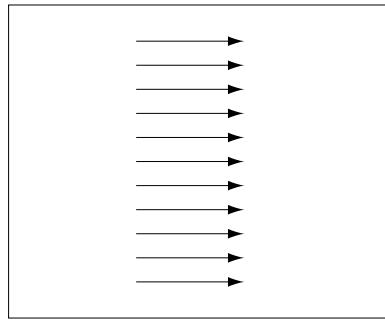


Figure 8.4: Example of a special initial condition; the arrows represent the magnitude and the direction of each particle's velocity.

3. Modify `Program md` so that the program runs for a predetermined time interval. Use the initial condition included in the `DATA` statements, and save the positions and velocities of all the particles at $t = 0.5$ in a file. Then consider the time reversed process, i.e., the motion that would occur if the direction of time were reversed. This reversal is equivalent to letting $\mathbf{v} \rightarrow -\mathbf{v}$ for all particles. Do the particles return to their original positions? What happens if you reverse the velocities at a later time? What happens if you choose a smaller value of Δt ?
4. Use the initial condition included in the `DATA` statements, but let $Lx = 12$. Save the positions and velocities of all the particles in a file at $t = 2.5$. Then consider the time-reversed process. Do the particles return to their original state? What happens if you reverse the velocities at $t = 5$ rather than at $t = 2.5$?
5. What can you conclude about the chaotic nature of the trajectories from your results? Are the computed trajectories the same as the “true” trajectories?

The trajectories generated in Problems 8.2 and 8.3 show the system in its greatest possible detail. From this `microscopic` viewpoint, the trajectories appear rather complex. The system can be described more simply by specifying its `macroscopic state`. For example, in Problem 8.2 we described the approach of the system to equilibrium by specifying n , the number of particles in the left half of the cell. Your observations of the macroscopic variable $n(t)$ should be consistent with the general properties of many body systems:

1. After the removal of an internal constraint, an isolated system changes in time from a “less random” to a “more random” state.
2. The equilibrium macroscopic state is characterized by relatively small fluctuations about a mean that is independent of time. A many particle system whose macroscopic state is independent of time is said to be in `equilibrium`. (The relative fluctuations become smaller as the number of particles becomes larger.)

In Problem 8.2(b) and (c) we found that the particles filled the box, and hence we were able to define a direction of time. Of course, this direction would be better defined if we considered

more particles. Note that there is nothing intrinsic in Newton's laws of motion that gives time a preferred direction.

Before we consider other macroscopic variables, we need to monitor the total energy and verify our claim that the Verlet algorithm maintains conservation of energy with a reasonable choice of Δt . We also introduce a check for momentum conservation.

Problem 8.4. Tests of the Verlet algorithm

1. One essential check of a molecular dynamics program is that the total energy be conserved to the desired accuracy. Use **Program md** and determine the value of Δt necessary for the total energy to be conserved to a given accuracy over a time interval of $t = 2$. One criterion is to compute $\Delta E_{\max}(t)$, the maximum of the difference $|E(t) - E(0)|$, over the time interval t , where $E(0)$ is the initial total energy, and $E(t)$ is the total energy at time t . Verify that $\Delta E_{\max}(t)$ decreases when Δt is made smaller for fixed t . If your program is working properly, then $\Delta E_{\max}(t)$ should decrease as approximately $(\Delta t)^2$.
2. Because of the use of periodic boundary conditions, all points in the central cell are equivalent and the system is translationally invariant. Explain why **Program md** should conserve the total linear momentum. Floating point error and the truncation error associated with a finite difference algorithm can cause the total linear momentum to drift. Programming errors also might be detected by checking for the conservation of momentum. Hence, it is a good idea to monitor the total linear momentum at regular intervals and reset the total momentum equal to zero if necessary. **SUB check_momentum**, listed in the following, should be called in **SUB initial** after the initial configuration is obtained and in the main loop of the program at regular intervals, e.g., every 100 – 1000 time steps. How well does **Program md** conserve the total linear momentum for $\Delta t = 0.01$?

```
SUB check_momentum
  DECLARE PUBLIC vx(),vy(),N
  LET vxsum = 0
  LET vysum = 0
  ! compute total center of mass velocity (momentum)
  FOR i = 1 to N
    LET vxsum = vxsum + vx(i)
    LET vysum = vysum + vy(i)
  NEXT i
  LET vxcm = vxsum/N
  LET vycm = vysum/N
  FOR i = 1 to N
    LET vx(i) = vx(i) - vxcm
    LET vy(i) = vy(i) - vycm
  NEXT i
END SUB
```

3. Another way of monitoring how well the program is conserving the total energy is to analyze the time series $E(t)$ using a least squares fit of $E(t)$ to a straight line. The slope of the line can be interpreted as the drift and the root mean square deviation from the straight line

can be interpreted as the noise (σ_y in the notation of Section 7.5). How does the drift and the noise depend on Δt for a fixed time interval t ? Most research applications conserve the energy to within 1 part in 10^3 or 10^4 or better over the duration of the run.

4. Consider one of the higher-order algorithms discussed in Appendix 5A for the solution of Newton's equations of motion. Can you choose a larger value of Δt to achieve the same degree of energy conservation that you found using the Verlet algorithm? Does the total energy fluctuate and/or eventually drift?

8.7 Thermodynamic Quantities

In the following we discuss how some of the macroscopic quantities of interest such as the temperature and the pressure can be related to time averages over the phase space trajectories of the particles. We then use our molecular dynamics program to explore the qualitative properties of gases and liquids.

The kinetic definition of the temperature follows from the equipartition theorem: each quadratic term in the energy of a classical system in equilibrium at temperature T has a mean value equal to $\frac{1}{2}kT$. Hence, we can define the temperature $T(t)$ at time t by the relation

$$N \frac{d}{2} kT(t) = K(t), \quad (8.5)$$

or

$$kT(t) = \frac{2}{d} \frac{K(t)}{N} = \frac{1}{dN} \sum_{i=1}^N m_i \mathbf{v}_i(t) \cdot \mathbf{v}_i(t). \quad (8.6)$$

K represents the total kinetic energy of the system, d is the spatial dimension, and the sum is over the N particles in the system. The mean temperature can be expressed as the time average of $T(t)$ over many configurations of the particles. For two dimensions ($d = 2$) and our choice of units, we write the mean temperature T as

$$T \equiv \bar{T} = \frac{1}{2N} \sum_{i=1}^N m_i \overline{\mathbf{v}_i(t) \cdot \mathbf{v}_i(t)}. \quad (\text{two dimensions}) \quad (8.7)$$

The relation (8.7) is an example of the relation of a macroscopic quantity, the mean temperature, to a time average over the trajectories of the particles.

Note that the relation (8.6) holds only if the momentum of the center of mass of the system is zero — we do not want the motion of the center of mass to change the temperature. In a laboratory system the walls of the container ensure that the center of mass motion is zero (if the mean momentum of the walls is zero). In our simulation, we impose the constraint that the center of mass momentum (in d directions) be zero. Consequently, the system has $dN - d$ independent velocity components rather than dN components, and we should replace (8.6) by

$$kT(t) = \frac{1}{dN - d} \sum_{i=1}^N m_i \mathbf{v}_i(t) \cdot \mathbf{v}_i(t). \quad (8.8)$$

The presence of the factor $d(N - 1)$ rather than dN in (8.8) is an example of a **finite size** correction which becomes unimportant for large N . We shall ignore this correction in the following.

Another macroscopic quantity of interest is the mean pressure of the system. The pressure is related to the force per unit area acting normal to an imaginary surface in the system. By Newton's second law, this force can be related to the momentum that crosses the surface per unit time. In general, this momentum flux has two contributions. The easiest contribution to understand is the one carried by the particles due to their motion. This contribution, equal to the pressure of an ideal gas, is derived in many texts (cf. Chapter 7 of Reif) using simple kinetic theory arguments and is given by $P_{\text{ideal}} = NkT/V$.

The other contribution to the momentum flux arises from the momentum transferred across the surface due to the forces between particles on different sides of the surface. The form of this contribution to the dynamical pressure is difficult to derive if periodic boundary conditions are used (cf. Haile). The instantaneous pressure at time t including both contributions to the momentum flux is given by

$$P(t) = \frac{N}{V}kT(t) + \frac{1}{dV} \sum_{i < j} \mathbf{r}_{ij}(t) \cdot \mathbf{F}_{ij}(t), \quad (8.9)$$

where $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$, and \mathbf{F}_{ij} is the force on particle i due to particle j .

The mean pressure $P \equiv \overline{P(t)}$ is found by computing a time average of the right-hand side of (8.9). The computed quantity of interest is not P , but the quantity

$$\frac{PV}{NkT} - 1 = \frac{1}{dNkT} \sum_{i < j} \overline{\mathbf{r}_{ij} \cdot \mathbf{F}_{ij}}. \quad (8.10)$$

In Program `md`, the right-hand side of (8.10), known as the **virial**, is computed in **SUB accel** and accumulated in the variable **vcum**. This quantity represents the correction to the ideal gas equation of state due to interactions between the particles.

The relation of information at the microscopic level to macroscopic quantities such as the temperature and pressure is one of the fundamental elements of statistical mechanics. In brief, molecular dynamics allows us to compute various time averages of the phase space trajectory over finite time intervals. The main practical question we must consider is whether our time intervals are sufficiently long to allow the system to explore phase space and yield meaningful averages. In equilibrium statistical mechanics, a time average is replaced by an **ensemble** average over all possible configurations. The quasi-ergodic hypothesis asserts the equivalence of these two averages if the same quantities are held fixed. In statistical mechanics, the ensemble of systems at fixed E , V , and N is called the microcanonical ensemble. Averages in this ensemble correspond to the time averages we find in molecular dynamics which are at fixed E , V and N . (Molecular dynamics also imposes an additional, but unimportant, constraint on the center of mass motion.) Ensemble averages are explored using Monte Carlo methods in Chapters ?? and ??.

The goal of the following problems is to explore some of the qualitative features of gases, liquids, and solids. Because we consider only small systems and relatively short run times, our results will only be suggestive.

Problem 8.5. Qualitative properties of a liquid and a gas

1. Use the initial configuration stored in the DATA statements in Program `md` for this problem. (If you do not want to type in the initial conditions, see Problem 8.11 for a discussion of how to generate your own.) For this initial condition $N = 16$ and $L_x = L_y = 6$. What is the reduced density? What is the initial energy of the system? Choose $\Delta t = 0.01$ and run the simulation for at least 500 time steps or $t = 5$. Compare your estimate for P with the value for an ideal gas.
2. Modify your program so that the instantaneous values of the temperature and pressure are not accumulated until the system has reached equilibrium. What is your criterion for equilibrium? One criterion is to compute the average values of T and P over finite time intervals and check that these averages do not drift with time.
3. One way of starting a simulation is to use the positions saved from an earlier run. The simplest way of obtaining an initial condition corresponding to a different density, but with the same value of N , is to rescale the positions of the particles and the linear dimensions of the cell. The following code shows one way to do so.

```

LET rscale = 0.95
FOR i = 1 to N
    LET x(i) = rscale*x(i)
    LET y(i) = rscale*y(i)
NEXT i
LET Lx = rscale*Lx
LET Ly = rscale*Ly
LET area = Lx*Ly

```

Incorporate the above code into your program in a separate subroutine. How do you expect P and T to change when the system is compressed? Use the same initial configuration as in part (a) and determine the mean temperature and pressure for the density $\rho = 16/(5.7)^2 \approx 0.49$ (`rescale = 0.95`). Compare your result for P to the ideal gas result. What do you think would happen if you choose a value of `rescale` that is much smaller? Save the final configuration of your simulation in a file and use it as the initial condition for another run with $\rho = 16/(0.95 \times 5.7)^2 \approx 0.55$. Determine T and P for this higher density and save the final configuration.

Problem 8.6. Distribution of speeds and velocities

1. Write a subroutine to compute the equilibrium probability $P(v)\Delta v$ that a particle has a speed between v and $v + \Delta v$. To do so, estimate the value of the maximum speed `vmax` that you need to bin. Choose bins of width $dv = vmax/nbin$, where `nbin` is the total number of bins. The following code illustrates one way of putting the speeds into their proper bins:

```

LET v2 = vx(i)*vx(i) + vy(i)*vy(i)
LET v = sqr(v2)
LET ibin = truncate(v/dv,0) + 1
IF ibin > nbin then LET ibin = nbin
LET prob(ibin) = prob(ibin) + 1

```

The array element `prob(ibin)` records the number of times the speed of a particle corresponds to `ibin`. Although it is inefficient, determine `prob(ibin)` after every time step for at least 100 time steps. Normalize `prob(ibin)` by dividing by the number of particles and by the number of time steps. Use the initial configuration stored in the `DATA` statements in `Program md` for this problem. Choose `nbin = 50`.

2. Plot the probability density $P(v)$ versus v . What is the qualitative form of $P(v)$? What is the most probable value of v ? What is the approximate width of $P(v)$? Compare your measured result to the theoretical form (in two dimensions)

$$P(v) dv = Ae^{-mv^2/2kT} v dv. \quad (8.11)$$

The form (8.11) of the distribution of speeds is known as the Maxwell-Boltzmann distribution.

3. Determine the probability density for the x and y components of the velocity. Make sure that you distinguish between positive and negative values. What is the most probable value for the x and y velocity components? What are their average values? Plot the probability densities $P(v_x)$ versus v_x and $P(v_y)$ versus v_y . Better results can be found by plotting the average $[P(v_x = w) + P(v_y = w)]/2$ versus w . What is the qualitative form of $P(\mathbf{v})$? Is it the same as the probability density for the speed?

Another useful thermal quantity is the `heat capacity` at constant volume defined by the relation $C_V = (\partial E / \partial T)_V$. C_V is an example of a linear response function, that is, the response of the temperature to energy in the form of heat added to the system.

One way to obtain C_V is to determine $T(E)$, the temperature as a function of E . (Remember that T is obtained as a function of E in the microcanonical ensemble.) The heat capacity is given by $\Delta E / \Delta T$ for two runs that have slightly different temperatures. This method is straightforward, but requires that simulations at different energies be done before the derivative can be estimated. An alternative way of estimating C_V from the fluctuations of the kinetic energy is discussed in Problem 8.7.

Problem 8.7. Energy dependence of the temperature and pressure

1. We have seen in Problem 8.5 that the total energy is determined by the initial conditions, and the temperature is a derived quantity found only after the system has reached thermal equilibrium. For this reason it is difficult to study the system at a particular temperature. The mean temperature can be changed to the desired temperature by rescaling the velocities of the system, but we have to be careful not to increase the velocities too quickly. Why? Choose the initial configuration of the system to be an equilibrium configuration from an earlier simulation for $Lx = Ly = 6$ and $N = 16$ and determine $T(E)$, the energy dependence of the mean temperature, in the range $T = 1.0$ to $T = 1.2$. Rescale the velocities by the desired factor. Is the equilibrium temperature increased to the desired value? If not, you need to rescale the velocities again. In general, the desired temperature is reached by a series of velocity rescalings over a sufficiently long time such that the system remains close to equilibrium during the rescaling.
2. Use your data for $T(E)$ found in part (a) to plot the total energy E as a function of T . Is T a monotonically increasing function of E ? Estimate the contribution to C_V from the potential

energy. What percentage of the contribution to the heat capacity is due to the potential energy? Why are accurate determinations of C_V difficult to achieve?

3. In our molecular dynamics simulations, the total energy is fixed, but the kinetic and potential energies can fluctuate. Another way of determining C_V is to relate it to the fluctuations of the kinetic energy. (In Chapter ?? we find that C_V is related to the fluctuations of the total energy in the constant T, V, N ensemble.) It can be shown that (cf. Ray and Graben)

$$\overline{T^2} - \overline{T}^2 = \frac{d}{2} N(k\overline{T})^2 \left[1 - \frac{dNk}{2C_V} \right]. \quad (8.12)$$

Note that the relation (8.12) reduces to the ideal gas result if $\overline{T^2} = \overline{T}^2$. Write a subroutine to determine C_V from (8.12) and compare your results with the determination of C_V in part (b). What are the advantages and disadvantages of determining C_V from the fluctuations of T compared to the method used in part (b)?

How can we generate a typical initial condition for a system of particles at high density? What happens if we simply place the particles at random in the central cell? The problem is that if the system is dense, some particles will be very close to one another and exert a very large repulsive force F on one another. Because the condition $(F/m)(\Delta t)^2 \ll \sigma$ must be satisfied for a finite difference method to be applicable, the random placement of particles is not practical. However, it is possible to use such an initial condition if a fictitious drag force proportional to the square of the velocity is introduced to equilibrate the system. The effect of such a force is to damp the velocity of those particles whose velocities become too large due to the large forces exerted on them.

In general, the best way of obtaining a configuration with the desired density and number of particles is to place the particles on the sites of a regular lattice. If the goal is to equilibrate the system at fluid densities, the symmetry of the lattice is not important. The initial velocities can be chosen at random according to the Maxwell-Boltzmann distribution you found in Problem 8.6c. However, as you have found, the velocities equilibrate very quickly, and we may simply choose each component of \mathbf{v}_i with uniform probability such that the desired temperature is obtained. Because the velocities are chosen at random, we need to use `SUB check_momentum` to ensure that the initial total momentum in the x and y direction is zero. After the system has come into partial equilibrium, you can increase the velocities of all the particles to reduce the time it takes to “melt” the system and reach the desired fluid state.

To simulate a solid we need to choose the shape of the central cell to be consistent with the symmetry of the solid phase of the system. This choice is necessary even though we have used periodic boundary conditions to minimize surface effects. If the cell does not correspond to the correct crystal structure, the particles cannot form a perfect crystal, and some of the particles will wander around in an endless search for their “correct” position. Consequently, a simulation of a small number of particles at high density and low temperature would lead to spurious results.

We know that the equilibrium structure of a crystalline solid at $T = 0$ is the configuration of lowest energy. In Problem 8.8 we compute the energy of a Lennard-Jones solid for both the square and triangular lattice structures (see Figure 8.5).

Problem 8.8. Ground state energy of two-dimensional lattices The nature of the triangular lattice can be seen from Figure 8.5. Each particle has six nearest neighbors. Although it is possible to

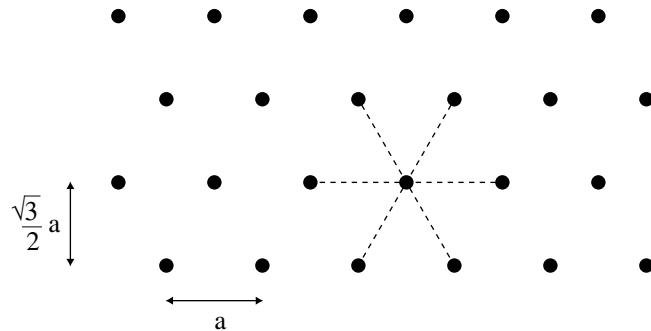


Figure 8.5: Each particle has six nearest neighbors in a triangular lattice.

choose the central cell of the triangular lattice to be a rhombus, it is more convenient to choose the cell to be rectangular. We take the linear dimensions of the cell to be L_x and $L_y = \sqrt{3}L_x/2$ respectively. For simplicity, we assume that \sqrt{N} is an integer so that the lattice spacings in the horizontal and vertical directions are $a_x = L_x/\sqrt{N}$ and $a_y = L_y/\sqrt{N}$, respectively. The lattice sites in each row are displaced by $\frac{1}{2}a_x$ from the preceding row. The following code generates a triangular lattice.

```

LET i = 0
FOR col = 1 to nx
    FOR row = 1 to ny
        LET i = i + 1
        IF (mod(row,2) = 0) then
            LET x(i) = (col - 0.75)*ax
        ELSE
            LET x(i) = (col - 0.25)*ax
        END IF
        LET y(i) = (row - 0.5)*ay
    NEXT row
NEXT col

```

Write a program to compute the potential energy per particle of a system of N particles interacting via the Lennard-Jones potential. Consider both the triangular and square lattices, and choose the linear dimension of the square lattice to be $L = \sqrt{L_x L_y}$, so that both lattices have the same density. Choose $N = 36$ and determine the energy for $L_x = 5$ and $L_x = 7$. What is the density of the system for each case? Do your results for E/N depend on the size of the lattice? Which lattice symmetry has a lower energy? Explain your results in terms of the ability of the triangular lattice to pack the particles closer together.

Problem 8.9. The solid state and melting

1. Choose $N = 16$, $L_x = 4$, and $L_y = \sqrt{3}L_x/2$, and place the particles on a triangular lattice. Give each particle zero initial velocity. What is the total energy of the system? Do a

simulation and measure the temperature and pressure as a function of time. Does the system remain a solid?

2. Give the particles a random velocity in the interval $[-0.5, +0.5]$. What is the total energy? Equilibrate the system and determine the mean temperature and pressure. Describe the trajectories of the particles. Are the particles localized? Is the system a solid? Save an equilibrium configuration for use in part (c).
3. Choose the initial configuration to be an equilibrium configuration from part (b), but increase the kinetic energy by a factor of two. What is the new total energy? Describe the qualitative behavior of the motion of the particles. What is the equilibrium temperature and pressure of the system? After equilibrium is reached, increase the temperature again by rescaling the velocities in the same way. Repeat this rescaling and measure $P(T)$ and $E(T)$ for several different temperatures.
4. Use your results from part (c) to plot $E(T) - E(0)$ and $P(T)$ as a function of T . Is the difference $E(T) - E(0)$ proportional to T ? What is the mean potential energy for a harmonic solid? What is its heat capacity?
5. Choose an equilibrium configuration from part (b) and decrease the density by rescaling L_x , L_y and the particle coordinates by a factor of 1.1. What is the nature of the trajectories? Decrease the density of the system until the system melts. What is your qualitative criterion for melting?

It is possible for a system to not be in equilibrium even though its properties do not change over a long time interval. For example, if we rapidly lower the temperature of a liquid below its freezing temperature, it is likely that the resulting state is not an equilibrium crystal, but rather a supercooled liquid that will eventually nucleate to a crystal. In general, we must carefully prepare our system so as to minimize the probability that the system becomes trapped in a **metastable state**. On the other hand, there is much interesting physics and many applications to material science in the kinetics of nonequilibrium systems as they evolve toward equilibrium.

Problem 8.10. Metastability

1. We can create a metastable state by placing the particles in a square cell whose shape is not consistent with the lowest energy state corresponding to a triangular lattice. Modify **SUB initial** so that the particle positions form a square lattice. If the initial velocities are set to zero, what happens when you run the program? Choose $N = 64$ and $L_x = L_y = 9$.
2. We can show that the system in part (a) is in a metastable state by giving the particles a small random initial velocity with **vmax** = 0.5. Does the square lattice immediately change? When do you begin to see local structure resembling a triangular lattice?
3. If time permits, repeat part (b) with **vmax** = 0.1.

8.8 Radial Distribution Function

We can gain more insight into the structure of a many body system by looking at how the positions of the particles are correlated with one another due to their interactions. The **radial**

distribution function $g(r)$ is the most common measure of this correlation and is defined as follows. Suppose that N particles are contained in a region of volume V with number density $\rho = N/V$. (In two and one dimensions, we replace V by the area and length respectively.) Choose one of the particles to be the origin. Then the mean number of other particles in the shell between \mathbf{r} and $\mathbf{r} + d\mathbf{r}$ is given by $\rho g(\mathbf{r}) d\mathbf{r}$, where the volume element $d\mathbf{r} = 4\pi r^2 dr$ ($d = 3$), $2\pi r dr$ ($d = 2$), or $2 dr$ ($d = 1$). If the interparticle interaction is spherically symmetric and the system is a gas or a liquid, then $g(\mathbf{r})$ depends only on the separation $r = |\mathbf{r}|$. The normalization condition for $g(r)$ is

$$\rho \int g(r) d\mathbf{r} = N - 1 \approx N. \quad (8.13)$$

Equation (8.13) implies that if we choose one particle as the origin and count all the other particles in the system, we obtain $N - 1$ particles. For an ideal gas, there are no correlations between the particles, and $g(r) = 1$ for all r . For the Lennard-Jones interaction, we expect that $g(r) \rightarrow 0$ as $r \rightarrow 0$, because the particles cannot penetrate one another. We also expect that $g(r) \rightarrow 1$ as $r \rightarrow \infty$, because the effect of one particle on another decreases as their separation increases.

The radial distribution function can be measured indirectly by elastic radiation scattering experiments, especially by the scattering of X-rays. Several thermodynamic properties also can be obtained from $g(r)$. Because $\rho g(r)$ can be interpreted as the local density about a given particle, the potential energy of interaction between this particle and all other particles between r and $r + dr$ is $u(r)\rho g(r) dr$, if we assume that only two-body interactions are present. The total potential energy is found by integrating over all values of r and multiplying by $N/2$. The factor of N is included because any of the N particles could be chosen as the particle at the origin, and the factor of $1/2$ is included so that each pair interaction is counted only once. The result is that the mean potential energy per particle can be expressed as

$$\frac{U}{N} = \frac{\rho}{2} \int g(r) u(r) dr. \quad (8.14)$$

It also can be shown that the relation (8.10) for the mean pressure can be rewritten in terms of $g(r)$ so that the equation of state can be expressed as

$$\frac{PV}{NkT} = 1 - \frac{\rho}{2dkT} \int g(r) r \frac{du(r)}{dr} dr. \quad (8.15)$$

To determine $g(r)$ for a particular configuration of particles, we first compute $n(r, \Delta r)$, the number of particles in a spherical (circular) shell of radius r and small, but nonzero width Δr , with the center of the shell centered about each particle. A subroutine for computing $n(r)$ is given in the following:

```
SUB compute_g(ncorrel)
  DECLARE PUBLIC x(),y()
  DECLARE PUBLIC N,Lx,Ly
  DECLARE PUBLIC gcum(),nbin,dr
  DECLARE DEF separation
  ! accumulate data for n(r)
  FOR i = 1 to N - 1
```

```

FOR j = i + 1 to N
    LET dx = separation(x(i) - x(j),Lx)
    LET dy = separation(y(i) - y(j),Ly)
    LET r2 = dx*dx + dy*dy
    LET r = sqr(r2)
    LET ibin = truncate(r/dr,0) + 1
    IF ibin <= nbin then
        LET gcum(ibin) = gcum(ibin) + 1
    END IF
NEXT j
NEXT i
LET ncorrel = ncorrel + 1      ! # times n(r) computed
END SUB

```

The results for $n(r)$ for different configurations are accumulated in the array `gcum`; the latter array is normalized in `SUB normalize_g` listed below. The use of periodic boundary conditions in `SUB compute_g` implies that the maximum separation between any two particles in the x and y direction is $Lx/2$ and $Ly/2$ respectively. Hence for a square cell, we can determine $g(r)$ only for $r \leq \frac{1}{2}L$.

To obtain $g(r)$ from $n(r)$, we note that for a given particle i , we consider only those particles whose j is greater than i (see `SUB compute_g`). Hence, there are a total of $\frac{1}{2}N(N-1)$ separations that are considered. In two dimensions we compute $n(r, \Delta r)$ for a circular shell whose area is $2\pi r \Delta r$. These considerations imply that $g(r)$ is related to $n(r)$ by

$$\rho g(r) = \frac{\overline{n(r, \Delta r)}}{\frac{1}{2}N 2\pi r \Delta r}. \quad (\text{two dimensions}) \quad (8.16)$$

Note the factor of $N/2$ in the denominator of (8.16). The following subroutine normalizes the array `gcum` and yields $g(r)$:

```

SUB normalize_g(ncorrel)
    DECLARE PUBLIC N,Lx,Ly
    DECLARE PUBLIC gcum(),dr
    LET density = N/(Lx*Ly)
    LET rmax = min(Lx/2,Ly/2)
    LET normalization = density*ncorrel*0.5*N
    LET bin = 1
    LET r = 0
    OPEN #2: name "gdata", access output,create new
    DO while r <= rmax
        LET area_shell = pi*((r + dr)^2 - r^2)
        LET g = gcum(bin)/(normalization*area_shell)
        PRINT r+dr/2,g
        PRINT #2: r+dr/2,g
        LET bin = bin + 1
        LET r = r + dr
    LOOP

```

```
CLOSE #2
END SUB
```

The shell thickness Δr needs to be sufficiently small so that the important features of $g(r)$ are found, but large enough so that each bin has a reasonable number of contributions. The value of Δr can be specified in **SUB initial**; a reasonable compromise choice for its magnitude is $dr = 0.025$.

Problem 8.11. The structure of $g(r)$ for a dense liquid and a solid

1. Incorporate **SUB compute_g** and **SUB normalize_g** into your molecular dynamics program and determine $g(r)$ for some of the same densities and temperatures that you have considered in previous problems. What are the qualitative features of $g(r)$?
2. Compute $g(r)$ for a system of $N = 64$ particles that are fixed on a triangular lattice with $L_x = 8$ and $L_y = \sqrt{3}L_x/2$. What is the density of the system? What is the nearest neighbor distance between sites? At what value of r does the first maximum of $g(r)$ occur? What is the next nearest distance between sites? At what value of r does the second maximum of $g(r)$ occur? Does your calculated $g(r)$ have any other relative maxima? If so, relate these maxima to the structure of the triangular lattice.
3. Use your molecular dynamics program to compute $g(r)$ for a dense fluid ($\rho > 0.6$, $T \approx 1.0$) using at least $N = 32$ particles. How many relative maxima can you observe? In what ways do they change as the density is increased? How does the behavior of $g(r)$ for a dense liquid compare to that of a dilute gas and a solid?

8.9 Hard disks

How can we understand the temperature and density dependence of the equation of state and the structure of a dense liquid? One way to gain more insight is to modify the interaction and see how the properties of the system change. In particular, we would like to understand the relative role of the repulsive and attractive parts of the interaction. For this reason, we consider an idealized system of hard disks for which the interaction $u(r)$ is purely repulsive:

$$u(r) = \begin{cases} +\infty, & r < \sigma \\ 0, & r \geq \sigma \end{cases} . \quad (8.17)$$

The length σ is the diameter of the hard disks (see Figure 8.6). In three dimensions the interaction (8.17) describes the interaction of hard spheres (billiard balls); in one dimension (8.17) describes the interaction of hard rods.

Because the interaction $u(r)$ between hard disks is a discontinuous function of r , the dynamics of hard disks is qualitatively different than it is for a continuous interaction such as the Lennard-Jones potential. For hard disks, the particles move in straight lines at constant speed between collisions and change their velocities instantaneously when a collision occurs. Hence the problem becomes finding the next collision and computing the change in the velocities of the colliding pair. We will see that the dynamics can be computed exactly in principle and is limited only by computer roundoff errors.

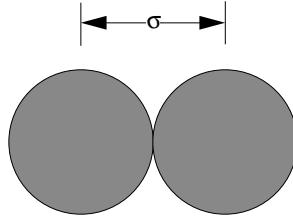


Figure 8.6: The closest distance between two hard disks is σ . The disks exert no force on one another unless they touch.

The dynamics of a system of hard disks can be treated as a sequence of two-body elastic collisions. The idea is to consider all pairs of particles i and j and to find the collision time t_{ij} for their next collision ignoring the presence of all other particles. In many cases, the particles will be going away from each other and the collision time is infinite. From the collection of collision times for all pairs of particles, we find the minimum collision time. We then move all particles forward in time until the collision occurs and calculate the postcollision velocities of the colliding pair.

We first determine the particle velocities after a collision. Consider a collision between particles 1 and 2. Let \mathbf{v}_1 and \mathbf{v}_2 be their velocities before the collision and \mathbf{v}'_1 and \mathbf{v}'_2 be their velocities after the collision. Because the particles have equal mass, it follows from conservation of energy and linear momentum that

$${v'_1}^2 + {v'_2}^2 = {v_1}^2 + {v_2}^2 \quad (8.18)$$

$$\mathbf{v}'_1 + \mathbf{v}'_2 = \mathbf{v}_1 + \mathbf{v}_2. \quad (8.19)$$

From (8.19) we have

$$\Delta\mathbf{v}_1 = \mathbf{v}'_1 - \mathbf{v}_1 = -(\mathbf{v}'_2 - \mathbf{v}_2) = -\Delta\mathbf{v}_2. \quad (8.20)$$

When two hard disks collide, the force is exerted along the line connecting their centers, $\mathbf{r}_{12} = \mathbf{r}_1 - \mathbf{r}_2$. Hence, the components of the velocities parallel to \mathbf{r}_{12} are exchanged, and the perpendicular components of the velocities are unchanged. It is convenient to write the velocity of particles 1 and 2 as a vector sum of its components parallel and perpendicular to the unit vector $\hat{\mathbf{r}}_{12} = \mathbf{r}_{12}/|\mathbf{r}_{12}|$. We write the velocity of particle 1 as:

$$\mathbf{v}_1 = \mathbf{v}_{1,\parallel} + \mathbf{v}_{1,\perp}, \quad (8.21)$$

where $\mathbf{v}_{1,\parallel} = (\mathbf{v}_1 \cdot \hat{\mathbf{r}}_{12})\hat{\mathbf{r}}_{12}$,

$$\mathbf{v}'_{1,\parallel} = \mathbf{v}_{2,\parallel} \quad \mathbf{v}'_{2,\parallel} = \mathbf{v}_{1,\parallel} \quad (8.22a)$$

and

$$\mathbf{v}'_{1,\perp} = \mathbf{v}_{1,\perp} \quad \mathbf{v}'_{2,\perp} = \mathbf{v}_{2,\perp}. \quad (8.22b)$$

Hence, we can write \mathbf{v}'_1 as

$$\begin{aligned}\mathbf{v}'_1 &= \mathbf{v}'_{1,\parallel} + \mathbf{v}'_{1,\perp} \\ &= \mathbf{v}_{2,\parallel} + \mathbf{v}_{1,\perp} \\ &= \mathbf{v}_{2,\parallel} - \mathbf{v}_{1,\parallel} + \mathbf{v}_{1,\parallel} + \mathbf{v}_{1,\perp} \\ &= [(\mathbf{v}_2 - \mathbf{v}_1) \cdot \hat{\mathbf{r}}_{12}] \hat{\mathbf{r}}_{12} + \mathbf{v}_1.\end{aligned}\quad (8.23)$$

The change in the velocity of particle 1 at a collision is given by

$$\Delta\mathbf{v}_1 = \mathbf{v}'_1 - \mathbf{v}_1 = -[(\mathbf{v}_1 - \mathbf{v}_2) \cdot \hat{\mathbf{r}}_{12}] \hat{\mathbf{r}}_{12} \quad (8.24)$$

or

$$\Delta\mathbf{v}_1 = -\Delta\mathbf{v}_2 = \left(\frac{\mathbf{r}_{12} b_{12}}{\sigma^2} \right)_{\text{contact}}, \quad (8.25)$$

where $b_{12} = \mathbf{v}_{12} \cdot \mathbf{r}_{12}$, $\mathbf{v}_{12} = \mathbf{v}_1 - \mathbf{v}_2$, and we have used the fact that $|\mathbf{r}_{12}| = \sigma$ at contact.

Problem 8.12. Velocity distribution of hard rods

Use (8.18) and (8.19) to show that $v'_1 = v_2$ and $v'_2 = v_1$ in one dimension, i.e., two colliding hard rods of equal mass exchange velocities. If you start a system of hard rods with velocities chosen from a uniform random distribution, will the velocity distribution approach the equilibrium Maxwell-Boltzmann distribution?

The most time consuming part of a hard disk dynamics program is computing the collision times of all pairs of particles. We now consider the criteria for a collision to occur. Consider disks 1 and 2 at positions \mathbf{r}_1 and \mathbf{r}_2 at $t = 0$. If they collide at a time t_{12} later, their centers will be separated by a distance σ :

$$|\mathbf{r}_1(t_{12}) - \mathbf{r}_2(t_{12})| = \sigma \quad (8.26)$$

During the time t_{12} , the disks move with constant velocities. Hence we have

$$\mathbf{r}_1(t_{12}) = \mathbf{r}_1(0) + \mathbf{v}_1(0) t_{12} \quad \text{and} \quad \mathbf{r}_2(t_{12}) = \mathbf{r}_2(0) + \mathbf{v}_2(0) t_{12}. \quad (8.27)$$

If we substitute (8.27) into (8.26), we find

$$[\mathbf{r}_{12} + \mathbf{v}_{12} t_{12}]^2 = \sigma^2 \quad (8.28)$$

or

$$t_{12} = \frac{-\mathbf{v}_{12} \cdot \mathbf{r}_{12} \pm \sqrt{(\mathbf{v}_{12} \cdot \mathbf{r}_{12})^2 - v_{12}^2(r_{12}^2 - \sigma^2)}}{v_{12}^2}. \quad (8.29)$$

Because $t_{12} > 0$ for a collision to occur, we see from (8.29) that the condition

$$\mathbf{v}_{12} \cdot \mathbf{r}_{12} < 0 \quad (8.30)$$

must be satisfied. That is if $\mathbf{v}_{12} \cdot \mathbf{r}_{12} > 0$, the particles are moving away from each other and there is no possibility of a collision.

If the condition (8.30) is satisfied, then the discriminant in (8.29) must satisfy the condition

$$(\mathbf{v}_{12} \cdot \mathbf{r}_{12})^2 - v_{12}^2(r_{12}^2 - \sigma^2) \geq 0. \quad (8.31)$$

If the condition (8.31) is satisfied, then the quadratic in (8.29) has two roots. The smaller root corresponds to the physically significant collision because the disks are impenetrable. Hence, the physically significant solution for the time of a collision t_{ij} for particles i and j is given by

$$t_{ij} = \frac{-b_{ij} - [b_{ij}^2 - v_{ij}^2(r_{ij}^2 - \sigma^2)]^{\frac{1}{2}}}{v_{ij}^2}. \quad (8.32)$$

Problem 8.13. Calculation of collision times

Use (8.32) and `SUB check_collision` listed in `Program hd` to write a program that determines the collision times (if any) of the following pairs of particles. It would be a good idea to draw the trajectories to confirm your results. Consider the three cases: $\mathbf{r}_1 = (2, 1)$, $\mathbf{v}_1 = (-1, -2)$, $\mathbf{r}_2 = (1, 3)$, $\mathbf{v}_2 = (1, 1)$; $\mathbf{r}_1 = (4, 3)$, $\mathbf{v}_1 = (2, -3)$, $\mathbf{r}_2 = (3, 1)$, $\mathbf{v}_2 = (-1, -1)$; and $\mathbf{r}_1 = (4, 2)$, $\mathbf{v}_1 = (-2, \frac{1}{2})$, $\mathbf{r}_2 = (3, 1)$, $\mathbf{v}_2 = (-1, 1)$. As usual, choose units so that $\sigma = 1$.

The main thermodynamic quantity of interest for hard disks is the mean pressure P . Because the forces act only when two disks are in contact, we have to modify the form of (8.10). We write $\mathbf{F}_{ij}(t) = \mathbf{I}_{ij} \delta(t - t_c)$, where t_c is the time at which the collision occurs. This form of \mathbf{F}_{ij} implies the force is nonzero only when there is a collision between i and j . (The delta function $\delta(t)$ is infinite for $t = 0$ and is zero otherwise; $\delta(t)$ is defined by its use in an integral as shown in (8.33).) This form of the force yields

$$\int_0^t \mathbf{I}_{ij} \delta(t' - t_c) dt' = \mathbf{I}_{ij} = m\Delta\mathbf{v}_{ij}, \quad (8.33)$$

where we have used Newton's second law and assumed that a single collision has occurred during the time interval t . The quantity $\Delta\mathbf{v}_{ij} = \mathbf{v}'_i - \mathbf{v}_i - (\mathbf{v}'_j - \mathbf{v}_j)$. If we explicitly include the time average to account for all collisions during a time interval t , we can write (8.10) as

$$\begin{aligned} \frac{PV}{NkT} - 1 &= \frac{1}{dNkT} \frac{1}{t} \sum_{ij} \int_0^t \mathbf{r}_{ij} \cdot \mathbf{I}_{ij} \delta(t' - t_c) dt' \\ &= \frac{1}{dNkT} \frac{1}{t} \sum_{c_{ij}} m\Delta\mathbf{v}_{ij} \cdot \mathbf{r}_{ij} \end{aligned} \quad (8.34)$$

The sum in (8.34) is over all collisions c_{ij} between disks i and j in the time interval t ; \mathbf{r}_{ij} is the vector between the centers of the disks at the time of a collision; the magnitude of \mathbf{r}_{ij} in (8.34) is σ .

Our hard disk dynamics program implements the following steps. Given the initial condition in `SUB initial`, we find the collision times and the collision partners for all pairs of particles i and j . We then

1. locate the minimum collision time t_{\min} ;

2. advance all particles using a straight line trajectory until the collision occurs, that is, displace particle i by $\mathbf{v}_i t_{\min}$ and update the collision time;
3. compute the postcollision velocities of the colliding pair i and j ;
4. calculate any quantities of interest and accumulate data;
5. update the collision partners of the colliding pair i and j and any other particles that were to collide with either i or j if i and j had not collided first;
6. repeat steps 1–5 indefinitely.

This steps are implemented in Program `hd` which is listed in the following:

```

PROGRAM hd
! dynamics of system of hard disks
! program based in part on Fortran program of Allen and Tildesley
PUBLIC x(100),y(100),vx(100),vy(100)
PUBLIC collision_time(100),partner(100)
PUBLIC N,Lx,Ly,t,timebig
LIBRARY "csgraphics"
CALL initial(vsum,rho,area)
CALL set_up_windows(ncolor,#1,#2)
CALL kinetic_energy(ke,#1)
CALL show_positions(flag$,#2)
LET temperature = ke/N
LET flag$ = ""
LET collisions = 0           ! number of collisions
DO
    CALL minimum_collision_time(i,j,tij)
    ! move particles forward and reduce collision times by tij
    CALL move(tij)
    LET t = t + tij
    LET collisions = collisions + 1
    CALL show_positions(flag$,#2)
    CALL contact(i,j,virial)      ! compute collision dynamics
    LET vsum = vsum + virial
    CALL show_output(t,collisions,temperature,vsum,rho,area,#1)
    ! reset collision list for relevant particles
    CALL reset_list(i,j)
LOOP until flag$ = "stop"
CALL save_config(#2)
END

```

The colliding pair and the next collision time are found in SUB `minimum_collision_time`, and all particles are moved forward in SUB `move` until contact occurs. The collision dynamics of the colliding pair is computed in SUB `contact`, where the contribution to the pressure virial also is

found. In SUB `reset_list` we update the collision partners of the colliding pair (i and j) and any other particles that were to collide with i and j if i and j had not collided first.

In SUB `initial` we initialize various variables and most importantly, call SUB `uplist` and SUB `check_collision` to compute the collision time for each particle assuming that no other particles are present. Note that the i th element in the array `collision_time` contains the minimum collision time for particle i with all particles j such that $j > i$. The array `partner(i)` stores the particle f the collision partner corresponding to this time with `partner(i)` always greater than i . The collision time for each particle is initially set to an arbitrarily large value, `timebig`, to account for the fact that at any given time, some particles have no collision partners.

```
SUB initial(vsum,rho,area)
DECLARE PUBLIC x(),y(),vx(),vy()
DECLARE PUBLIC N,Lx,Ly,t
DECLARE PUBLIC collision_time(),partner(),timebig
LET t = 0
INPUT prompt "read file (f) or lattice start (l) = ": start$
IF start$ = "f" or start$ = "F" then
    INPUT prompt "file name = ": file$
    OPEN #1: name file$, access input
    INPUT #1: N
    INPUT #1: Lx
    INPUT #1: Ly
    INPUT #1: heading$
    FOR i = 1 to N
        INPUT #1: x(i),y(i)
    NEXT i
    INPUT #1: heading$
    FOR i = 1 to N
        INPUT #1: vx(i),vy(i)
    NEXT i
    CLOSE #1
ELSE IF start$ = "l" or start$ = "L" then
    RANDOMIZE
    INPUT prompt "N = ": N      ! choose N so that sqr(N) an integer
    INPUT prompt "Lx = ": Lx
    INPUT prompt "Ly = ": Ly
    INPUT prompt "vmax = ": vmax
    LET nx = sqr(N)
    LET ny = nx
    IF nx >= Lx or nx >= Ly then
        PRINT "box too small"
        STOP
    END IF
    LET ax = Lx/nx           ! "lattice" spacing
    LET ay = Ly/ny
    LET i = 0
```

```

FOR col = 1 to nx
    FOR row = 1 to ny
        LET i = i + 1
        LET x(i) = (col - 0.5)*ax
        LET y(i) = (row - 0.5)*ay
        ! choose random positions and velocities
        LET vx(i) = (2*rnd - 1)*vmax
        LET vy(i) = (2*rnd - 1)*vmax
    NEXT row
NEXT col
END IF
CLEAR
CALL check_overlap           ! check if two disks overlap
CALL check_momentum
LET area = Lx*Ly
LET rho = N/area
LET timebig = 1.0e10
LET vsum = 0                 ! virial sum
FOR i = 1 to N
    LET partner(i) = N
NEXT i
LET collision_time(N) = timebig
! set up initial collision lists
FOR i = 1 to N
    CALL uplist(i)
NEXT i
END SUB

SUB uplist(i)
    DECLARE PUBLIC N,collision_time(),timebig
    ! look for collisions with particles j > i
    IF i = N then EXIT SUB
    LET collision_time(i) = timebig
    FOR j = i + 1 to N
        CALL check_collision(i,j)
    NEXT j
END SUB

```

SUB `check_collision` uses the relations (8.30) and (8.32) to determine whether particles i and j will collide and if so, the time t_{ij} until their collision. We consider not only the nearest periodic image of j , but also the images of j in the adjoining cells, and determine the minimum collision time using all of these images. As shown in Figure 8.7, it is possible for i to collide with an image of j that is not the image closest to i . For dense systems the probability that j is in a more distant cell is very small and these distant cells are ignored in the program.

```
SUB check_collision(i,j)
```

```

DECLARE PUBLIC x(),y(),vx(),vy()
DECLARE PUBLIC Lx,Ly,collision_time(),partner()
! consider collisions between i and periodic images of j
FOR xcell = -1 to 1
    FOR ycell = -1 to 1
        LET dx = x(i) - x(j) + xcell*Lx
        LET dy = y(i) - y(j) + ycell*Ly
        LET dvx = vx(i) - vx(j)
        LET dvy = vy(i) - vy(j)
        LET bij = dx*dvx + dy*dvy
        IF bij < 0 then
            LET r2 = dx*dx + dy*dy
            LET v2 = dvx*dvx + dvy*dvy
            LET descr = bij*bij - v2*(r2 - 1)
            IF descr > 0 then
                LET tij = (-bij - sqr(descr))/v2
                IF tij < collision_time(i) then
                    LET collision_time(i) = tij
                    LET partner(i) = j
                END IF
            END IF
        END IF
    NEXT ycell
NEXT xcell
END SUB

```

The minimum collision time t_{ij} is found in SUB `minimum_collision_time` and all particles are moved forward by this time in SUB `move`.

```

SUB minimum_collision_time(i,j,tij)
    DECLARE PUBLIC N,collision_time(),partner(),timebig
    ! locate minimum collision time
    LET tij = timebig
    FOR k = 1 to N
        IF collision_time(k) < tij then
            LET tij = collision_time(k)
            LET i = k
        END IF
    NEXT k
    LET j = partner(i)
END SUB

SUB move(tij)
    DECLARE PUBLIC x(),y(),vx(),vy()
    DECLARE PUBLIC N,collision_time()
    DECLARE PUBLIC Lx,Ly

```

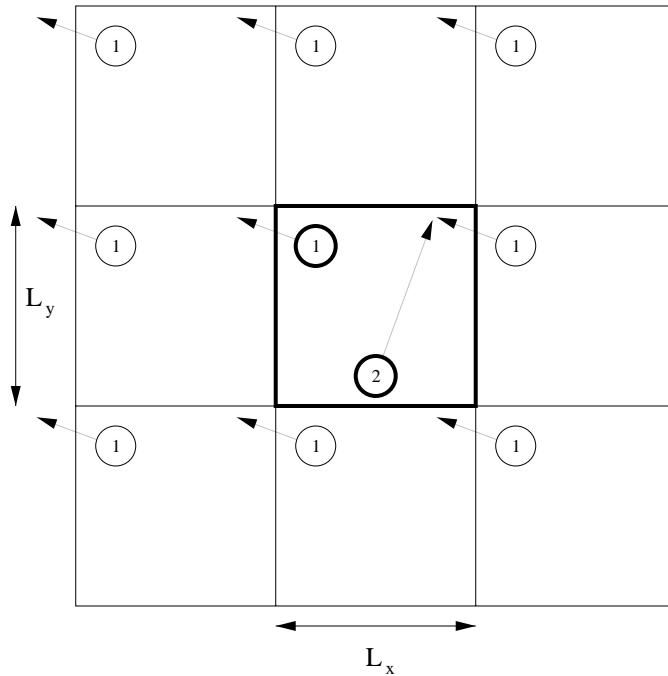


Figure 8.7: The positions and velocities of disks 1 and 2 are such that disk 1 collides with an image of disk 2 that is not the image closest to disk 1. The periodic images of disk 2 are not shown.

```

DECLARE DEF pbc
FOR k = 1 to N
    LET collision_time(k) = collision_time(k) - tij
    LET x(k) = x(k) + vx(k)*tij
    LET y(k) = y(k) + vy(k)*tij
    LET x(k) = pbc(x(k),Lx)
    LET y(k) = pbc(y(k),Ly)
NEXT k
END SUB

```

The function `pbc` allows for the possibility that a disk has moved further than the linear dimension of the central cell between a collision. We have written it as a separate function to emphasize its purpose.

```

DEF pbc(pos,L)
    LET pbc = mod(pos,L)
END DEF

```

The function `separation` is identical to the function listed in Program `md` and is not listed here.

The collision dynamics for the colliding particles i and j and the contribution of the collision to the virial are computed in SUB `contact`:

```
SUB contact(i,j,virial)
  DECLARE PUBLIC x(),y(),vx(),vy(),Lx,Ly
  DECLARE DEF separation
    ! compute collision dynamics for particles i and j at contact
  LET dx = separation(x(i) - x(j),Lx)
  LET dy = separation(y(i) - y(j),Ly)
  LET dvx = vx(i) - vx(j)
  LET dvy = vy(i) - vy(j)
  LET factor = dx*dvx + dy*dvy
  LET delvx = - factor*dx
  LET delvy = - factor*dy
  LET vx(i) = vx(i) + delvx
  LET vx(j) = vx(j) - delvx
  LET vy(i) = vy(i) + delvy
  LET vy(j) = vy(j) - delvy
  LET virial = delvx*dx + delvy*dy
END SUB
```

In SUB `reset_list` we find the new collision partners of particles i and j and those particles that were due to collide with i and j . Note that this update procedure must be done for particles whose labels are greater than i and j (SUB `uplist`) and for particles whose labels are less than i and j (SUB `downlist`).

```
SUB reset_list(i,j)
  DECLARE PUBLIC N,partner()
  ! reset collision list for relevant particles
  FOR k = 1 to N
    LET test = partner(k)
    IF k = i or test = i or k = j or test = j then
      CALL uplist(k)
    END IF
  NEXT k
  CALL downlist(i)
  CALL downlist(j)
END SUB

SUB downlist(j)
  DECLARE PUBLIC collision_time()
  ! look for collisions with particles  $i < j$ 
  IF j = 1 then EXIT SUB
  FOR i = 1 to j - 1
    CALL check_collision(i,j)
  NEXT i
END SUB
```

SUB `check_momentum` and SUB `set_up_windows` are identical to the subroutines listed in Program `md` and are not listed here. SUB `save_config` is similar to the one listed in Program `md` except that the disks are moved so that none of the disks are in contact before the configuration is saved. Because the main loop of the program is not completed until two disks are in contact, this move is made to remove the possibility that two disks might appear to overlap due to floating point error.

```
SUB save_config(#2)
  DECLARE PUBLIC x(),y(),vx(),vy()
  DECLARE PUBLIC N,Lx,Ly
  WINDOW #2
  SET COLOR "black"
  ! move particles away from collision for final configuration
  CALL minimum_collision_time(i,j,tij)
  CALL move(0.5*tij)
  SET CURSOR 1,1
  INPUT prompt "name of saved configuration = ": file$
  OPEN #1: name file$, access output, create new
  PRINT #1: N
  PRINT #1: Lx
  PRINT #1: Ly
  PRINT #1: "x(i)","y(i)"
  FOR i = 1 to N
    PRINT #1, using "----.#####, ----.#####": x(i),y(i)
  NEXT i
  PRINT #1: "vx(i)","vy(i)"
  FOR i = 1 to N
    PRINT #1, using "----.#####, ----.#####": vx(i),vy(i)
  NEXT i
  CLOSE #1
END SUB
```

As discussed in Problem 8.14, an important check on the calculated trajectories of a hard disk system is that no two disks overlap. SUB `check_overlap` tests for this condition.

```
SUB check_overlap
  DECLARE PUBLIC x(),y()
  DECLARE PUBLIC N,Lx,Ly
  DECLARE DEF separation
  LET tol = 1.0e-4
  FOR i = 1 to N - 1
    FOR j = i + 1 to N
      LET dx = separation(x(i) - x(j),Lx)
      LET dy = separation(y(i) - y(j),Ly)
      LET r2 = dx*dx + dy*dy
      IF r2 < 1 then
        LET r = sqr(r2)
```

```

        IF (1 - r) > tol then
            PRINT "particles ";i;" and ";j;"overlap"
            STOP
        END IF
    END IF
NEXT j
NEXT i
END SUB

```

The remaining output subroutines are similar to those in Program `md`, but are listed below for completeness.

```

SUB headings(#1)
    WINDOW #1
    SET CURSOR 1,1
    PRINT using "#####": "collisions";
    PRINT,
    PRINT using "#####": "time";
    PRINT,
    PRINT using "#####.###": "<P>";
    PRINT,
    PRINT using "###.###": "T";
    PRINT,
END SUB

SUB show_output(t,collisions,temperature,vsum,rho,area,#1)
    WINDOW #1
    SET CURSOR 2,1
    SET COLOR "black/white"
    PRINT using "#####": collisions;
    PRINT,
    PRINT using "###.###": t;
    PRINT,
    LET mean_virial = vsum/(2*t)
    LET mean_pressure = rho*temperature + mean_virial/area
    PRINT using "###.###": mean_pressure;
END SUB

SUB show_positions(flag$,#2)
    DECLARE PUBLIC x(),y(),N,Lx,Ly
    IF key input then
        GET KEY k
        IF k = ord("r") then
            WINDOW #2
            CLEAR
            SET COLOR "black"

```

```

    BOX LINES 0,Lx,0,Ly
    LET flag$ = ""
ELSEIF k = ord("s") then
    LET flag$ = "stop"
ELSEIF k = ord("n") then
    LET flag$ = "no_show"
END IF
END IF
IF flag$ <> "no_show" then
    SET COLOR "red"
    WINDOW #2
    FOR i = 1 to N
        PLOT x(i),y(i)
    NEXT i
END IF
END SUB

```

SUB `show_disks` can be substituted for SUB `show_positions` to represent the positions of the hard disks as circles rather than as points.

```

SUB show_disks(ncolor,flag$,#2)
    DECLARE PUBLIC x(),y(),N,Lx,Ly
    WINDOW #2
    IF key input then
        GET KEY k
        IF k = ord("r") then
            CLEAR
            BOX LINES 0,Lx,0,Ly
            LET flag$ = ""
        ELSEIF k = ord("s") then
            LET flag$ = "stop"
        ELSEIF k = ord("n") then
            LET flag$ = "no_show"
        END IF
    END IF
    IF flag$ <> "no_show" then
        LET ncolor = mod(ncolor,6) + 1
        IF ncolor = 1 then
            CLEAR
            BOX LINES 0,Lx,0,Ly
        END IF
        SET COLOR MIX(ncolor) rnd,rnd,rnd
        SET COLOR ncolor
        FOR i = 1 to N
            BOX CIRCLE x(i)-0.5,x(i)+0.5,y(i)-0.5,y(i)+0.5
        NEXT i
    END IF

```

```
END IF
END SUB
```

Finally, we list SUB `kinetic_energy` which is needed to compute the kinetic temperature:

```
SUB kinetic_energy(ke,#1)
DECLARE PUBLIC vx(),vy()
DECLARE PUBLIC N
WINDOW #1
LET ke = 0
FOR i = 1 to N
    LET ke = ke + vx(i)*vx(i) + vy(i)*vy(i)
NEXT i
LET ke = 0.5*ke
SET CURSOR 2,1
PRINT,,,
PRINT using "##.###": ke/N;
PRINT,
END SUB
```

Problem 8.14. Initial tests of Program `hd`

1. Because even a small error in computing the trajectories of the disks will eventually lead to their overlap and hence to a fatal error, it is necessary to test Program `hd` carefully. For simplicity, start from a lattice configuration. The most important test of the program is to monitor the computed positions of the hard disks at regular intervals for overlaps. If the distance between the centers of any two hard disks is less than unity (distances are measured in units of σ), there must be a serious error in the program. To check for the overlap of hard disks, include SUB `overlap` in the main loop of Program `hd` while you are testing the program.
2. The temperature for a system of hard disks is constant and can be defined as in (8.7). Why does the temperature not fluctuate as it does for a system of particles interacting with a continuous potential? The constancy of the temperature can be used as another check on your program. What is the effect of increasing all the velocities by a factor of two? What is the natural unit of time? Explain why the state of the system is determined only by the density and not by the temperature.
3. Use Program `hd` to generate equilibrium configurations of a system of $N = 16$ disks in a square cell of linear dimension $L = 6$. Suppose that at $t = 0$, the constraint that $0 \leq x \leq 6$ is removed, and the disks are allowed to move in a rectangular cell with $L_x = 12$ and $L_y = 6$. Does the system become more or less random? What is the qualitative nature of the time dependence of $n(t)$, the number of disks on the left half of the cell?
4. Modify your program so that averages are not computed until the system is in equilibrium. For simplicity, use the initial positions in the DATA statements in Program `md` as the initial condition. Compute the virial (8.34) and make a rough estimate of the error in your determination of the mean pressure due to statistical fluctuations.

5. Modify your program so that you can compute the velocity and speed distributions and verify that the computed distributions have the desired forms.

Problem 8.15. Static properties of hard disks

1. As we have seen in Section 8.7, a very time consuming part of the simulation is equilibrating the system from an arbitrary initial configuration. One way to obtain a set of initial positions is to add the hard disks sequentially with random positions and reject an additional hard disk if it overlaps any disks already present. Although this method is very inefficient at high densities, try it so that you will have a better idea of how difficult it is to obtain a high density configuration in this way. A much better method is to place the disks on the sites of a lattice.
2. We first consider the dependence of the mean pressure P on the density ρ . Is P a monotonically increasing function of ρ ? Is a system of hard disks always a fluid or is there a fluid to solid transition at higher densities? We will not be able to find definitive answers to these questions for $N = 16$. However, many simulations in the 1960's and 70's were done for systems of $N = 108$ hard disks and the largest simulations were for several hundred particles.
3. Compute the radial distribution function $g(r)$ for the same densities as you considered for the Lennard-Jones interaction. Compare the qualitative behavior of $g(r)$ for the two interactions. On the basis of your results, which part of the Lennard-Jones interaction plays the dominant role in determining the structure of a dense Lennard-Jones liquid?
4. The largest number of hard disks that can be placed into a fixed volume defines the maximum density. What is the maximum density if the disks are placed on a square lattice? What is the maximum density if the disks are placed on a triangular lattice? Suppose that the initial condition is chosen to be a square lattice with $N = 100$ and $L = 11$ so that each particle has four nearest neighbors initially. What is the qualitative nature of the system after several hundred collisions have occurred? Do most particles still have four nearest neighbors or are there regions where most particles have six neighbors?

In Problem 8.16 we consider two physical quantities associated with the dynamics of a system of hard disks, namely the mean free time and the mean free path, quantities that are discussed in texts on kinetic theory (cf. Reif).

Problem 8.16. Mean free path and collision time

1. Program `hd` provides the information needed to determine the mean free time t_c , i.e., the average time a particle travels between collisions. For example, suppose we know that 40 collisions occurred in a time $t = 2.5$ for a system of $N = 16$ disks. Because two particles are involved in each collision, there was an average of $80/16$ collisions per particle. Hence $t_c = 2.5/(80/16) = 0.5$. Write a subroutine to compute t_c and determine t_c as a function of ρ .
2. Write a subroutine to determine the distribution of times between collisions. What is the qualitative form of the distribution? How does the width of this distribution depend on ρ ?

3. The mean free path ℓ is the mean distance a particle travels between collisions. Is ℓ simply related to t_c by the relation $\ell = \bar{v}t_c$, where $\bar{v} = \sqrt{\langle v^2 \rangle}$? Write a subroutine to compute the mean free path of the particles. Note that the displacement of particle i during the time t is $v_i t$, where v_i is the speed of particle i .

8.10 Dynamical Properties

The mean free time and the mean free path are well defined for hard disks for which the meaning of a collision is clear. As discussed in texts on kinetic theory (cf. Reif), both quantities are related to the transport properties of a dilute gas. However, the concept of a collision is not well-defined for systems with a continuous interaction such as the Lennard-Jones potential. In the following, we take a more general approach to the dynamics of a many body system and discuss how the transport of particles in a system near equilibrium is related to the **equilibrium** properties of the system.

Suppose that we tag a certain fraction of the particles in our system (hem blue), and let $n(\mathbf{r}, t)$ be the mean number density of the tagged particles, that is, $n(\mathbf{r}, t)d\mathbf{r}$ is the mean number of particles with positions in the region $d\mathbf{r}$ about \mathbf{r} at time t . In equilibrium, we expect that the tagged particles would be distributed uniformly and hence $n(\mathbf{r}, t)$ would be independent of \mathbf{r} and t . Suppose however, that we start with a nonuniform distribution of tagged particles and ask how $n(\mathbf{r}, t)$ changes with \mathbf{r} and t to make the density of tagged particles more uniform. We expect that \mathbf{J} , the flux of tagged particles, is proportional to the density gradient of the tagged particles. In one dimension we write

$$J_x = -D \frac{\partial n(x, t)}{\partial x}. \quad (8.35a)$$

More generally, we have that

$$\mathbf{J} = -D \nabla n(\mathbf{r}, t). \quad (8.35b)$$

The empirical relation (8.35) is known as Fick's law and expresses the fact that the flow of tagged particles acts to equalize the density. The quantity D in (8.35) is known as the **self-diffusion coefficient**. If we combine (8.35) with the statement that the number of tagged particles is conserved:

$$\frac{\partial n(\mathbf{r}, t)}{\partial t} + \nabla \cdot \mathbf{J}(\mathbf{r}, t) = 0, \quad (8.36)$$

we obtain the diffusion equation:

$$\frac{\partial n(\mathbf{r}, t)}{\partial t} = D \nabla^2 n(\mathbf{r}, t). \quad (8.37)$$

As the above discussion implies, we usually think of the transport of particles (and the transport of energy and momentum and other quantities) in the context of nonequilibrium situations. However, as we have already seen in our simulations, the particles in an equilibrium system are in continuous motion and hence continuously create local density fluctuations. We expect that these spontaneous density fluctuations behave in the same way as the density fluctuations that

are created by weak external perturbations. Hence, we also expect (8.37) to apply to a system in equilibrium.

Consider the trajectory of a particular particle, e.g., particle 1, in equilibrium. At some arbitrarily chosen time $t = 0$, its position is $\mathbf{r}_1(0)$. At a later time t , its displacement is $\mathbf{r}_1(t) - \mathbf{r}_1(0)$. If there were no net force on the particle during this time interval, then $\mathbf{r}_1(t) - \mathbf{r}_1(0)$ would increase linearly with t . However, a particle in a fluid undergoes many collisions and on the average its net displacement would be zero. A more interesting quantity is the mean square displacement defined as

$$\overline{R_1^2(t)} = \overline{[\mathbf{r}_1(t) - \mathbf{r}_1(0)]^2}. \quad (8.38)$$

The average in (8.38) is over all possible choices of the time origin. If the system is in equilibrium, the choice of $t = 0$ is arbitrary, and $\overline{R_1^2(t)}$ depends only on the time difference t . We have seen in Appendix 7A that the diffusion equation (8.37) implies that the t dependence of $\overline{R^2(t)}$ is given by

$$\overline{R^2(t)} = 2dDt, \quad (t \rightarrow \infty) \quad (8.39)$$

where d is the spatial dimension. We have omitted the subscript because the average behavior of all the particles is the same.

The relation (8.39) relates the macroscopic transport coefficient D to a microscopic quantity, $\overline{R^2(t)}$, and gives us a straightforward way of computing D for an equilibrium system. The easiest way of computing $\overline{R^2(t)}$ is to write the position of a particle at regular time intervals into a file. We later can use a separate program to read the data file and compute $\overline{R^2(t)}$. Of course, we would find much better results if we average over all particles.

To understand the procedure for computing $\overline{R^2(t)}$, we consider a simple example. Suppose that the position of a particle in a one-dimensional system is given by $x(t = 0) = 1.65$, $x(t = 1) = 1.62$, $x(t = 2) = 1.84$, and $x(t = 3) = 2.22$. If we average over all possible time origins, we obtain

$$\begin{aligned} \overline{R^2(t=1)} &= \frac{1}{3}[(x(1) - x(0))^2 + (x(2) - x(1))^2 + (x(3) - x(2))^2] \\ &= \frac{1}{3}[0.0009 + 0.0484 + 0.1444] = 0.0646 \\ \overline{R^2(t=2)} &= \frac{1}{2}[(x(2) - x(0))^2 + (x(3) - x(1))^2] \\ &= \frac{1}{2}[0.0361 + 0.36] = 0.1981 \\ \overline{R^2(t=3)} &= (x(3) - x(0))^2 = 0.3249 \end{aligned}$$

Note that there are fewer combinations of the positions as the time difference increases. Program R2, listed in the following, reads a data file consisting of the x and y coordinates of a particle and computes $\overline{R^2(t)}$ by averaging over all possible choices of the time origin.

```
PROGRAM R2
! compute mean square displacement of one particle
! by averaging over all possible origins
```

```

DIM x(1000),y(1000),R2cum(20),norm(20)
CALL initial(Lx,Ly,R2cum(),ndiff)
CALL read_data(x(),y(),ndata)
CALL displacements(x(),y(),Lx,Ly,R2cum(),norm(),ndata,ndiff)
CALL normalization(ndiff,R2cum(),norm())
END

SUB initial(Lx,Ly,R2cum(),ndiff)
  LET Lx = 5
  LET Ly = 5
  LET ndiff = 10           ! maximum time difference
  FOR idiff = 1 to ndiff
    LET R2cum(idiff) = 0
  NEXT idiff
END SUB

SUB read_data(x(),y(),ndata)
  ! read file for position of particle at regular intervals
  OPEN #1: name "xy.dat",access input
  LET t = 0
  DO while more #1
    LET t = t + 1
    INPUT #1: x(t),y(t)
  LOOP
  CLOSE #1
  LET ndata = t           ! # of data points
END SUB

SUB displacements(x(),y(),Lx,Ly,R2cum(),norm(),ndata,ndiff)
  DECLARE DEF separation      ! function same as in Program md
  FOR idiff = 1 to ndiff
    FOR i = 1 to ndata - idiff
      LET dx = separation(x(i+idiff) - x(i),Lx)
      LET dy = separation(y(i+idiff) - y(i),Lx)
      LET R2cum(idiff) = R2cum(idiff) + dx*dx + dy*dy
      LET norm(idiff) = norm(idiff) + 1
    NEXT i
  NEXT idiff
END SUB

SUB normalization(ndiff,R2cum(),norm())
  PRINT "time difference"," <R2>"
  PRINT
  FOR idiff = 1 to ndiff
    IF R2cum(idiff) > 0 then
      LET R2bar = R2cum(idiff)/norm(idiff)

```

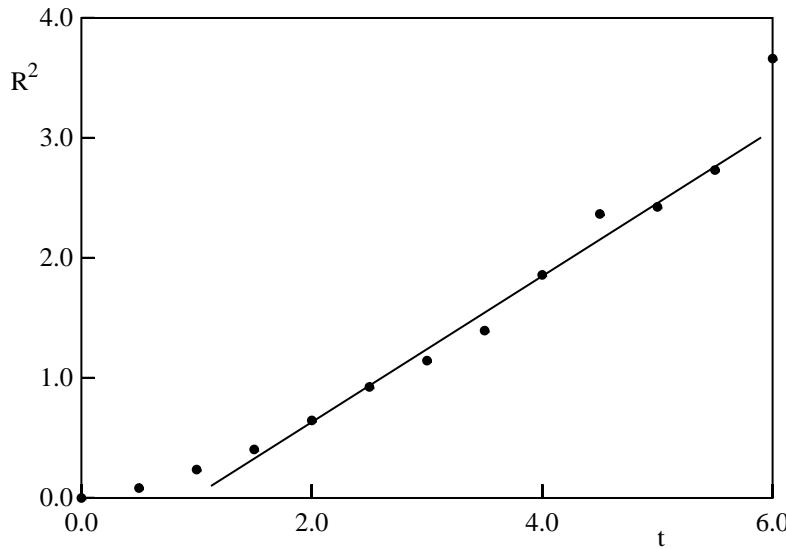


Figure 8.8: The time dependence of the mean square displacement $\overline{R^2(t)}$ for one particle in a two-dimensional Lennard-Jones system with $N = 16$, $L = 5$, and $E = 5.8115$. The position of a particle was saved at intervals of 0.5. Much better results can be obtained by averaging over all particles and over a longer run. The least squares fit was made between $t = 1.5$ and $t = 5.5$. As expected, this fit does not pass through the origin. The slope of the fit is 0.61.

```

PRINT idiff,R2bar
END IF
NEXT idiff
END SUB

```

We show our results for $\overline{R^2(t)}$ for a system of Lennard-Jones particles in Figure 8.8. Note that $\overline{R^2(t)}$ increases approximately linearly with t with a slope of roughly 0.61. From (8.39) the corresponding self-diffusion coefficient is $D = 0.61/4 \approx 0.15$. In Problem 8.17 we use Program R2 to compute the self-diffusion coefficient. An alternative way of computing D is discussed in Project 8.19.

Problem 8.17. The self-diffusion coefficient

1. Use Program md or Program hd and visually follow the motion of a particular particle by “tagging” it, e.g., by drawing its path with a different color. Describe its motion qualitatively.
2. Modify Program md or Program hd so that the coordinates of a particular particle are saved at regular intervals. The desired time interval needs to be determined empirically. If we save the coordinates too often, the data file will be too large, and we will waste time writing the data to a file. If we do not save the positions often enough, we lose information. Because the time step Δt must be small compared to any interesting time scale, we know that the

time interval for saving the positions must be at least a factor of ten greater than Δt . A good first guess is to choose the time interval to be the order of 10–50 time steps. The easiest procedure for hard disks is to save the positions at intervals measured in terms of the number of collisions. If we average over a sufficient number of collisions, we can find the relation between the elapsed time and the number of collisions.

3. Compute $\overline{R^2(t)}$ for conditions that correspond to a dense fluid. Does $\overline{R^2(t)}$ increase as t^2 as for a free particle or more slowly? Does $\overline{R^2(t)}$ increase linearly with t for longer times? What is the maximum value of $\overline{R^2(t)}$ that you find? Why does the use of periodic boundary conditions imply that there is an upper bound to the maximum time difference t we can consider when computing $\overline{R^2(t)}$?
4. Use the relation (8.39) to estimate the magnitude of D from the slope of $\overline{R^2(t)}$. Obtain D for several different temperatures and densities. (A careful study of $\overline{R^2(t)}$ for much larger systems and much longer times would show that $\overline{R^2(t)}$ is not proportional to t in two dimensions. Instead $\overline{R(t)^2}$ has a term proportional to $t \log t$, which dominates the linear t term if t is sufficiently large. However, we will not be able to observe the effects of this logarithmic term, and we can interpret our results for $\overline{R^2(t)}$ in terms of an “effective” diffusion coefficient. No such problem exists for three dimensions.)
5. Compute $\overline{R^2(t)}$ for an equilibrium configuration corresponding to a harmonic solid. What is the qualitative behavior of $\overline{R^2(t)}$?
6. Compute $\overline{R^2(t)}$ for an equilibrium configuration corresponding to a dilute gas. Is $\overline{R^2(t)}$ proportional to t for small times? Can we consider the particles to diffuse over short time intervals?

Another physically important single particle property is the **velocity autocorrelation function** $\psi(t)$. Suppose that particle i has velocity \mathbf{v}_i at time t_1 . If there were no net force on particle i , its velocity would remain constant. However, the interactions with other particles in the fluid will change the particle’s velocity, and we expect that after several collisions, its velocity will not be strongly correlated with its velocity at an earlier time. We define $\psi(t)$ as

$$\psi(t) = \frac{1}{v_0^2} \overline{\mathbf{v}_i(t_2) \cdot \mathbf{v}_i(t_1)}, \quad (8.40)$$

where $v_0^2 = \overline{\mathbf{v}_i(0) \cdot \mathbf{v}_i(0)} = dkT/m$ and $t = t_2 - t_1$. As in our discussion of the mean square displacement, the average in (8.40) is over all possible time origins. We have defined $\psi(t)$ such that $\psi(t=0) = 1$. For large time differences $t_2 - t_1$, we expect $\mathbf{v}_i(t_2)$ to be independent of $\mathbf{v}_i(t_1)$, and hence $\psi(t) \rightarrow 0$ for $t \rightarrow \infty$. (Note that we have implicitly assumed that $\overline{\mathbf{v}_i(t)} = 0$.) It can be shown that the self-diffusion coefficient defined by (8.39) can be related to an integral of $\psi(t)$:

$$D = v_0^2 \int_0^\infty \psi(t) dt. \quad (8.41)$$

Other transport coefficients such as the shear viscosity and the thermal conductivity can also be expressed as a time integral over a corresponding autocorrelation function. The qualitative properties of the velocity autocorrelation function are explored in Problem 8.18.

**Problem 8.18.* The velocity autocorrelation function

1. Modify your molecular dynamics program so that the velocity of a particular particle is saved to a file at regular time intervals. Then modify **Program R2** so that you can compute $\psi(t)$. The following code might be useful.

```
FOR idiff = 1 to ndiff
    FOR i = 1 to nmax - idiff
        LET psi(idiff) = psi(idiff) + vx(i + idiff)*vx(i)
        LET psi(idiff) = psi(idiff) + vy(i + idiff)*vy(i)
        LET norm(idiff) = norm(idiff) + 1
    NEXT i
NEXT idiff
```

Compute $\psi(t)$ for the same equilibrium configurations as in Problem 8.17c. Plot $\psi(t)$ versus t and describe its qualitative behavior. Estimate D from the relation (8.41). (To estimate the integral of $\psi(t)$, add your results for $\psi(t)$ at the different values of t and multiply the sum by the time difference between successive values of t .) How does your result for D compare to the determination using (8.39)?

2. Assume that $\psi(t)$ satisfies the form $\psi(t) = e^{-t/t_r}$ for all t . Substitute this form for $\psi(t)$ into (8.41) and determine the relationship between D and the **relaxation time** t_r . Plot the natural logarithm of $\psi(t)$ versus t and estimate t_r from the linear behavior of $\ln \psi(t)$. (At very long times, $\psi(t)$ exhibits slower than exponential decay. This “long-time tail” is due to hydrodynamic effects.) Use your derived relationship between D and t_r to find D . Compare your estimates for D found from the slope of $\overline{R^2(t)}$, the relation (8.41), and the estimate of t_r . Are these estimates consistent?
3. Increase the density by 50% and compute $\psi(t)$. What is the qualitative behavior of $\psi(t)$? What is the implication of the fact that $\psi(t)$ becomes negative after a relatively short time?
4. Compute $\psi(t)$ for an equilibrium solid. Plot $\psi(t)$ versus t and describe its qualitative behavior. Explain your results in terms of the oscillatory motion of the particles about their lattice sites.
5. Contrast the behavior of the mean square displacement, the velocity autocorrelation function, and the radial distribution function in the solid and fluid phases and explain how these quantities can be used to indicate the nature of the phase.
6. Modify your program so that $\overline{R^2(t)}$ and $\psi(t)$ are averaged over all particles.

8.11 Extensions

The primary goals of this chapter have been to introduce the method of molecular dynamics and some of the concepts of statistical mechanics and kinetic theory. Although we found that simulations of systems as small as sixteen particles show some of the qualitative properties of macroscopic systems, we would need to simulate larger systems to make quantitative conclusions.

How do we know if the size of our system is sufficiently large to yield quantitative results that are independent of N ? The straightforward answer is to repeat the simulation for larger N . Fortunately, most simulations of equilibrium systems with simple interactions require only several hundred to several thousand particles for reliable results. How do we know if our runs are long enough to give statistically meaningful averages? The simple answer is to run longer and see if the averages change significantly.

In general, the most time consuming parts of a molecular dynamics simulation are generating an appropriate initial configuration and doing the bookkeeping necessary for the force and energy calculations. If the force is sufficiently short range, there are a number of ways to reduce the equilibration time. For example, suppose we want to simulate a system of 864 particles in three dimensions. We first can simulate a system of 108 particles and allow the small system to come to equilibrium at the desired temperature. After equilibrium has been established, the small system can be replicated twice in each direction to generate the desired system of 864 particles. All of the velocities are reassigned at random using the Maxwell-Boltzmann distribution. Equilibration of the new system usually is established quickly.

The computer time required for our simple molecular dynamics program is order N^2 for each time step. The reason for this quadratic dependence on N is that the energy and force calculations require sums over all $\frac{1}{2}N(N - 1)$ pairs of particles. If the interactions are short range, the time required for these sums can be reduced to approximately order N . The idea is to take advantage of the fact that at any given time, most pairs of particles are separated by a distance much greater than the effective range r_c of the interparticle interaction ($r_c \approx 2.5\sigma$ for the Lennard-Jones potential). Hence the calculation of the force and the energy requires the consideration of only those pairs of particles whose separation is less than r_c . Because testing whether each pair satisfies this criterion is an order N^2 calculation, we have to limit the number of pairs tested. One method is to divide the box into small cells and to compute the distance between particles that are in the same cell or in nearby cells. Another method is to maintain a list for each particle of its neighbors whose separation is less than a distance r_l , where r_l is chosen to be slightly greater than r_c so that the neighbor list can be used for several time steps before it is updated again. Both the cell method and the neighbor list method do not become efficient until N is approximately a few hundred.

So far we have discussed molecular dynamics simulations at fixed energy, volume, and number of particles. In laboratory systems we usually keep the temperature rather than the energy fixed, and frequently consider systems at fixed pressure rather than at fixed volume. It is possible to do molecular dynamics simulations at constant temperature and/or pressure. It also is possible to do simulations in which the shape of the cell is determined by the dynamics rather than imposed by the program. Such a simulation is essential for the study of solid-to-solid transitions where the major change is the shape of the crystal.

In addition to these technical advances, there is much more to learn about the properties of the system by doing averages over the trajectories. For example, how are transport properties such as the viscosity and the thermal conductivity related to the trajectories? We also have not discussed one of the most fundamental properties of a many body system, namely, its entropy. In brief, not all macroscopic properties of a many body system can be simply defined as a time average over some function of the phase space coordinates of the particles. The entropy is an example of such a quantity (but see Ma). However, changes in the entropy can be computed by using thermodynamic integration or a test particle method (see references).

There is another fundamental limitation of molecular dynamics, namely the **multiple time scale problem**. We know that we must choose the time step Δt to be smaller than any physical time scale in the system. For a solid, the smallest time scale is the period of the oscillatory motion of individual particles about their equilibrium positions. If we want to know how the solid responds if we add an interstitial particle or create a vacancy, we would have to run for millions of small time steps for the vacancy to move several interparticle distances. Although this particular problem can be overcome by using a faster computer, there are many problems for which no imaginable supercomputer would be sufficient. One of the biggest challenges of present interest is the **protein folding problem**. The biological function of a protein is determined by its three-dimensional structure which is encoded by the sequence of amino acids in the protein. At present, we know little about how the protein forms its three-dimensional structure. Such formidable computational challenges remind us that we cannot simply put a problem on a computer and let the computer tell us the answer. In particular, molecular dynamics methods need to be complemented by other simulation methods, especially Monte Carlo methods (see Chapters ?? and ??).

Now that we are familiar with the method of molecular dynamics, we briefly discuss its historical role in aiding our present understanding of simple equilibrium liquids. Simulations of systems of hard disks and hard spheres have shown that the structure of these systems does not differ significantly from the structure of systems with more complicated interactions. Given this insight, our present theories of liquids are based on the use of the hard sphere (disk) system as a reference system; the differences between the hard sphere interaction and the more complicated interaction of interest are treated as a perturbation about this reference system.

The emphasis in current applications of molecular dynamics is shifting from the studies of simple equilibrium fluids to studies of more complex fluids and studies of nonequilibrium systems. For example, how does a solid form when the temperature of a liquid is lowered quickly? How does a crack propagate in a brittle solid? What is the nature of the glass transition? Molecular dynamics and related methods will play an important role in aiding our understanding of these and many other problems.

8.12 Projects

Many of the pioneering applications of molecular dynamics were done on relatively small systems. It is interesting to peruse the research literature of the past three decades and to see how much physical insight was obtained from these simulations. Many research-level problems can be generated by first reproducing previously published work and then extending the work to larger systems or longer run times to obtain better statistics. An interesting project based on recent research is suggested in the following. Some related projects are discussed in Section ??.

Project 8.19. Single particle fluctuation metric

As we discussed briefly in Section 8.7, the quasi-ergodic hypothesis assumes that time averages and ensemble averages are identical for a system in thermodynamic equilibrium. The assumption is that if we run a molecular dynamics simulation for a sufficiently long time, then the dynamical trajectory will fill the accessible phase space.

One way to confirm the quasi-ergodic hypothesis is to compute an ensemble average by simulating many independent copies of the system of interest using different initial configurations.

Another way is to simulate a very large system and compare the behavior of different parts. A more direct and computationally efficient measure of the ergodicity has been proposed by Thirumalai and Mountain. This measure is called the fluctuation metric and is based on a comparison of the time averaged quantity $\overline{f_i(t)}$ of f_i for particle i to its average for all other particles. Effectively, we take the ensemble average of f over all particles, rather than over different parts of the system. If the system is ergodic, then all particles see the same average environment, and the time average $\overline{f_i(t)}$ for each particle will be the same if t is sufficiently long. Note that $\overline{f_i(t)}$ is the average of the quantity f_i over the time interval t and not the value of f_i at time t . The time average of f_i is defined as

$$\overline{f_i(t)} = \frac{1}{t} \int_0^t f_i(t') dt', \quad (8.42)$$

and the average of $\overline{f_i(t)}$ over all particles is given by

$$\langle f(t) \rangle = \frac{1}{N} \sum_{i=1}^N \overline{f_i(t)}. \quad (8.43)$$

One of the physical quantities of interest is the energy of a particle, e_i , defined as

$$e_i = \frac{p_i^2}{2m_i} + \frac{1}{2} \sum_{i \neq j} u(r_{ij}). \quad (8.44)$$

The factor of $1/2$ is included in the potential energy term in (8.44) because the interaction energy is shared between pairs of particles. The above considerations lead us to define the energy fluctuation metric, $\Omega_e(t)$, as

$$\Omega_e(t) = \frac{1}{N} \sum_{i=1}^N \left[\overline{e_i(t)} - \langle e(t) \rangle \right]^2. \quad (8.45)$$

1. Compute $\Omega_e(t)$ for a system of Lennard-Jones particles at a relatively high temperature. Determine $e_i(t)$ at time intervals of 0.5 or less and average Ω_e over as many time origins as possible. If the system is displaying behavior that is expected of an ergodic system over the time interval t , it can be shown that $\Omega_e(t)$ decreases as $1/t$. Do you find $1/t$ behavior for relatively short times? Nonergodic behavior might be found by rapidly reducing the kinetic energy (a temperature quench) and obtaining an amorphous solid or glass rather than a crystalline solid. However, it would be necessary to consider three-dimensional rather than two-dimensional systems because the latter nucleate to a crystalline solid very quickly.
2. Another quantity of interest is the velocity fluctuation metric Ω_v :

$$\Omega_v(t) = \frac{1}{dN} \sum_{i=1}^N \left[\overline{\mathbf{v}_i(t)} - \langle \mathbf{v}(t) \rangle \right]^2. \quad (8.46)$$

The factor of $1/d$ in (8.46) is included because the velocity is a vector with d components. If we choose the total momentum of the system to be zero, then $\langle \mathbf{v}(t) \rangle = 0$, and we can write

(8.46) as

$$\Omega_v(t) = \frac{1}{dN} \sum_{i=1}^N \overline{\mathbf{v}_i(t)} \cdot \overline{\mathbf{v}_i(t)}. \quad (8.47)$$

We now show that the t dependence of $\Omega_v(t)$ is not a good indicator of ergodicity, but can be used to determine the diffusion coefficient D . We write

$$\overline{\mathbf{v}_i(t)} = \frac{1}{t} \int_0^t \mathbf{v}_i(t') dt' = \frac{1}{t} [\mathbf{r}_i(t) - \mathbf{r}_i(0)]. \quad (8.48)$$

If we substitute (8.48) into (8.47), we can express the velocity fluctuation metric in terms of the mean square displacement:

$$\Omega_v(t) = \frac{1}{dNt^2} \sum_{i=1}^N [\mathbf{r}_i(t) - \mathbf{r}_i(0)]^2 = \frac{\langle R^2(t) \rangle}{dt^2}. \quad (8.49)$$

The average in (8.49) is over all particles. If the particles are diffusing during the time interval t , then $\langle R^2(t) \rangle = 2dDt$, and

$$\Omega_v(t) = 2D/t. \quad (8.50)$$

From (8.50) we see that $\Omega_v(t)$ goes to zero as $1/t$ as claimed in part (a). However, if the particles are localized (as in a crystalline solid and a glass), then $\langle R^2 \rangle$ is bounded for all t , and $\Omega_v(t) \sim 1/t^2$. Because a crystalline solid is ergodic and a glass is not, the velocity fluctuation metric is not a good measure of the lack of ergodicity. Use the t dependence of $\Omega_v(t)$ in (8.50) to determine D for the same configurations as in Problem 8.17. Note that the determination of D from Ω_v does not require a correction for the use of periodic boundary conditions.

References and Suggestions for Further Reading

- Farid F. Abraham, “Computational statistical mechanics: methodology, applications and supercomputing,” *Adv. Phys.* **35**, 1 (1986). The author discusses both molecular dynamics and Monte Carlo techniques.
- M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Clarendon Press (1987). See Chapter 7 for a discussion of the test particle method.
- R. P. Bonomo and F. Riggi, “The evolution of the speed distribution for a two-dimensional ideal gas: A computer simulation,” *Am. J. Phys.* **52**, 54 (1984). The authors consider a system of hard disks and show that the system always evolves toward the Maxwell-Boltzmann distribution.
- J. P. Boon and S. Yip, *Molecular Hydrodynamics*, Dover (1991). Their discussion of transport properties is an excellent supplement to our brief discussion.

Giovanni Ciccotti and William G. Hoover, editors, *Molecular-Dynamics Simulation of Statistical-Mechanics Systems*, North-Holland (1986).

Giovanni Ciccotti, Daan Frenkel, and Ian R. McDonald, editors, *Simulation of Liquids and Solids*, North-Holland (1987). A collection of reprints on the simulation of many body systems. Of particular interest are B. J. Alder and T. E. Wainwright, "Phase transition in elastic disks," *Phys. Rev.* **127**, 359 (1962) and earlier papers by the same authors; A. Rahman, "Correlations in the motion of atoms in liquid argon," *Phys. Rev.* **136**, A405 (1964), the first application of molecular dynamics to systems with continuous potentials; and Loup Verlet, "Computer 'experiments' on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules," *Phys. Rev.* **159**, 98 (1967).

J. M. Haile, *Molecular Dynamics Simulation*, John Wiley & Sons (1992). A derivation of the mean pressure using periodic boundary conditions is given in Appendix B.

Jean Pierre Hansen and Ian R. McDonald, *Theory of Simple Liquids*, second edition, Academic Press (1986). An excellent reference that derives most of the theoretical results used in this chapter.

R. M. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*, Adam Hilger (1988).

W. G. Hoover, *Molecular Dynamics*, Springer-Verlag (1986) and W. G. Hoover, *Computational Statistical Mechanics*, Elsevier (1991).

J. Kushick and B. J. Berne, "Molecular dynamics methods: continuous potentials" in *Statistical Mechanics Part B: Time-Dependent Processes*, Bruce J. Berne, editor, Plenum Press (1977). Also see the article by Jerome J. Erpenbeck and William Wood on "Molecular dynamics techniques for hard-core systems" in the same volume.

Shang-keng Ma, "Calculation of entropy from data of motion," *J. Stat. Phys.* **26**, 221, (1981). See also Chapter 25 of Ma's graduate level text, *Statistical Mechanics*, World Scientific (1985). Ma discusses a novel approach for computing the entropy directly from the trajectories. Note that the coincidence rate in Ma's approach is related to the recurrence time for a finite system to return to an arbitrarily small neighborhood of almost any given initial state.

S. Ranganathan, G. S. Dubey, and K. N. Pathak, "Molecular-dynamics study of two-dimensional Lennard-Jones fluids," *Phys. Rev. A* **45**, 5793 (1992). Two-dimensional systems are of interest because they are simpler theoretically and computationally and are related to single layer films.

Dennis Rapaport, *The Art of Molecular Dynamics Simulation*, Cambridge University Press (1995).

John R. Ray and H. W. Graben, "Direct calculation of fluctuation formulae in the microcanonical ensemble," *Mol. Phys.* **43**, 1293 (1981).

F. Reif, *Fundamentals of Statistical and Thermal Physics*, McGraw-Hill (1965.) An intermediate level text on statistical physics with a more thorough discussion of kinetic theory than found in most undergraduate texts. *Statistical Physics*, Vol. 5 of the Berkeley Physics Course, McGraw-Hill (1965) by the same author was one of the first texts to use computer simulations to illustrate the approach of macroscopic systems to equilibrium.

Marco Ronchetti and Gianni Jacucci, editors, *Simulation Approach to Solids*, Kluwer Academic Publishers (1990). Another collection of reprints.

R. M. Sperandeo Mineo and R. Madonia, "The equation of state of a hard-particle system: a model experiment on a microcomputer," *Eur. J. Phys.* **7**, 124 (1986).

D. Thirumalai and Raymond D. Mountain, "Ergodic convergence properties of supercooled liquids and glasses," *Phys. Rev. A* **42**, 4574 (1990).

James H. Williams and Glenn Joyce, "Equilibrium properties of a one-dimensional kinetic system," *J. Chem. Phys.* **59**, 741 (1973). Simulations in one dimension are even easier than in two.

Chapter 9

Normal Modes and Waves

©2000 by Harvey Gould and Jan Tobochnik
6 December 2000

We discuss the physics of wave phenomena and the motivation and use of Fourier transforms.

9.1 Coupled Oscillators and Normal Modes

Terms such as period, amplitude, and frequency are used to describe both waves and oscillatory motion. To understand the relation between the latter two phenomena, consider a flexible rope that is under tension with one end fixed. If we flip the free end, a pulse propagates along the rope with a speed that depends on the tension and on the inertial properties of the rope. At the *macroscopic* level, we observe a transverse wave that moves along the length of the rope. In contrast, at the *microscopic* level we see discrete particles undergoing oscillatory motion in a direction perpendicular to the motion of the wave. One goal of this chapter is to use simulations to understand the relation between the microscopic dynamics of a simple mechanical model and the macroscopic wave motion that the model can support. For simplicity, we consider a one-dimensional chain of L particles each of mass M . The particles are coupled by massless springs with force constant K . The equilibrium separation between the particles is a . We denote the displacement of particle j from its equilibrium position at time t by $u_j(t)$ (see Figure ??). For many purposes the most realistic boundary conditions are to attach particles $j = 1$ and $j = L$ to springs which are attached to fixed walls. We denote the walls by $j = 0$ and $j = L + 1$, and require that $u_0(t) = u_{L+1}(t) = 0$.

The force on an individual particle is determined by the compression or extension of the adjacent springs. The equation of motion of particle j is given by

$$\begin{aligned} M \frac{d^2 u_j(t)}{dt^2} &= -K[u_j(t) - u_{j+1}(t)] - K[u_j(t) - u_{j-1}(t)] \\ &= -K[2u_j(t) - u_{j+1}(t) - u_{j-1}(t)]. \end{aligned} \tag{9.1}$$

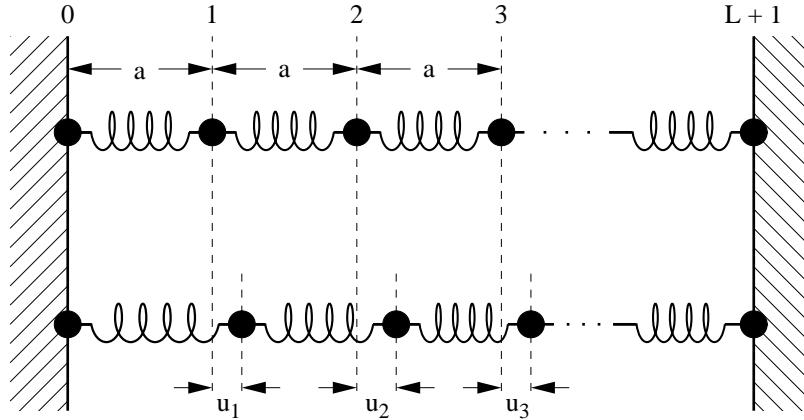


Figure 9.1: A one-dimensional chain of L particles of mass M coupled by massless springs with force constant K . The first and last particles (0 and $L + 1$) are attached to fixed walls. The top chain shows the oscillators in equilibrium. The bottom chain shows the oscillators displaced from equilibrium.

As expected, the motion of particle j is coupled to its two nearest neighbors. The equations of motion (9.1) describe *longitudinal* oscillations, i.e., motion along the length of the system. It is straightforward to show that identical equations hold for the *transverse* oscillations of L identical mass points equally spaced on a stretched massless string (cf. French).

The equations of motion (9.1) are linear, that is, only terms proportional to the displacements appear. It is straightforward to obtain analytical solutions of (9.1). Although the analytical solution will help us interpret the numerical solutions in terms of normal modes, it is not necessary to understand the analytical solution in detail to understand the numerical solutions.

To find the normal modes, we look for solutions for which the displacement of each particle is proportional to $\sin \omega t$ or $\cos \omega t$. We write

$$u_j(t) = u_j \cos \omega t, \quad (9.2)$$

where u_j is the amplitude of vibration of the j th particle. If we substitute the form (??) into (9.1), we obtain

$$-\omega^2 u_j = -\frac{K}{M} [2u_j - u_{j+1} - u_{j-1}]. \quad (9.3)$$

We next assume that u_j depends sinusoidally on the distance ja :

$$u_j = C \sin qja. \quad (9.4)$$

The magnitude of the constant C will be determined later. If we substitute the form (??) into (??), we find the following condition for ω :

$$-\omega^2 \sin qja = -\frac{K}{M} [2 \sin qja - \sin q(j-1)a - \sin q(j+1)a]. \quad (9.5)$$

We write $\sin q(j \pm 1)a = \sin qja \cos qa \pm \cos qja \sin qa$ and find that (??) is a solution if

$$\omega^2 = 2\frac{K}{M}(1 - \cos qa). \quad (9.6)$$

We need to find the values of the wavenumber q that satisfy the boundary conditions $u_0 = 0$ and $u_{L+1} = 0$. The former condition is automatically satisfied by assuming a sine instead of a cosine solution in (??). The latter boundary condition implies that

$$q = q_n = \frac{\pi n}{a(L+1)} \quad n = 1, \dots, L \quad (\text{fixed boundary conditions}) \quad (9.7)$$

What are the corresponding possible values of the wavelength λ ? The latter is related to q by $q = 2\pi/\lambda$. The corresponding values of the angular frequencies are given by

$$\omega_n^2 = 2\frac{K}{M}[1 - \cos q_n a] = 4\frac{K}{M} \sin^2 \frac{q_n a}{2} \quad (9.8)$$

or

$$\omega_n = 2\sqrt{\frac{K}{M}} \sin \frac{q_n a}{2}. \quad (9.9)$$

The relation (??) between ω_n and q_n is known as a dispersion relation.

A particular value of the integer n corresponds to the n th *normal mode*. We write the (time-independent) normal mode solutions as

$$u_{j,n} = C \sin q_n j a. \quad (9.10)$$

The linear nature of the equation of motion (9.1) implies that the time dependence of the displacement of the j th particle can be written as a superposition of normal modes:

$$u_j(t) = C \sum_{n=1}^L (A_n \cos \omega_n t + B_n \sin \omega_n t) \sin q_n j a \quad (9.11)$$

The coefficients A_n and B_n are determined by the initial conditions:

$$u_j(t=0) = C \sum_{n=1}^L A_n \sin q_n j a \quad (9.12a)$$

and

$$v_j(t=0) = C \sum_{n=1}^L \omega_n B_n \sin q_n j a. \quad (9.12b)$$

To solve (??) for A_n and B_n , we note that the normal mode solutions, $u_{j,n}$, are *orthogonal*, that is, they satisfy the condition

$$\sum_{j=1}^L u_{j,n} u_{j,m} \propto \delta_{n,m}. \quad (9.13)$$

The Kronecker δ symbol $\delta_{n,m} = 1$ if $n = m$ and is zero otherwise. It is convenient to normalize the $u_{j,n}$ so that they are *orthonormal*, i.e.,

$$\sum_{j=1}^L u_{j,n} u_{j,m} = \delta_{n,m}. \quad (9.14)$$

It is easy to show that the choice, $C = 1/\sqrt{(L+1)/2}$, in (??) and (??) insures that (??) is satisfied.

We now use the orthonormality condition to determine the coefficients A_n and B_n . If we multiply both sides of (??) by $C \sin q_m ja$, sum over j , and use the orthogonality condition (??), we obtain

$$A_n = C \sum_{j=1}^L u_j(0) \sin q_n ja \quad (9.15)$$

and

$$B_n = C \sum_{j=1}^L (v_j(0)/\omega_n) \sin q_n ja. \quad (9.16)$$

For example, if the initial displacement of every particle is zero, and the initial velocity of every particle is zero except for $v_1(0) = 1$, we find $A_n = 0$ for all n , and

$$B_n = \frac{C}{\omega_n} \sin q_n a. \quad (9.17)$$

The corresponding solution for $u_j(t)$ is

$$u_j(t) = \frac{2}{L+1} \sum_{n=1}^L \frac{1}{\omega_n} \cos \omega_n t \sin q_n a \sin q_n ja. \quad (9.18)$$

What is the solution if the particles start in a normal mode, i.e., $u_j(t=0) \propto \sin q_2 ja$?

The analytical solution (??) together with the initial conditions represent the complete solution of the displacement of the particles. If we wish, we can use a computer to compute the sum in (??) and plot the time dependence of the displacements $u_j(t)$. There are many interesting extensions that are amenable to analytical solutions. What is the effect of changing the boundary conditions? What happens if the spring constants are not all equal, but are chosen from a probability distribution? What happens if we vary the masses of the particles? For these cases we can follow a similar approach and look for the eigenvalues ω_n and eigenvectors $u_{j,n}$ of the matrix equation

$$\mathbf{T} \mathbf{u} = \omega^2 \mathbf{u}. \quad (9.19)$$

The matrix elements $T_{i,j}$ are zero except for

$$T_{i,i} = \frac{1}{M_i} [K_{i,i+1} + K_{i,i-1}] \quad (9.20a)$$

$$T_{i,i+1} = -\frac{K_{i,i+1}}{M_i} \quad (9.20b)$$

and

$$T_{i,i-1} = -\frac{K_{i,i-1}}{M_i}, \quad (9.20c)$$

where $K_{i,j}$ is the spring constant between particles i and j . The solution of matrix equations is a well studied problem in linear programming, and a commercial subroutine package such as IMSL or a symbolic programming language such as Maple, MatLab, or Mathematica can be used to obtain the solutions.

For our purposes it is easier to find the numerical solution of the equations of motion (9.1) directly because we also are interested in the effects of nonlinear forces between the particles, a case for which the matrix approach is inapplicable. In **Program oscillators** we use the Euler-Richardson algorithm to simulate the dynamics of L linearly coupled oscillators. The particle displacements are displayed as transverse oscillations using techniques similar to **Program animation** in Chapter 5. Note that we have used the **MAT** instruction to assign the array **useave** to **u** (see **SUB initial**). This instruction is equivalent to assigning every element of the array **useave** the corresponding value of the array **u**.

```

PROGRAM oscillators
! simulate coupled linear oscillators in one dimension
DIM u(0 to 21),v(0 to 21),useave(0 to 21)
CALL initial(L,u(),v(),t,dt,useave(),mass$,erase$)
DO
    CALL update(L,u(),v(),ke,t,dt,useave())
    CALL animate(L,u(),ke,t,useave(),mass$,erase$)
LOOP until key input
END

SUB initial(L,u(),v(),t,dt,useave(),mass$,erase$)
DATA 0,0.5,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0,0
LET t = 0
LET dt = 0.025
LET L = 10                      ! number of particles
SET WINDOW -1,L+1,-4,4
SET COLOR "red"
BOX AREA 0.9,1.1,-0.1,0.1
BOX KEEP 0.9,1.1,-0.1,0.1 in mass$
SET COLOR "background"
BOX AREA -0.1,0.1,-0.1,0.1
BOX KEEP -0.1,0.1,-0.1,0.1 in erase$
PLOT line -2,0;L+2,0
FOR j = 1 to L
    READ u(j)                  ! initial displacements
    BOX SHOW mass$ at j-0.1,u(j)-0.1
NEXT j
FOR j = 1 to L
    READ v(j)                  ! initial velocities

```

```

NEXT j
LET u(0) = 0           ! fixed wall boundary conditions
LET u(L+1) = 0
MAT usave = u          ! note use of matrix assignment instruction
END SUB

SUB update(L,u(),v(),ke,t,dt,usave())
  ! Euler-Richardson algorithm
  DIM a(20),amid(20),umid(0 to 21),vmid(20)
  LET ke = 0
  ! K/M equal to unity
  FOR j = 1 to L
    LET usave(j) = u(j)
    LET a(j) = -2*u(j) + u(j+1) + u(j-1)
    LET umid(j) = u(j) + 0.5*v(j)*dt
    LET vmid(j) = v(j) + 0.5*a(j)*dt
  NEXT j
  LET umid(0) = 0
  LET umid(L+1) = 0
  FOR j = 1 to L
    LET amid(j) = -2*umid(j) + umid(j+1) + umid(j-1)
    LET u(j) = u(j) + vmid(j)*dt
    LET v(j) = v(j) + amid(j)*dt
    LET ke = ke + v(j)*v(j)
  NEXT j
  LET t = t + dt
END SUB

SUB animate(L,u(),ke,t,usave(),mass$,erase$)
  LET pe = (u(1) - u(0))^2      ! interaction with left spring
  FOR j = 1 to L
    ! compute potential energy
    LET pe = pe + (u(j+1) - u(j))^2
    ! transverse oscillation
    BOX SHOW erase$ at j-0.1,usave(j)-0.1
    BOX SHOW mass$ at j-0.1,u(j)-0.1
  NEXT j
  PLOT line -2,0;L+2,0
  SET CURSOR 1,1
  SET COLOR "black"
  PRINT using "t = #####.##": t
  LET E = 0.5*(ke + pe)
  PRINT using "E = #.#####": E
END SUB

```

Problem 9.1. Motion of coupled oscillators

1. Run **Program oscillators** for $L = 2$ and choose the initial values of $u(1)$ and $u(2)$ so that the system is in one of its two normal modes, e.g., $u(1) = u(2) = 0.5$. Set the initial velocities equal to zero. Note that the program sets the ratio $K/M = 1$. Describe the displacement of the particles. Is the motion of each particle periodic in time? To answer this question, add a subroutine that plots the displacement of each particle versus the time. Then consider the other normal mode, e.g., $u(1) = 0.5$, $u(2) = -0.5$. What is the period in this case? Does the system remain in a normal mode indefinitely? Finally, choose the initial particle displacements equal to random values between -0.5 and $+0.5$. Is the motion of each particle periodic in this case?
2. Consider the same questions as in part (a), but with $L = 4$ and $L = 10$. Consider the $n = 2$ mode for $L = 4$ and the $n = 3$ and $n = 8$ modes for $L = 10$. (See (??) for the form of the normal mode solutions.) Also consider random initial displacements.
3. **Program oscillators** depicts the oscillations as transverse because they are easier to visualize. Modify the program to represent longitudinal oscillations instead. Define the density as the number of particles within a certain range of x . For example, set $L = 20$ and describe how the average density varies as a function of the time within the region defined by $8 < x < 12$. Use the initial condition $u_j = \sin(3j\pi/(L+1))$ corresponding to the third normal mode. Repeat for another normal mode.
4. Write a program to verify that the normal mode solutions (??) are orthonormal. Then compare the analytical results and the numerical results for $L = 10$ using the initial conditions listed in the DATA statements in **Program oscillators**. How much faster is it to calculate the analytical solution? What is the maximum deviation between the analytical and numerical solution of $u_j(t)$? How well is the total energy conserved in **Program oscillators**? How does the maximum deviation and the conservation of the total energy change when the time step Δt is reduced?

Problem 9.2. Motion of coupled oscillators with external forces

1. Modify **Program oscillators** so that an external force F_e is exerted on the first particle,

$$F_e/m = 0.5 \cos \omega_e t, \quad (9.21)$$

where ω_e is the angular frequency of the external force. (Note that ω_e is an angular frequency, but as is common practice, we frequently refer to ω_e as a frequency.) Let the initial displacements and velocities of all L particles be zero. Choose $L = 3$ and then $L = 10$ and consider the response of the system to an external force for $\omega = 0.5$ to 4.0 in steps of 0.5 . Record $A(\omega)$, the maximum amplitude of any particle, for each value of ω . Explain how this system can be used as a high frequency filter.

2. Choose ω_e to be one of the normal mode frequencies. Does the maximum amplitude remain constant or does it increase with time? How can you use the response of the system to an external force to determine the normal mode frequencies? Discuss your results in terms of the power input, $F_e v_1$?
3. In addition to the external force exerted on the first particle, add a damping force equal to $-\gamma v_i$ to all the oscillators. Choose the damping constant $\gamma = 0.05$. How do you expect the

system to behave? How does the maximum amplitude depend on ω_e ? Are the normal mode frequencies changed when $\gamma \neq 0$?

Problem 9.3. Different boundary conditions

1. Modify **Program oscillators** so that periodic boundary conditions are used, i.e., $u(L+1) = u(1)$ and $u(0) = u(L)$. Choose $L = 10$, and the initial condition corresponding to the normal mode (??) with $n = 2$. Does this initial condition yield a normal mode solution for periodic boundary conditions? It might be easier to answer this question by plotting $u(i)$ versus time for two or more particles. For fixed boundary conditions there are $L + 1$ springs, but for periodic boundary conditions there are L springs. Why? Choose the initial condition corresponding to the $n = 2$ normal mode, but replace $L + 1$ by L in (??). Does this initial condition correspond to a normal mode? Now try $n = 3$, and other values of n . Which values of n give normal modes? Only sine functions can be normal modes for fixed boundary conditions (see (??)). Can there be normal modes with cosine functions if we use periodic boundary conditions?
2. Modify **Program oscillators** so that free boundary conditions are used, that is, $u(L+1) = u(L)$ and $u(0) = u(1)$. Choose $L = 10$. Use the initial condition corresponding to the $n = 3$ normal mode found using fixed boundary conditions. Does this condition correspond to a normal mode for free boundary conditions? Is $n = 2$ a normal mode for free boundary conditions? Are the normal modes purely sinusoidal?
3. Choose free boundary conditions and $L \geq 10$. Let the initial condition be a pulse of the form, $u(1) = 0.2, u(2) = 0.6, u(3) = 1.0, u(4) = 0.6, u(5) = 0.2$, and all other $u(j) = 0$. After the pulse reaches the right end, what is the phase of the reflected pulse, i.e., are the displacements in the reflected pulse in the same direction as the incoming pulse (a phase shift of zero degrees) or in the opposite direction (a phase shift of 180 degrees)? What happens for fixed boundary conditions? Choose L to be as large as possible so that it is easy to distinguish the incident and reflected waves.
4. Set $L = 20$ and let the spring constants on the right half of the system be four times greater than the spring constants on the left half. Use fixed boundary conditions. Set up a pulse on the left side. Is there a reflected pulse at the boundary between the two types of springs? If so, what is its relative phase? Compare the amplitude of the reflected and transmitted pulses. Consider the same questions with a pulse that is initially on the right side.

9.2 Fourier Transforms

In Section 9.1, we showed that the displacement of a single particle can be written as a linear combination of normal modes, that is, a linear superposition of sinusoidal terms. In general, an arbitrary periodic function $f(t)$ of period T can be expressed as a Fourier series of sines and cosines:

$$f(t) = \frac{1}{2}a_0 + \sum_{k=1}^{\infty} (a_k \cos \omega_k t + b_k \sin \omega_k t), \quad (9.22)$$

where

$$\omega_k = k\omega_0 \quad \text{and} \quad \omega_0 = \frac{2\pi}{T}. \quad (9.23)$$

The quantity ω_0 is the fundamental frequency. The sine and cosine terms in (??) for $k = 2, 3, \dots$ represent the second, third, \dots , and higher order harmonics. The *Fourier coefficients* a_k and b_k are given by

$$a_k = \frac{2}{T} \int_0^T f(t) \cos \omega_k t dt \quad (9.24a)$$

$$b_k = \frac{2}{T} \int_0^T f(t) \sin \omega_k t dt. \quad (9.24b)$$

The constant term $\frac{1}{2}a_0$ in (??) is the average value of $f(t)$. The expressions (??) for the coefficients follow from the orthogonality conditions:

$$\frac{2}{T} \int_0^T \sin \omega_k t \sin \omega_{k'} t dt = \delta_{k,k'} \quad (9.25a)$$

$$\frac{2}{T} \int_0^T \cos \omega_k t \cos \omega_{k'} t dt = \delta_{k,k'}. \quad (9.25b)$$

$$\frac{2}{T} \int_0^T \sin \omega_k t \cos \omega_{k'} t dt = 0. \quad (9.25c)$$

In general, an infinite number of terms is needed to represent an arbitrary periodic function exactly. In practice, a good approximation usually can be obtained by including a relatively small number of terms. Unlike a power series, which can approximate a function only near a particular point, a Fourier series can approximate a function at all points. Program `synthesize`, listed in the following, plots the sum (??) for various values of N , the number of terms in the series. One purpose of the program is to help us visualize how well a finite sum of harmonic terms can represent an arbitrary periodic function.

```

PROGRAM synthesize
CALL plotf(0,0.5,0.5,1)
CALL plotf(0,0.5,0,0.5)
CALL plotf(0.5,1,0.5,1)
CALL plotf(0.5,1,0,0.5)
END

SUB plotf(xmin,xmax,ymin,ymax)
OPEN #1: screen xmin,xmax,ymin,ymax
SET WINDOW -4,4,-2,2
PLOT LINES: -pi,0;pi,0
PLOT LINES: 0,-1.5;0,1.5
INPUT prompt "number of modes = ": N
SET COLOR "red"

```

```

CALL fourier(N)
CLOSE #1
END SUB

SUB fourier(N)
    ! compute Fourier series and plot function
    DIM a(0 to 1000),b(1000)
    CALL coefficients(N,a(),b())
    LET nplot = 100
    LET t = -pi
    LET dt = pi/100
    DO while t <= pi
        LET f = a(0)/2
        FOR k = 1 to N
            IF a(k) <> 0 then LET f = f + a(k)*cos(k*t)
            IF b(k) <> 0 then LET f = f + b(k)*sin(k*t)
        NEXT k
        PLOT LINES: t,f;
        LET t = t + dt
    LOOP
END SUB

SUB coefficients(N,a(),b())
    ! generate Fourier coefficients for special case
    LET a(0) = 0
    FOR k = 1 to N
        LET a(k) = 0
        IF mod(k,2) <> 0 then
            LET b(k) = 2/(k*pi)
        ELSE
            LET b(k) = 0
        END IF
    NEXT k
END SUB

```

Problem 9.4. Fourier synthesis

1. The process of constructing a function by adding together a fundamental frequency and harmonics of various amplitudes is called *Fourier synthesis*. Use **Program synthesize** to visualize how a sum of harmonic functions can represent an arbitrary periodic function. Consider the series

$$f(t) = \frac{2}{\pi}(\sin t + \frac{1}{3} \sin 3t + \frac{1}{5} \sin 5t + \dots). \quad (9.26)$$

Describe the nature of the plot when only the first three terms in (??) are retained. Increase the number of terms until you are satisfied that (??) represents the function sufficiently accurately. What function is represented by the infinite series?

2. Modify **Program** `synthesize` so that you can “zoom in” on the visual representation of $f(t)$ for different intervals of t . Consider the series (??) with at least 32 terms. For what values of t does the finite sum most faithfully represent the exact function? For what values of t does it not? Why is it necessary to include a large number of terms to represent $f(t)$ where it has sharp edges? The small oscillations that increase in amplitude as a sharp edge is approached are known as the Gibbs phenomenon.
3. Use **Program** `synthesize` to determine the function that is represented by the Fourier series with coefficients $a_k = 0$ and $b_k = (2/k\pi)(-1)^{k-1}$ for $k = 1, 2, 3, \dots$. Approximately how many terms in the series are required?

So far we have considered how a sum of sines and cosines can approximate a known periodic function. More typically, we measure a time series consisting of N data points, $f(t_i)$, where $t_i = 0, \Delta, 2\Delta, \dots, (N-1)\Delta$. We assume that the data repeats itself with a period T given by $T = N\Delta$. (The time interval Δ between the measurements should not be confused with the finite time step Δt used in the numerical solution of a differential equation.) Our goal is to determine the Fourier coefficients a_k and b_k because, as we will see, these coefficients contain important physical information.

If we know only a finite number of terms in a time series, it is possible to find only a finite set of Fourier coefficients. For a given value of Δ , what is the largest frequency component we can extract? In the following, we give a plausibility argument that suggests that the maximum frequency we can analyze is

$$\omega_c = \frac{\pi}{\Delta}. \quad (\text{Nyquist critical frequency}) \quad (9.27)$$

One way to understand this result is to imagine that $f(t)$ is a sine wave. If $f(t_i)$ has the same value for all t_i , the period is equal to either Δ or Δ/n , where n is an integer. The largest frequency component we can determine in this case is $\omega = 2\pi n/\Delta$, an arbitrarily large quantity. Hence, a constant data set does not impose any limitations on the maximum frequency. Now suppose that $f(t_i)$ has one value for even i and another value for odd i . In this case we know that the period is 2Δ , and hence the maximum possible frequency of this function is $\omega = 2\pi/(2\Delta) = \pi/\Delta$. More variations in $f(t_i)$ would correspond to lower frequencies, and hence we conclude that the highest frequency is π/Δ .

One consequence of (??) is that there are $\omega_c/\omega_0 + 1$ independent coefficients for a_k (including a_0), and ω_c/ω_0 independent coefficients for b_k , a total of $N + 1$ independent coefficients. (Recall that $\omega_c/\omega_0 = N/2$, where $\omega_0 = 2\pi/T$ and $T = N\Delta$.) However, because $\sin \omega_c t = 0$ for all values of t that are multiples of Δ , we have that $b_{N/2} = 0$ from (??). Consequently, there are $N/2 - 1$ values for b_k , and hence a total of N Fourier coefficients that can be computed. This conclusion is reasonable because the number of meaningful Fourier coefficients should be the same as the number of data points.

Program `analyze` computes the Fourier coefficients a_k and b_k of a function $f(t)$ defined between $t = 0$ and $t = T$ at intervals of Δ , and plots a_k and b_k versus k . To compute the coefficients we do the integrals in (??) numerically using the simple rectangular approximation (see Section ??):

$$a_k \approx \frac{2\Delta}{T} \sum_{i=0}^{N-1} f(t_i) \cos \omega_k t_i \quad (9.28a)$$

$$b_k \approx \frac{2\Delta}{T} \sum_{i=0}^{N-1} f(t_i) \sin \omega_k t_i, \quad (9.28b)$$

where the ratio $2\Delta/T = 2/N$.

```

PROGRAM analyze
! determine the Fourier coefficients a_k and b_k
CALL parameters(N,nmax,delta,period)
CALL screen(nmax,period,#1,#2)
CALL coefficents(N,nmax,delta,period,#1,#2)
END

SUB parameters(N,nmax,delta,period)
INPUT prompt "number of data points N (even) = ": N
INPUT prompt "sampling time dt = ": delta
LET period = N*delta           ! assumed period
! maximum value of mode corresponding to Nyquist frequency
LET nmax = N/2
END SUB

SUB screen(nmax,period,#1,#2)
LET ymax = 2
LET ticksize = ymax/50
OPEN #1: screen 0,1,0.5,1
PRINT "      a_k";
PRINT "      ";
PRINT using "frequency interval = #.#####": 2*pi/period
SET WINDOW -1,nmax+1,-ymax,ymax
CALL plotaxis(nmax,ticksize)
SET COLOR "red"
OPEN #2: screen 0,1,0,0.5
PRINT "      b_k"
SET WINDOW -1,nmax+1,-ymax,ymax
CALL plotaxis(nmax,ticksize)
SET COLOR "red"
END SUB

SUB plotaxis(nmax,ticksize)
PLOT LINES: 0,0;nmax,0
FOR k = 1 to nmax
    PLOT LINES: k,-ticksize;k,ticksize

```

```

NEXT k
END SUB

SUB coefficents(N,nmax,delta,period,#1,#2)
DECLARE DEF f
FOR k = 0 to nmax
    LET ak = 0
    LET bk = 0
    LET wk = 2*pi*k/period
    ! rectangular approximation
    FOR i = 0 to N - 1
        LET t = i*delta
        LET ak = ak + f(t)*cos(wk*t)
        LET bk = bk + f(t)*sin(wk*t)
    NEXT i
    LET ak = 2*ak/N
    LET bk = 2*bk/N
    WINDOW #1
    PLOT LINES: k,0;k,ak
    WINDOW #2
    PLOT LINES: k,0;k,bk
NEXT k
END SUB

FUNCTION f(t)
LET w0 = 0.1*pi
LET f = sin(w0*t)           ! simple example
END DEF

```

In Problem ?? we compute the Fourier coefficients for several known functions. We will see that if $f(t)$ is a sum of sinusoidal functions with different periods, it is essential that the period T in Program `analyze` be an integer multiple of the periods of all the functions in the sum. If T does not satisfy this condition, then the results for some of the Fourier coefficients will be spurious. In practice, the solution to this problem is to vary the sampling rate and the total time over which the signal $f(t)$ is sampled. Fortunately, the results for the power spectrum (see below) are less ambiguous than the values for the Fourier coefficients themselves.

Problem 9.5. Fourier analysis

1. Use Program `analyze` with $f(t) = \sin \pi t / 10$. Determine the Fourier coefficients by doing the integrals in (??) analytically before running the program. Choose the number of data points to be $N = 200$ and the sampling time $\Delta = 0.1$. Which Fourier components are nonzero? Repeat your analysis for $N = 400, \Delta = 0.1$; $N = 200, \Delta = 0.05$; $N = 205, \Delta = 0.1$; and $N = 500, \Delta = 0.1$, and other combinations of N and Δ . Explain your results by comparing the period of $f(t)$ with $N\Delta$, the assumed period. If the combination of N and Δ are not chosen properly, do you find any spurious results for the coefficients?

2. Consider the functions $f_1(t) = \sin \pi t/10 + \sin \pi t/5$, $f_2(t) = \sin \pi t/10 + \cos \pi t/5$, and $f_3(t) = \sin \pi t/10 + \frac{1}{2} \cos \pi t/5$, and answer the same questions as in part (a). What combinations of N and Δ give reasonable results for each function?
3. Consider a function that is not periodic, but falls to zero for large $\pm t$. For example, try $f(t) = t^4 e^{-t^2}$ and $f(t) = t^3 e^{-t^2}$. Interpret the difference between the Fourier coefficients of these two functions.

Fourier analysis can be simplified by using exponential notation and combining the sine and cosine functions in one expression. We express $f(t)$ as

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{i\omega_k t}, \quad (9.29)$$

and use (??) to express the complex coefficients c_k in terms of a_k and b_k :

$$c_k = \frac{1}{2}(a_k - i b_k) \quad (9.30a)$$

$$c_0 = \frac{1}{2}a_0 \quad (9.30b)$$

$$c_{-k} = \frac{1}{2}(a_k + i b_k). \quad (9.30c)$$

The coefficients c_k can be expressed in terms of $f(t)$ by using (??) and (??) and the fact that $e^{\pm i\omega_k t} = \cos \omega_k t \pm i \sin \omega_k t$. The result is

$$c_k = \frac{1}{T} \int_0^T f(t) e^{-i\omega_k t} dt. \quad (9.31)$$

As in (??), we can approximate the integral in (??) using the rectangular approximation. We write

$$g(\omega_k) \equiv c_k \frac{T}{\Delta} \approx \sum_{j=0}^{N-1} f(j\Delta) e^{-i\omega_k j\Delta} = \sum_{j=0}^{N-1} f(j\Delta) e^{-i2\pi k j/N}. \quad (9.32)$$

If we multiply (??) by $e^{i2\pi k j'/N}$, sum over k , and use the orthogonality condition

$$\sum_{k=0}^{N-1} e^{i2\pi k j/N} e^{-i2\pi k j'/N} = N \delta_{j,j'}, \quad (9.33)$$

we obtain the inverse Fourier transform

$$f(j\Delta) = \frac{1}{N} \sum_{k=0}^{N-1} g(\omega_k) e^{i2\pi k j/N} = \frac{1}{N} \sum_{k=0}^{N-1} g(\omega_k) e^{i\omega_k t_j}. \quad (9.34)$$

The frequencies ω_k for $k > N/2$ in the summations in (??) are greater than the Nyquist frequency ω_c . However, from (??), we see that $g(\omega_k) = g(\omega_k - \omega_N)$. Hence, we can interpret all frequencies

for $k > N/2$ as negative frequencies equal to $(k - N)\omega_0$. If $f(t)$ is real, then $g(-\omega_k) = g(\omega_k)$. The occurrence of negative frequency components is a consequence of the use of the exponential functions rather than a sum of sines and cosines.

The importance of a particular frequency component within a signal is measured by the power $P(\omega)$ associated with that frequency. To obtain this power, we use the discrete form of Parseval's theorem which can be written as

$$\sum_{j=0}^{N-1} |f(t_j)|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |g(\omega_k)|^2. \quad (9.35)$$

In most measurements the function $f(t)$ corresponds to an amplitude, and the power or intensity is proportional to the square of this amplitude or for complex functions, the modulus squared. Note that the left-hand sum in (??) (and hence the right-hand side) is proportional to N , and hence we need to divide both sides by N to obtain a quantity independent of N . The power in the frequency component ω_k is proportional to

$$P(\omega_k) = \frac{1}{N^2} [|g(\omega_k)|^2 + |g(-\omega_k)|^2] = \frac{2}{N^2} |g(\omega_k)|^2. \quad (0 < \omega_k < \omega_c) \quad (9.36a)$$

The last equality follows if $f(t)$ is real. Because the Fourier coefficients for $\omega = \omega_c$ and $\omega = -\omega_c$ are identical, we write for this case:

$$P(\omega_c) = \frac{1}{N^2} |g(\omega_c)|^2. \quad (9.36b)$$

Similarly, there is only one term with zero frequency, and hence $P(0)$ is given by

$$P(0) = \frac{1}{N^2} |g(0)|^2. \quad (9.36c)$$

The *power spectrum* $P(\omega)$ defined in (??) is proportional to the power associated with a particular frequency component embedded in the quantity of interest.

What happens to the power associated with frequencies greater than the Nyquist frequency? To answer this question, consider two choices of the Nyquist frequency, ω_c^a and $\omega_c^b > \omega_c^a$, and the corresponding sampling times, $\Delta^b < \Delta^a$. The calculation with $\Delta = \Delta^b$ represents the more accurate calculation because the sampling time is smaller. Suppose that this calculation of the spectrum yields the result that $P(\omega > \omega_c^a) > 0$. What happens if we compute the power spectrum using $\Delta = \Delta^a$? The power associated with $\omega > \omega_c^a$ must be “folded” back into the $\omega < \omega_c^a$ frequency components. For example, the frequency component at $\omega + \omega_c^a$ is added to the true value at $\omega - \omega_c^a$ to produce an incorrect value at $\omega - \omega_c^a$ in the computed power spectrum. This phenomenon is called *aliasing* and leads to spurious results. Aliasing occurs in calculations of $P(\omega)$ if the latter does not vanish above the Nyquist frequency. To avoid aliasing, it is necessary to sample more frequently, or to remove the high frequency components from the signal before computing the Fourier transform.

The power spectrum can be computed by a simple modification of **Program analyze**. The procedure is order N^2 , because there are N integrals for the N Fourier components, each of which is divided into N intervals. However, many of the calculations are redundant, and it is possible to organize the calculation so that the computational time is order $N \log N$. Such an algorithm is called a *fast Fourier transform* (FFT) and is discussed in [Appendix 9A](#). It is a good idea to use the FFT for many of the following problems.

Problem 9.6. Examples of power spectra

1. Create a data set corresponding to $f(t) = 0.3 \cos(2\pi t/T) + r$, where r is a random number between 0 and 1. Plot $f(t)$ versus t in intervals of $\Delta = 4T/N$ for $N = 128$ values. Can you visually detect any periodicity? Then compute the power spectrum using the same sampling interval $\Delta = 4T/N$. Does the behavior of the power spectrum indicate that there are any special frequencies?
2. Simulate a one-dimensional random walk, and compute $x^2(t)$, where $x(t)$ is the distance from the origin of the walk after t steps. Compute the power spectrum for a walk of $t = 256$. In this case $\Delta = 1$, the time between steps. Do you observe any special frequencies? Remember to average over several samples.
3. Let f_n be the n th number of a random number sequence so that the time $t = n$ with $\Delta = 1$. Compute the power spectrum of the random number generator. Do you detect any periodicities? If so, is the random number generator acceptable?

Problem 9.7. Power spectrum of coupled oscillators

1. Modify **Program oscillators** so that the power spectrum of one of the L particles is computed at the end of the simulation. Set $\Delta = 0.1$ so that the Nyquist frequency is $\omega_c = \pi/\Delta \approx 31.4$. Choose the time of the simulation equal to $T = 25.6$ and let $K/M = 1$. Plot the power spectrum $P(\omega)$ at frequency intervals equal to $\Delta\omega = \omega_0 = 2\pi/T$. First choose $L = 2$ and choose the initial conditions so that the system is in a normal mode. What do you expect the power spectrum to look like? What do you find? Then choose $L = 10$ and choose initial conditions corresponding to various normal modes.
2. Repeat part (a) for $L = 2$ and $L = 10$ with the initial particle displacements equal to random values between -0.5 and 0.5 . Can you detect all the normal modes in the power spectrum? Repeat for a different set of random initial displacements.
3. Repeat part (a) for initial displacements corresponding to the sum of two normal modes.
4. Recompute the power spectrum for $L = 10$ with $T = 6.4$. Is this time long enough? How can you tell?

**Problem 9.8.* Quasiperiodic power spectra

1. Write a program to compute the power spectrum of the circle map (6.56). Begin by exploring the power spectrum for $K = 0$. Plot $\ln P(\omega)$ versus ω , where $P(\omega)$ is proportional to the modulus squared of the Fourier transform of x_n . Begin with $N = 256$ iterations. How does the power spectra differ for rational and irrational values of the parameter Ω ? How are the locations of the peaks in the power spectra related to the value of Ω ?
2. Set $K = 1/2$ and compute the power spectra for $0 < \Omega < 1$. Does the power spectra differ from the spectra found in part (a)?
3. Set $K = 1$ and compute the power spectra for $0 < \Omega < 1$. How does the power spectra compare to those found in parts (a) and (b)?

In Problem ?? we found that the peaks in the power spectrum yield information about the normal mode frequencies. In Problem ?? and ?? we compute the power spectra for a system of coupled oscillators where disorder is present. Disorder can be generated by having random masses and/or random spring constants. We will see that one effect of disorder is that the normal modes are no longer simple sinusoidal functions. Instead, some of the modes are localized, meaning that only some of the particles move significantly while the others remain essentially at rest. This effect is known as *Anderson localization*. Typically, we find that modes above a certain frequency are *localized*, and those below this threshold frequency are *extended*. This threshold frequency is well defined for large systems. In one dimension with a finite disorder (e.g., a finite density of defects) all states are localized in the limit of an infinite chain.

Problem 9.9. Localization with a single defect

1. Modify **Program oscillators** so that the mass of one oscillator is equal to one fourth that of the others. Set $L = 20$ and use fixed boundary conditions. Compute the power spectrum over a time $T = 51.2$ using random initial displacements between -0.5 and 0.5 and zero initial velocities. Sample the data at intervals of $\Delta = 0.1$. The normal mode frequencies correspond to the well defined peaks in $P(\omega)$. Consider at least three different sets of random initial displacements to insure that you find all the normal mode frequencies.
2. Apply an external force $F_e = 0.3 \sin \omega_e t$ to each particle. (The steady state behavior occurs sooner if we apply an external force to each particle instead of just one particle.) Because the external force pumps energy into the system, it is necessary to add a damping force to prevent the oscillator displacements from becoming too large. Add a damping force equal to $-\gamma v_i$ to all the oscillators with $\gamma = 0.1$. Choose random initial displacements and zero initial velocities and use the frequencies found in part (a) as the driving frequencies ω_e . Describe the motion of the particles. Is the system driven to a normal mode? Take a “snapshot” of the particle displacements after the system has run for a sufficiently long time so that the patterns repeat themselves. Are the particle displacements simple sinusoidal functions of position? Sketch the approximate normal mode patterns for each normal mode frequency. Which of the modes appear localized and which modes appear to be extended? What is the approximate cutoff frequency that separates the localized from the extended modes?

Problem 9.10. Localization in a disordered chain of oscillators

1. Modify **Program oscillators** so that the spring constants can be varied by the user. Set $L = 10$ and use fixed wall boundary conditions. Consider the following set of 11 spring constants:

```
DATA 0.704,0.388,0.707,0.525,0.754,0.721
DATA 0.006,0.479,0.470,0.574,0.904
```

To help you determine all the normal modes, we provide two of the normal mode frequencies: $\omega \approx 0.28$ and 1.15 . Find the power spectrum using the procedure outlined in Problem ??a.

2. Apply an external force $F_e = 0.3 \sin \omega_e t$ to each particle, and find the normal modes as outlined in Problem ??b.

3. Repeat parts (a) and (b) for another set of random spring constants. If you have sufficient computer resources, consider $L = 40$. Discuss the nature of the localized modes in terms of the specific values of the spring constants. For example, is the edge of a localized mode at a spring that has a relatively large or small spring constant?
4. Repeat parts (a) and (b) for uniform spring constants, but random masses between 0.5 and 1.5. Is there a qualitative difference between the two types of disorder?

In 1955 Fermi, Pasta, and Ulam used the Maniac I computer at Los Alamos to study a chain of oscillators. Their surprising discovery might have been the first time a qualitatively new result, instead of a more precise number, was found from a computer simulation. To understand their results (known as the FPU problem), we need to discuss an idea from statistical mechanics that was mentioned briefly in Project 8.19. Some of the ideas of statistical mechanics are introduced in greater depth in later chapters.

A fundamental assumption of statistical mechanics is that an isolated system of particles is ergodic, that is, the system will evolve through all configurations consistent with the conservation of energy. Clearly, a set of linearly coupled oscillators is not ergodic, because if the system is initially in a normal mode, it stays in that normal mode forever. Before 1955 it was believed that if the interaction between the particles is slightly nonlinear (and the number of particles is sufficiently large), the system would be ergodic and evolve through the different normal modes of the linear system. In Problem ?? we will find, as did Fermi, Pasta, and Ulam, that the behavior of the system is much more complicated.

Problem 9.11. Nonlinear oscillators

1. Modify **Program oscillators** so that cubic forces between the particles are added to the linear spring forces, i.e., let the force on particle i due to particle j be

$$F_{ij} = -(u_i - u_j) - \alpha(u_i - u_j)^3, \quad (9.37)$$

where α is the amplitude of the nonlinear term. Choose the masses of the particles to be unity. Consider $L = 10$ and choose initial displacements corresponding to a normal mode of the linear ($\alpha = 0$) system. Compute the power spectrum over a time interval of 51.2 with $\Delta = 0.1$ for $\alpha = 0, 0.1, 0.2$, and 0.3 . For what value of α does the system become ergodic, i.e., the heights of all the normal mode peaks are approximately the same?

2. Repeat part (a) for the case where the displacements of the particles are initially random. Make sure the same set of random displacements are used for each value of α .
3. We now know that the number of oscillators is not as important as the magnitude of the nonlinear interaction. Repeat parts (a) and (b) for $L = 20$ and 40 and discuss the effect of increasing the number of particles.

**Problem 9.12.* Spatial Fourier transforms

1. So far we have considered Fourier transforms in time and frequency. Spatial Fourier transforms are of interest in many contexts. The main difference is that spatial transforms usually

involve positive and negative values of x , whereas we have considered only nonnegative values of t . Modify Program `analyze` so that it computes the real and imaginary parts of the Fourier transform $\phi(k)$ of a complex function $\psi(x)$, where both x and k can have negative values. That is, instead of doing the integral (??) from 0 to T , integrate from $-L/2$ to $L/2$, where $\psi(x+L) = \psi(x)$.

2. Compute the Fourier transform of the Gaussian function $\psi(x) = Ae^{-bx^2}$. Plot $\psi(x)$ and $\phi(k)$ for at least three values of b . Does $\phi(k)$ appear to be a Gaussian? Choose a reasonable criterion for the half-width of $\psi(x)$ and measure its value. Use the same criterion to measure the half-width of $\phi(k)$. How do these widths depend on b ? How does the width of $\phi(k)$ change as the width of $\psi(x)$ increases?
3. Repeat part (b) with the function $\psi(x) = Ae^{-bx^2} e^{ik_0 x}$ for various values of k_0 .

9.3 Wave Motion

Our simulations of coupled oscillators have shown that the microscopic motion of the individual oscillators leads to macroscopic wave phenomena. To understand the transition between microscopic and macroscopic phenomena, we reconsider the oscillations of a linear chain of L particles with equal spring constants K and equal masses M . As we found in Section 9.1, the equations of motion of the particles can be written as (see (9.1))

$$\frac{d^2u_j(t)}{dt^2} = -\frac{K}{M}[2u_j(t) - u_{j+1}(t) - u_{j-1}(t)]. \quad (i = 1, \dots, L). \quad (9.38)$$

We consider the limits $L \rightarrow \infty$ and $a \rightarrow 0$ with the length of the chain La fixed. We will find that the discrete equations of motion (??) can be replaced by the continuous *wave equation*

$$\frac{\partial^2 u(x, t)}{\partial t^2} = c^2 \frac{\partial^2 u(x, t)}{\partial x^2}, \quad (9.39)$$

where c has the dimension of velocity.

We can obtain the wave equation (??) as follows. First we replace $u_j(t)$, where j is a discrete variable, by the function $u(x, t)$, where x is a *continuous* variable, and rewrite (??) in the form

$$\frac{\partial^2 u(x, t)}{\partial t^2} = \frac{Ka^2}{M} \frac{1}{a^2} [u(x+a, t) - 2u(x, t) + u(x-a, t)]. \quad (9.40)$$

We have written the time derivative as a partial derivative because the function u depends on two variables. If we use the Taylor series expansion

$$u(x \pm a) = u(x) \pm a \frac{\partial u}{\partial x} + \frac{a^2}{2} \frac{\partial^2 u}{\partial x^2} + \dots, \quad (9.41)$$

it is easy to show that as $a \rightarrow 0$, the quantity

$$\frac{1}{a^2} [u(x+a, t) - 2u(x, t) + u(x-a, t)] \rightarrow \frac{\partial^2 u(x, t)}{\partial x^2}. \quad (9.42)$$

The wave equation (??) is obtained by substituting (??) into (??) with $c^2 = Ka^2/M$. If we introduce the linear mass density $\mu = M/a$ and the tension $T = Ka$, we can express c in terms of μ and T and obtain the familiar result $c^2 = T/\mu$.

It is straightforward to show that any function of the form $f(x \pm ct)$ is a solution to (??). Among these many solutions to the wave equation are the familiar forms:

$$u(x, t) = A \cos \frac{2\pi}{\lambda} (x \pm ct) \quad (9.43a)$$

$$u(x, t) = A \sin \frac{2\pi}{\lambda} (x \pm ct). \quad (9.43b)$$

Because the wave equation is linear, and hence satisfies a superposition principle, we can understand the behavior of a wave of arbitrary shape by representing its shape as a sum of sinusoidal waves.

One way to solve the wave equation numerically is to retrace our steps back to the discrete equations (??) to find a discrete form of the wave equation that is convenient for numerical calculations. This procedure of converting a continuum equation to a physically motivated discrete form frequently leads to useful numerical algorithms. From (??) we see how to approximate the second derivative by a finite difference. If we replace a by Δx and take Δt to be the time step, we can rewrite (??) by

$$\begin{aligned} \frac{1}{(\Delta t)^2} [u(x, t + \Delta t) - 2u(x, t) + u(x, t - \Delta t)] = \\ \frac{c^2}{(\Delta x)^2} [u(x + \Delta x, t) - 2u(x, t) + u(x - \Delta x, t)]. \end{aligned} \quad (9.44)$$

The quantity Δx is the spatial interval. The result of solving (??) for $u(x, t + \Delta t)$ is

$$\begin{aligned} u(x, t + \Delta t) = & 2[1 - b]u(x, t) \\ & + b[u(x + \Delta x, t) + u(x - \Delta x, t)] - u(x, t - \Delta t), \end{aligned} \quad (9.45)$$

where $b \equiv (c\Delta t/\Delta x)^2$. Equation (??) expresses the displacements at time $t + \Delta t$ in terms of the displacements at the current time t and at the previous time $t - \Delta t$.

Problem 9.13. Solution of the discrete wave equation

1. Write a program to compute the numerical solutions of the discrete wave equation (??). Three spatial arrays corresponding to $u(x)$ at times $t + \Delta t$, t , and $t - \Delta t$ are needed, where Δt is the time step. We denote the displacement $u(j\Delta x)$ by the array element $u(j)$, where Δx is the size of the spatial grid. Use periodic boundary conditions so that $u(0) = u(L)$ and $u(L+1) = u(1)$, where L is the total number of spatial intervals. Draw lines between the displacements at neighboring values of x . Note that the initial conditions require the specification of the array u at $t = 0$ and at $t = -\Delta t$. Let the waveform at $t = 0$ and $t = -\Delta t$ be $u(x, t = 0) = \exp(-(x - 10)^2)$ and $u(x, t = -\Delta t) = \exp(-(x - 10 + c\Delta t)^2)$, respectively. What is the direction of motion implied by these initial conditions?
2. Our first task is to determine the optimum value of the parameter b . Let $\Delta x = 1$ and $L \geq 100$, and try the following combinations of c and Δt : $c = 1, \Delta t = 0.1$; $c = 1, \Delta t = 0.5$; $c = 1, \Delta t = 1$; $c = 1, \Delta t = 1.5$; $c = 2, \Delta t = 0.5$; and $c = 2, \Delta t = 1$. Verify that the value $b = (c\Delta t)^2 = 1$ leads to the best results, that is, for this value of b , the initial form of the wave is preserved.

3. It is possible to show that the discrete form of the wave equation with $b = 1$ is exact up to numerical roundoff error (cf. DeVries). Hence, we can replace (??) by the simpler algorithm

$$u(x, t + \Delta t) = u(x + \Delta x, t) + u(x - \Delta x, t) - u(x, t - \Delta t). \quad (9.46)$$

That is, the solutions of (??) are equivalent to the solutions of the original partial differential equation (??). Try several different initial waveforms, and show that if the displacements have the form $f(x \pm ct)$, then the waveform maintains its shape with time. For the remaining problems we use (??) corresponding to $b = 1$. Unless otherwise specified, choose $c = 1$, $\Delta x = \Delta t = 1$, and $L \geq 100$ in the following problems.

Problem 9.14. Velocity of waves

1. Use the waveform given in Problem ??a and measure the speed of the wave by determining the distance traveled on the screen in a given amount of time. Add tick marks to the x axis. Because we have set $\Delta x = \Delta t = 1$ and $b = 1$, the speed $c = 1$. (A way of incorporating different values of c is discussed in Problem ??d.)
2. Replace the waveform considered in part (a) by a sinusoidal wave that fits exactly, i.e., choose $u(x, t) = \sin(qx - \omega t)$ such that $\sin q(L + 1) = 0$. Measure the period T of the wave by measuring the time it takes for successive maxima to pass a given point. What is the wavelength λ of your wave? Does it depends on the value of q ? The frequency of the wave is given by $f = 1/T$. Verify that $\lambda f = c$.

Problem 9.15. Reflection of waves

1. Consider a wave of the form $u(x, t) = e^{-(x-10-ct)^2}$. Use fixed boundary conditions so that $u(0) = u(L+1) = 0$. What happens to the reflected wave?
2. Modify your program so that free boundary conditions are incorporated: $u(0) = u(1)$ and $u(L+1) = u(L)$. Compare the phase of the reflected wave to your result from part (a).
3. Modify your program so that a “sluggish” boundary condition, e.g., $u(0) = \frac{1}{2}u(1)$ and $u(L+1) = \frac{1}{2}u(L)$, is used. What do you expect the reflected wave to look like? What do you find from your numerical solution?
4. What happens to a pulse at the boundary between two media? Set $c = 1$ and $\Delta t = 1$ on the left side of your grid and $c = 2$ and $\Delta t = 0.5$ on the right side. These choices of c and Δt imply that $b = 1$ on both sides, but that the right side is updated twice as often as the left side. What happens to a pulse that begins on the left side and moves to the right? Is there both a reflected and transmitted wave at the boundary between the two media? What is their relative phase? Find a relation between the amplitude of the incident pulse and the amplitudes of the reflected and transmitted pulses. Repeat for a pulse starting from the right side.

Problem 9.16. Superposition of waves

1. Consider the propagation of the wave determined by $u(x, t = 0) = \sin(4\pi x/L)$. What must $u(x, -\Delta t)$ be such that the wave moves in the positive x direction? Test your answer by doing the simulation. Use periodic boundary conditions. Repeat for a wave moving in the negative x direction.

2. Simulate two waves moving in opposite directions each with the same spatial dependence given by $u(x, 0) = \sin 4\pi x/L$. Describe the resultant wave pattern. Repeat the simulation for $u(x, 0) = \sin 8\pi x/L$.
3. Assume that $u(x, 0) = \sin q_1 x + \sin q_2 x$, with $\omega_1 = cq_1 = 10\pi/L$, $\omega_2 = cq_2 = 12\pi/L$, and $c = 1$. Describe the qualitative form of $u(x, t)$ for fixed t . What is the distance between modulations of the amplitude? Estimate the wavelength associated with the fine ripples of the amplitude. Estimate the wavelength of the *envelope* of the wave. Find a simple relationship for these two wavelengths in terms of the wavelengths of the two sinusoidal terms. This phenomena is known as *beats*.
4. Consider the motion of the wave described by $u(x, 0) = e^{-(x-10)^2} + e^{-(x-90)^2}$; the two Gaussian pulses move in opposite directions. What happens to the two pulses as they travel through each other? Do they maintain their shape? While they are going through each other, is the displacement $u(x, t)$ given by the sum of the displacements of the individual pulses?

Problem 9.17. Standing waves

1. In Problem ??c we considered a *standing wave*, the continuum analog of a normal mode of a system of coupled oscillators. As is the case for normal modes, each point of the wave has the same time dependence. For fixed boundary conditions, the displacement is given by $u(x, t) = \sin qx \cos \omega t$, where $\omega = cq$ and q is chosen so that $\sin qL = 0$. Choose an initial condition corresponding to a standing wave for $L = 100$. Describe the motion of the particles, and compare it with your observations of standing waves on a rope.
2. Establish a standing wave by displacing one end of a system periodically. The other end is fixed. Let $u(x, 0) = u(x, -\Delta t) = 0$, and $u(x = 0, t) = A \sin \omega t$ with $A = 0.1$.

We have seen that the wave equation can support pulses that propagate indefinitely without distortion. In addition, because the wave equation is linear, the sum of any two solutions also is a solution, and the principle of superposition is satisfied. As a consequence, we know that two pulses can pass through each other unchanged. We also have seen that similar phenomena exist in the discrete system of linearly coupled oscillators. What happens if we create a pulse in a system of nonlinear oscillators? As an introduction to nonlinear wave phenomena, we consider a system of L coupled oscillators with the potential energy of interaction given by

$$V = \frac{1}{2} \sum_{j=1}^L (e^{-(u_j - u_{j-1})} - 1)^2. \quad (9.47)$$

This form of the interaction is known as the Morse potential. All parameters in the potential (such as the overall strength of the potential) have been set to unity. The force on the j th particle is

$$F_j = -\frac{\partial V}{\partial u_j} = Q_j(1 - Q_j) - Q_{j+1}(1 - Q_{j+1}), \quad (9.48a)$$

where

$$Q_j \equiv e^{-(u_j - u_{j-1})}. \quad (9.48b)$$

In linear systems it is possible to set up a pulse of any shape and maintain the shape of the pulse indefinitely. In a nonlinear system there also exist solutions that maintain their shape, but we will find in Problem ?? that not all pulse shapes do so. The pulses that maintain their shape are called *solitons*.

Problem 9.18. Solitons

1. Modify **Program oscillators** so that the force on particle j is given by (??). Use periodic boundary conditions. Choose $L \geq 60$ and an initial Gaussian pulse of the form $u(x, t) = 0.5 e^{-(x-10)^2}$. You should find that the initial pulse splits into two pulses plus some noise. Describe the motion of the pulses (solitons). Do they maintain their shape, or is this shape modified as they move? Describe the motion of the particles far from the pulse. Are they stationary?
2. Save the displacements of the particles when the peak of one of the solitons is located near the center of your screen. Is it possible to fit the shape of the soliton to a Gaussian? Continue the simulation, and after one of the solitons is relatively isolated, set $u(j) = 0$ for all j far from this soliton. Does the soliton maintain its shape?
3. Repeat part (b) with a pulse given by $u(x, 0) = 0$ everywhere except for $u(20, 0) = u(21, 0) = 1$. Do the resulting solitons have the same shape as in part (b)?
4. Begin with the same Gaussian pulse as in part (a), and run until the two solitons are well separated. Then change at random the values of $u(j)$ for particles in the larger soliton by about 5%, and continue the simulation. Is the soliton destroyed? Increase this perturbation until the soliton is no longer discernible.
5. Begin with a single Gaussian pulse as in part (a). The two resultant solitons will eventually "collide." Do the solitons maintain their shape after the collision? The principle of superposition implies that the displacement of the particles is given by the sum of the displacements due to each pulse. Does the principle of superposition hold for solitons?
6. Compute the speeds, amplitudes, and width of the solitons produced from a single Gaussian pulse. Take the amplitude of a soliton to be the largest value of its displacement and the half-width to correspond to the value of x at which the displacement is half its maximum value. Repeat these calculations for solitons of different amplitudes by choosing the initial amplitude of the Gaussian pulse to be 0.1, 0.3, 0.5, 0.7, and 0.9. Plot the soliton speed and width versus the corresponding soliton amplitude.
7. Change the boundary conditions to free boundary conditions and describe the behavior of the soliton as it reaches a boundary. Compare this behavior with that of a pulse in a system of linear oscillators.
8. Begin with an initial sinusoidal disturbance that would be a normal mode for a linear system. Does the sinusoidal mode maintain its shape? Compare the behavior of the nonlinear and linear systems.

9.4 Interference and Diffraction

Interference is one of the most fundamental characteristics of all wave phenomena. The term *interference* is used when relatively few sources of waves separately derived from the same source are brought together. The term *diffraction* has a similar meaning and is commonly used if there are many sources. Because it is relatively easy to observe interference and diffraction phenomena with light, we discuss these phenomena in this context.

The classic example of interference is Young's double slit experiment (see Figure ??). Imagine two narrow parallel slits separated by a distance a and illuminated by a light source that emits light of only one frequency (monochromatic light). If the light source is placed on the line bisecting the two slits and the slit opening is very narrow, the two slits become coherent light sources with equal phases. We first assume that the slits act as point sources, e.g., pinholes. A screen that displays the intensity of the light from the two sources is placed a distance L away. What do we see on the screen?

The electric field at position \mathbf{r} associated with the light emitted from a monochromatic point source at \mathbf{r}_1 has the form

$$E(\mathbf{r}, t) = \frac{A}{|\mathbf{r} - \mathbf{r}_1|} \cos(q|\mathbf{r} - \mathbf{r}_1| - \omega t), \quad (9.49)$$

where $|\mathbf{r} - \mathbf{r}_1|$ is the distance between the source and the point of observation. The superposition principle implies that the total electric field at \mathbf{r} from N point sources at \mathbf{r}_i is

$$E(\mathbf{r}, t) = \sum_{i=1}^N \frac{A}{|\mathbf{r} - \mathbf{r}_i|} \cos(q|\mathbf{r} - \mathbf{r}_i| - \omega t). \quad (9.50)$$

Equation ?? assumes that the amplitude of each source is the same. The observed intensity is proportional to the time-averaged value of $|E|^2$.

In Problem ?? we discuss writing a program to determine the intensity of light that is observed on a screen due to an arrangement of point sources. The wavelength of the light sources, the positions of the sources \mathbf{r}_i , and the observation points on the screen need to be specified. The program sums the fields due to all the sources for a given observation point, and computes $|E|^2$. The part of the program that is not straightforward is the calculation of the time average of $|E|^2$. One way of obtaining the time average is to compute the integral

$$\overline{E^2} = \frac{1}{T} \int_0^T |E|^2 dt, \quad (9.51)$$

where $T = 1/f$ is the period, and f is the frequency of the light sources. We now show that such a calculation is not necessary if the sources are much closer to each other than they are to the screen. In this case (the *far field condition*), we can ignore the slow r dependence of $|\mathbf{r} - \mathbf{r}_i|^{-1}$ and write the field in the form

$$E(\mathbf{r}) = E_0 \cos(\phi - \omega t). \quad (9.52)$$

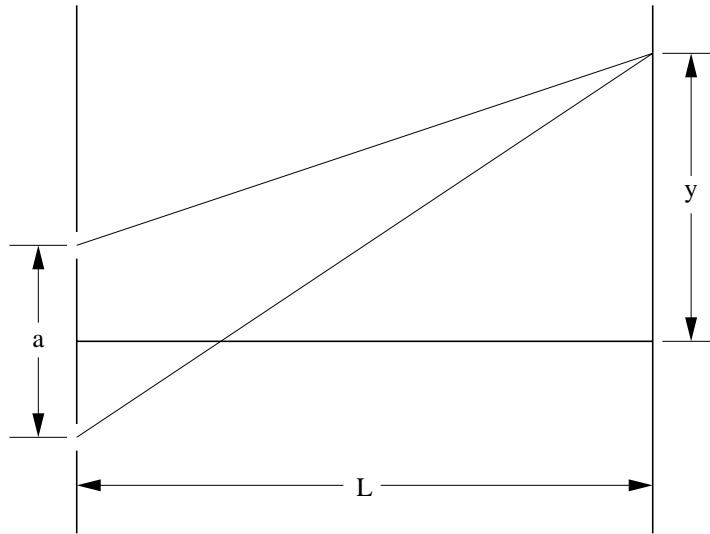


Figure 9.2: Young's double slit experiment. The figure defines the quantities a , L , and y that are used in Problem ??.

We have absorbed the factors of $|\mathbf{r} - \mathbf{r}_i|^{-1}$ into E_0 . The phase ϕ is a function of the source positions and \mathbf{r} . The form of (??) allows us to write

$$\frac{1}{T} \int_0^T \cos^2(\phi - \omega t) dt = \frac{1}{M} \sum_{m=1}^M \cos^2\left(\phi - \frac{2\pi m}{M}\right) = \frac{1}{2}. \quad (M > 2) \quad (9.53)$$

The result (??) is independent of ϕ and allows us to perform the time average by using the summation in (??) with $M = 3$.

In the following problems we discuss a variety of geometries. The main part of your program that must be changed is the specification of the source positions.

Problem 9.19. Double slit interference

1. Verify (??) by finding an analytical expression for E from two and three point sources. Explain why the form (??) is valid for an arbitrary number of sources. Then write a program to verify the result (??) for $M = 3, 4, 5$, and 10. What is the value of the sum in (??) for $M = 2$?
2. Write a program to plot the intensity of light on a screen due to two slits. Calculate E using (??) and do the time average using the relation (??). Let a be the distance between the slits and y be the vertical position along the screen as measured from the central maximum. Set $L = 200$ mm, $a = 0.1$ mm, the wavelength of light $\lambda = 5000$ Å (1 Å = 10^{-7} mm), and consider -5.0 mm $\leq y \leq 5.0$ mm (see Figure ??). Describe the interference pattern you observe with $M = 3$. Identify the locations of the intensity maxima, and plot the intensity of the maxima as a function of y .

3. Repeat part (b) for $L = 0.5\text{ mm}$ and $1.0\text{ mm} \leq y \leq 1.0\text{ mm}$. Do your results change as you vary M from 3 to 5? Note that in this case L is not much greater than a , and hence we cannot ignore the r dependence of $|\mathbf{r} - \mathbf{r}_i|^{-1}$ in (??). How large must M be so that your results are approximately independent of M ?

Problem 9.20. Diffraction grating

High resolution optical spectroscopy is done with multiple slits. In its simplest form, a diffraction grating consists of N parallel slits made, for example, by ruling with a diamond stylus on aluminum plated glass. Compute the intensity of light for $N = 3, 4, 5$, and 10 slits with $\lambda = 5000\text{ \AA}$, slit separation $a = 0.01\text{ mm}$, screen distance $L = 200\text{ mm}$, and $-15\text{ mm} \leq y \leq 15\text{ mm}$. How does the intensity of the peaks and their separation vary with N ?

In our analysis of the double slit and the diffraction grating, we assumed that each slit was a pinhole that emits spherical waves. In practice, real slits are much wider than the wavelength of visible light. In Problem ?? we consider the pattern of light produced when a plane wave is incident on an aperture such as a single slit. To do so, we use Huygens' principle and replace the slit by many coherent sources of spherical waves. This equivalence is not exact, but is applicable when the aperture width is large compared to the wavelength.

Problem 9.21. Single slit diffraction

1. Compute the time averaged intensity of light diffracted from a single slit of width 0.02 mm by replacing the slit by $N = 20$ point sources spaced 0.001 mm apart. Choose $\lambda = 5000\text{ \AA}$, $L = 200\text{ mm}$, and consider $-30\text{ mm} \leq y \leq 30\text{ mm}$. What is the width of the central peak? How does the width of the central peak compare to the width of the slit? Do your results change if N is increased?
2. Determine the position of the first minimum of the diffraction pattern as a function of wavelength, slit width, and distance to the screen.
3. Compute the intensity pattern for $L = 1\text{ mm}$ and 50 mm . Is the far field condition satisfied in this case? How do the patterns differ?

Problem 9.22. A more realistic double slit simulation

Reconsider the intensity distribution for double slit interference using slits of finite width. Modify your program to simulate two “thick” slits by replacing each slit by 20 point sources spaced 0.001 mm apart. The centers of the thick slits are $a = 0.1\text{ mm}$ apart. How does the intensity pattern change?

**Problem 9.23.* Diffraction pattern due to a rectangular aperture

We can use a similar approach to determine the diffraction pattern due to an aperture of finite width and height. The simplest approach is to divide the aperture into little squares and to consider each square as a source of spherical waves. Similarly we can divide the screen or photographic plate into small regions or cells and calculate the time averaged intensity at the center of each cell. The calculations are straightforward, but time consuming, because of the necessity of evaluating the cosine function many times. The less straightforward part of the problem is deciding how to plot the different values of the calculated intensity on the screen. One way is to plot “points” at random locations in each cell so that the number of points is proportional to the computed

intensity at the center of the cell. Suggested parameters are $\lambda = 5000 \text{ \AA}$ and $L = 200 \text{ mm}$ for an aperture of dimensions $1\text{mm} \times 3 \text{ mm}$.

Appendix 9A: Fast Fourier Transform

The fast Fourier transform (FFT) has been discovered independently by many workers in a variety of contexts, and there are a number of variations on the basic algorithm. In the following, we describe a version due to Danielson and Lanczos. The goal is to compute the Fourier transform $g(\omega_k)$ given the data set $f(j) \equiv f_j$ of (??). For convenience we rewrite the relation:

$$g_k \equiv g(\omega_k) = \sum_{j=0}^{N-1} f(j\Delta) e^{-i2\pi kj/N}, \quad (9.54)$$

and introduce the complex number W given by

$$W = e^{-i2\pi/N}. \quad (9.55)$$

The following algorithm works with any complex data set if we require that N is a power of two. Real data sets can be transformed by setting the array elements corresponding to the imaginary part equal to 0.

To understand the FFT algorithm, we consider the case $N = 8$, and rewrite (??) as

$$g_k = \sum_{j=0,2,4,8} f(j\Delta) e^{-i2\pi kj/N} + \sum_{j=1,3,5,7} f(j\Delta) e^{-i2\pi kj/N} \quad (9.56)$$

$$= \sum_{j=0,1,2,3} f(2j\Delta) e^{-i2\pi k2j/N} + \sum_{j=0,1,2,3} f((2j+1)\Delta) e^{-i2\pi k(2j+1)/N} \quad (9.57)$$

$$= \sum_{j=0,1,2,3} f(2j\Delta) e^{-i2\pi kj/(N/2)} + W^k \sum_{j=0,1,2,3} f((2j+1)\Delta) e^{-i2\pi kj/(N/2)} \quad (9.58)$$

$$= g_k^e + W^k g_k^o, \quad (9.59)$$

where $W^k = e^{-i2\pi k/N}$. The quantity g^e is the Fourier transform of length $N/2$ formed from the even components of the original $f(j)$; g^o is the Fourier transform of length $N/2$ formed from the odd components.

Of course, we do not have to stop here, and we can continue this decomposition if N is a power of two. That is, we can decompose g^e into its $N/4$ even and $N/4$ odd components, g^{ee} and g^{eo} , and decompose g^o into its $N/4$ even and $N/4$ odd components, g^{oe} and g^{oo} . We find

$$g_k = g_k^{ee} + W^{2k} g_k^{eo} + W^k g_k^{oe} + W^{3k} g_k^{oo}. \quad (9.60)$$

One more decomposition leads to

$$\begin{aligned} g_k = & g_k^{eee} + W^{4k} g_k^{eao} + W^{2k} g_k^{eo} + W^{6k} g_k^{eo} \\ & + W^k g_k^{oee} + W^{5k} g_k^{oao} + W^{3k} g_k^{oe} + W^{7k} g_k^{ooo}. \end{aligned} \quad (9.61)$$

At this stage each of the Fourier transforms in (??) use only one data point. We see from (??) with $N = 1$, that the value of each of these Fourier transforms, $g_k^{\text{eee}}, g_k^{\text{eoo}}, \dots$, is equal to the value of f at the corresponding data point. Note that for $N = 8$, we have performed $3 = \log_2 N$ decompositions. In general, we would perform $\log_2 N$ decompositions.

There are two steps to the FFT. First, we reorder the components so that they appear in the order given in (??). This step makes the subsequent calculations easier to organize. To see how to do the reordering, we rewrite (??) using the values of f :

$$\begin{aligned} g_k = & f(0) + W^{4k}f(4\Delta) + W^{2k}f(2\Delta) + W^{6k}f(6\Delta) \\ & + W^kf(\Delta) + W^{5k}f(5\Delta) + W^{3k}f(3\Delta) + W^{7k}f(7\Delta). \end{aligned} \quad (9.62)$$

We use a trick to obtain the ordering in (??) from the original order $f(0\Delta), f(1\Delta), \dots, f(7\Delta)$. Part of the trick is to refer to each g in (??) by a string of ‘e’ and ‘o’ characters. We assign 0 to ‘e’ and 1 to ‘o’ so that each string represents the binary representation of a number. If we reverse the order of the representation, i.e., set 110 to 011, we obtain the value of f we want. For example, the fifth term in (??) contains g^{oee} corresponding to the binary number 100. The reverse of this number is 001 which equals 1 in decimal notation, and hence the fifth term in (??) contains the function $f(1)$. Convince yourself that this bit reversal procedure works for the other seven terms.

The first step in the FFT algorithm is to use this bit reversal procedure on the original array representing the data. In the next step this array is replaced by its Fourier transform. If you want to save your original data, it is necessary to first copy the data to another array before passing the array to a FFT subroutine. The Danielson-Lanczos algorithm involves three loops. The outer loop runs over $\log_2 N$ steps. For each of these steps, N calculations are performed in the two inner loops. As can be seen in **SUB FFT**, in each pass through the innermost loop each element of the array g is updated once by the quantity **temp** formed from a power of W multiplied by the current value of an appropriate element of g . The power of W used in **temp** is changed after each pass through the innermost loop. The power of the FFT algorithm is that we do not separately multiply each $f(j\Delta)$ by the appropriate power of W . Instead, we first take pairs of $f(j)$ and multiply them by an appropriate power of W to create new values for the array g . Then we repeat this process for pairs of the new array elements (each array element now contains four of the $f(j\Delta)$). We repeat this process until each array element contains a sum of all N values of $f(j)$ with the correct powers of W multiplying each term to form the Fourier transform.

```
SUB FFT(g(,),p)
  ! fast Fourier transform of complex input data set g
  ! transform returned in g
  DIM Wp(2),factor(2),temp(2)
  LET N = 2^p                      ! number of data points
  LET N2 = N/2
  ! rearrange input data according to bit reversal
  LET j = 1
  FOR i = 1 to N-1
    ! g(i,1) is real part of f((i-1)*del_t)
    ! g(i,2) is imaginary part of f((i-1)*del_t)
    ! set g(i,2) = 0 if data real
    IF i < j then                  ! swap values
```

```

LET temp(1) = g(j,1)
LET temp(2) = g(j,2)
LET g(j,1) = g(i,1)
LET g(j,2) = g(i,2)
LET g(i,1) = temp(1)
LET g(i,2) = temp(2)
END IF
LET k = N2
DO while k < j
    LET j = j-k
    LET k = k/2
LOOP
LET j = j + k
NEXT i
! begin Danielson-Lanczos algorithm
LET jmax = 1
FOR L = 1 to p
    LET del_i = 2*jmax
    LET Wp(1) = 1           ! Wp initialized at W^0
    LET Wp(2) = 0
    LET angle = pi/jmax
    LET factor(1) = cos(angle)      ! ratio of new to old W^p
    LET factor(2) = -sin(angle)
    FOR j = 1 to jmax
        FOR i = j to N step del_i
            ! calculate transforms of length 2^L
            LET ip = i + jmax
            LET temp(1) = g(ip,1)*Wp(1) - g(ip,2)*Wp(2)
            LET temp(2) = g(ip,1)*Wp(2) + g(ip,2)*Wp(1)
            LET g(ip,1) = g(i,1) - temp(1)
            LET g(ip,2) = g(i,2) - temp(2)
            LET g(i,1) = g(i,1) + temp(1)
            LET g(i,2) = g(i,2) + temp(2)
        NEXT i
        ! find new W^p
        LET temp(1) = Wp(1)*factor(1) - Wp(2)*factor(2)
        LET temp(2) = Wp(1)*factor(2) + Wp(2)*factor(1)
        MAT Wp = temp
    NEXT j
    LET jmax = del_i
NEXT L
END SUB

```

Exercise 9.24. Testing the FFT algorithm

1. Test sub FFT for $N = 8$ by going through the code by hand and showing that the subroutine reproduces (??).

2. Write a subroutine to do the Fourier transform in the conventional manner based on (??). Make sure that your subroutine has the same arguments as `SUB FFT`, that is, write a subroutine to convert $f(j\Delta)$ to the two-dimensional array `g`. If the data is real, let $g(i,2) = 0$. The subroutine should consist of two nested loops, one over k and one over j . Print the Fourier transform of random real values of $f(j\Delta)$ for $N = 8$, using both `SUB FFT` and the direct computation of the Fourier transform. Compare the two sets of data to insure that there are no errors in `SUB FFT`. Repeat for a random collection of complex data points.
3. Compute the CPU time as a function of N for $N = 16, 64, 256$, and 1024 for the FFT algorithm and the direct computation. You can use the `time` function in True BASIC before and after each call to the subroutines. Verify that the dependence on N is what you expect.
4. Modify `SUB FFT` to compute the inverse Fourier transform defined by (??). The inverse Fourier transform of a Fourier transformed data set should be the original data set.

References and Suggestions for Further Reading

- E. Oran Brigham, *The Fast Fourier Transform*, Prentice Hall (1988). A classic text on Fourier transform methods.
- David C. Champeney, *Fourier Transforms and Their Physical Applications*, Academic Press (1973).
- James B. Cole, Rudolph A. Krutar, Susan K. Numrich, and Dennis B. Creamer, “Finite-difference time-domain simulations of wave propagation and scattering as a research and educational tool,” *Computers in Physics* **9**, 235 (1995).
- Frank S. Crawford, *Waves*, Berkeley Physics Course, Vol. 3, McGraw-Hill (1968). A delightful book on waves of all types. The home experiments are highly recommended. One observation of wave phenomena equals many computer demonstrations.
- Paul DeVries, *A First Course in Computational Physics*, John Wiley & Sons (1994). Part of our discussion of the wave equation is based on Chapter 7. There also are good sections on the numerical solution of other partial differential equations, Fourier transforms, and the FFT.
- N. A. Dodd, “Computer simulation of diffraction patterns,” *Phys. Educ.* **18**, 294 (1983).
- P. G. Drazin and R. S. Johnson, *Solitons: an Introduction*, Cambridge (1989). This book focuses on analytical solutions to the Korteweg-de Vries equation which has soliton solutions.
- Richard P. Feynman, Robert B. Leighton, and Matthew Sands, *The Feynman Lectures on Physics*, Vol. 1, Addison-Wesley (1963). Chapters relevant to wave phenomena include Chapters 28–30 and Chapter 33.
- A. P. French, *Vibrations and Waves*, W. W. Norton & Co. (1971). An introductory level text that emphasizes mechanical systems.

Eugene Hecht, *Optics*, second edition, Addison-Wesley & Sons (1987). An intermediate level optics text that emphasizes wave concepts.

Akira Hirose and Karl E. Lonngren, *Introduction to Wave Phenomena*, John Wiley & Sons (1985). An intermediate level text that treats the general properties of waves in various contexts.

Amy Kolan, Barry Cipra, and Bill Titus, “Exploring localization in nonperiodic systems,” *Computers in Physics* **9**, to be published (1995). An elementary discussion of how to solve the problem of a chain of coupled oscillators with disorder using transfer matrices.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes*, second edition, Cambridge University Press (1992). See Chapter 12 for a discussion of the fast Fourier transform.

Timothy J. Rolfe, Stuart A. Rice, and John Dancz, “A numerical study of large amplitude motion on a chain of coupled nonlinear oscillators,” *J. Chem. Phys.* **70**, 26 (1979). Problem ?? is based on this paper.

Garrison Sposito, *An Introduction to Classical Dynamics*, John Wiley & Sons (1976). A good discussion of the coupled harmonic oscillator problem is given in Chapter 6.

William J. Thompson, *Computing for Scientists and Engineers*, John Wiley & Sons (1992). See Chapters 9 and 10 for a discussion of Fourier transform methods.

Michael L. Williams and Humphrey J. Maris, “Numerical study of phonon localization in disordered systems,” *Phys. Rev. B* **31**, 4508 (1985). The authors consider the normal modes of a two-dimensional system of coupled oscillators with random masses. The idea of using mechanical resonance to extract the normal modes is the basis of a new numerical method for finding the eigenmodes of large lattices. See Kousuke Yukubo, Tsuneyoshi Nakayama, and Humphrey J. Maris, “Analysis of a new method for finding eigenmodes of very large lattice systems,” *J. Phys. Soc. Japan* **60**, 3249 (1991).

Chapter 10

Electrodynamics

©2000 by Harvey Gould and Jan Tobochnik
12 December 2000

We compute the electric fields due to static and moving charges, describe methods for computing the electric potential in boundary value problems, and solve Maxwell's equations numerically.

10.1 Static Charges

Suppose we want to know the electric field $\mathbf{E}(\mathbf{r})$ at the point \mathbf{r} due to N point charges q_1, q_2, \dots, q_N at fixed positions r_1, r_2, \dots, r_N . We know that $\mathbf{E}(\mathbf{r})$ satisfies a superposition principle and is given by

$$\mathbf{E}(\mathbf{r}) = K \sum_i^N \frac{q_i}{|\mathbf{r} - \mathbf{r}_i|^3} (\mathbf{r} - \mathbf{r}_i), \quad (10.1)$$

where \mathbf{r}_i is the fixed location of the i th charge and K is a constant that depends on the choice of units. One of the difficulties associated with electrodynamics is the competing systems of units. In the SI (or rationalized MKS) system of units, the charge is measured in coulombs (C) and the constant K is given by

$$K = \frac{1}{4\pi\epsilon_0} \approx 9.0 \times 10^9 \text{ N} \cdot \text{m}^2/\text{C}^2. \quad (\text{SI units}) \quad (10.2)$$

The constant ϵ_0 is known as the electrical permittivity of free space. This choice of units is not convenient for computer programs because $K \gg 1$. Another popular system of units is the Gaussian (cgs) system for which the constant K is absorbed into the unit of charge so that $K = 1$. Charge is in "electrostatic units" or esu. One feature of Gaussian units is that the electric and magnetic fields have the same units. For example, the (Lorentz) force on a particle of charge q and velocity \mathbf{v} in an electric field \mathbf{E} and a magnetic field \mathbf{B} has the form

$$\mathbf{F} = q(\mathbf{E} + \frac{\mathbf{v}}{c} \times \mathbf{B}). \quad (\text{Gaussian units}) \quad (10.3)$$

These virtues of the Gaussian system of units lead us to adopt this system for this chapter even though SI units are used in introductory texts.

The usual way of visualizing the electric field is to draw *electric field lines*. The properties of these lines are as follows:

1. An electric field line is a directed line whose tangent at every position is parallel to the electric field at that position.
2. The lines are smooth and continuous except at singularities such as point charges. (It makes no sense to talk about the electric field *at* a point charge.)
3. The density of lines at any point in space is proportional to the magnitude of the field at that point. This property implies that the total number of electric field lines from a point charge is proportional to the magnitude of that charge. The value of the proportionality constant is chosen to provide the clearest pictorial representation of the field. The drawing of field lines is art plus science.

Program `fieldline`, which is listed in the following, draws electric field lines in two dimensions starting at positive charges if the net charge $q_{\text{net}} \geq 0$ or at negative charges if $q_{\text{net}} < 0$. The program implements the following algorithm:

1. Begin at a point (x, y) near a charge and compute the components E_x and E_y of the electric field vector \mathbf{E} using (9.1).
2. Draw a small line segment of size $\Delta s = |\Delta \mathbf{s}|$ tangent to \mathbf{E} at that point. If $q_{\text{net}} \geq 0$, then $\Delta s > 0$, otherwise $\Delta s < 0$. The components of the line segment are given by

$$\Delta x = \Delta s \frac{E_x}{|\mathbf{E}|} \text{ and } \Delta y = \Delta s \frac{E_y}{|\mathbf{E}|}. \quad (10.4)$$

The program uses a small value for Δs if the field line is close to the charges or if the field magnitude is large. To speed up the program, a large value of Δs is used when a field line moves off the screen and the field has a small magnitude.

3. Repeat the process beginning at the new point $(x + \Delta x, y + \Delta y)$. Continue until the field line approaches another charge.
4. Repeat steps (1)–(3) for equally spaced starting positions on a circle around the charge. The spacing is inversely proportional to the magnitude of the charge.
5. Repeat steps (1)–(4) for each charge of the same sign.

Program `fieldline` draws the correct density of lines in most cases. In some cases a field line will go far away from the charges, but eventually return to a negative charge. Because such a field line might take too much time to draw, the user can hit any key to stop drawing this field line. There are some other situations where the algorithm breaks down. For example, if a field line is directed along the line connecting two charges that are equal in magnitude and opposite in sign, and begins by going away from both charges, then the field line will never return. Note the use of the `GET POINT` statement which allows the user to point to a position in the current window and click on the mouse to select it.

```

PROGRAM fieldline
! draw electric field lines in two dimensions
LIBRARY "csgraphics"
DIM x(10),y(10),q(10)
CALL screen(a,pos$,neg$)
CALL charges(N,x(),y(),q(),qtotal,a,pos$,neg$)
! draw field lines
CALL draw_lines(N,x(),y(),q(),qtotal,a)
END

SUB screen(a,pos$,neg$)
LET L = 10
CALL compute_aspect_ratio(L,xwin,ywin)
SET WINDOW -xwin,xwin,-ywin,ywin
LET a = 0.2           ! "radius" of visual image of charges
SET COLOR "blue"
BOX CIRCLE -a,a,-a,a
FLOOD 0,0
BOX KEEP -a,a,-a,a in pos$
CLEAR
SET COLOR "red"
BOX CIRCLE -a,a,-a,a
FLOOD 0,0
BOX KEEP -a,a,-a,a in neg$
CLEAR
END SUB

SUB charges(N,x(),y(),q(),qtotal,a,pos$,neg$)
! input charge values and location
LET N = 0           ! # of point charges
SET COLOR "black"
DO
    CALL draw_charges(N,x(),y(),q(),a,pos$,neg$)
    INPUT prompt "charge (0 to exit input mode) = ": charge
    IF charge <> 0 then
        PRINT "place mouse at charge location and click."
        LET N = N + 1
        GET POINT x(N),y(N)      ! location of charge
        LET q(N) = charge
        LET qtotal = qtotal + charge
    END IF
    CLEAR
LOOP until charge = 0
! redraw charges
CALL draw_charges(N,x(),y(),q(),a,pos$,neg$)
END SUB

```

```

SUB draw_charges(N,x(),y(),q(),a,pos$,neg$)
  FOR i = 1 to N
    IF q(i) > 0 then
      BOX SHOW pos$ at x(i)-a,y(i)-a
    ELSE
      BOX SHOW neg$ at x(i)-a,y(i)-a
    END IF
  NEXT i
END SUB

SUB draw_lines(N,x(),y(),q(),qtotal,a)
  ! number of lines per unit charge leaving positive charges
  SET COLOR "black"
  LET lpc = 8           ! lines per charge
  LET Emin = 0.01        ! if E < Emin stop drawing fieldline
  LET sign = sgn(qtotal)
  IF sign = 0 then LET sign = 1
  LET ds_small = 0.01*sign
  LET ds_big = sign
  FOR i = 1 to N          ! loop over all charges
    IF q(i)*sign > 0 then    ! start fields at positive charges
      LET dtheta = 2*pi/(lpc*abs(q(i)))
      FOR theta = 0 to 2*pi step dtheta
        LET stop_plot$ = "no"
        LET xline = x(i) + a*cos(theta)
        LET yline = y(i) + a*sin(theta)
        DO
          LET Ex = 0
          LET Ey = 0
          FOR j = 1 to N
            ! x-distance from point to charge j
            LET dx = xline - x(j)
            ! y-distance from point to charge j
            LET dy = yline - y(j)
            LET r = sqr(dx*dx + dy*dy)
            IF r > 0.9*a then
              LET E0 = q(j)/(r*r*r)
              LET Ex = Ex + E0*dx
              LET Ey = Ey + E0*dy
            ELSE
              ! field line reached another charge
              LET stop_plot$ = "yes"
            END IF
          NEXT j
          LET E = sqr(Ex*Ex + Ey*Ey)
        
```

```

IF E > Emin or r < 20 then
    ! new position on fieldline
    LET xline = xline + ds_small*Ex/E
    LET yline = yline + ds_small*Ey/E
ELSE
    ! new position on fieldline
    LET xline = xline + ds_big*Ex/E
    LET yline = yline + ds_big*Ey/E
END IF
PLOT xline,yline;
IF key input then
    ! user can stop drawing field line
    GET KEY key
    LET stop_plot$ = "yes"
END IF
LOOP until stop_plot$ = "yes"
PLOT          ! turn beam off
NEXT theta
END IF
NEXT i
END SUB

```

Problem 10.1. Electric field lines from point charges

1. Program **fieldline** is written so that the user inputs the value of the charge from the keyboard and its position using the mouse. Enter some simple charge configurations and check that the program is working properly. Then let $q(1) = 1$, $q(2) = -4$, and $q(3) = 3$, and place the three charges at the vertices of an approximate equilateral triangle. Are the units of charge and distance relevant? Remember that the number of field lines entering a negative charge might not be correct. Verify that the field lines never connect charges of the same sign. Why do field lines never cross?
2. Modify **SUB charge** so that values of $q(i)$, $x(i)$, and $y(i)$ can be assigned without entering them from the keyboard. In this way we can assign their values more precisely. Draw the field lines for an electric dipole.
3. Draw the field lines for the electric quadrupole with $q(1) = 1$, $x(1) = 1$, $y(1) = 1$, $q(2) = -1$, $x(2) = -1$, $y(2) = 1$, $q(3) = 1$, $x(3) = -1$, $y(3) = -1$, and $q(4) = -1$, $x(4) = 1$, and $y(4) = -1$.
4. A continuous charge distribution can be approximated by a large number of closely spaced point charges. Draw the electric field lines due to a row of ten equally spaced unit charges located between -2.5 and $+2.5$ on the x axis. How does the electric field distribution compare to the distribution due to a single point charge?
5. Repeat part (d) with two rows of equally spaced positive charges on the lines $y = 0$ and $y = 1$, respectively. Then consider one row of positive charges and one row of negative charges.

6. Another way of representing the electric field is to divide space into a discrete grid and to draw arrows in the direction of \mathbf{E} at the vertices of the grid. The magnitude of the arrow can be chosen to be proportional to the magnitude of the electric field. Another possibility is to use color or gray scale to represent the magnitude. Which representation do you think conveys more information?

Problem 10.2. Field lines due to infinite line of charge

1. **Program fieldline** plots field lines in two dimensions. Sometimes this restriction can lead to spurious results (see Freeman). Consider four identical charges placed at the corners of a square. Use **Program fieldline** to plot the field lines. What is wrong with the results? What should happen to the field lines near the center of the square?
2. The two-dimensional analog of a point charge is an infinite line of charge perpendicular to the plane. The electric field due to an infinite line of charge is proportional to the linear charge density and inversely proportional to the distance (instead of the distance squared) from the line of charge to a point in the plane. Modify the calculation of the electric field in **Program fieldline** so that field lines from infinite lines of charge are drawn. Use your program to draw the field lines due to four identical infinite lines of charge located at the corners of a square, and compare the field lines with your results in part (a).
3. Use your modified program from part (b) to draw the field lines for the two-dimensional analogs of the distributions considered in Problem 9.1. Compare the results for two and three dimensions, and discuss any qualitative differences.

Problem 10.3. Motion of a charged particle in an electric field

1. Write a program to compute the motion of a particle of mass m and charge q in the presence of the electric field created by a fixed distribution of point charges. Use the Euler-Richardson algorithm to update the position and velocity of the particle. The acceleration of the charge is given by $q\mathbf{E}/m$, where \mathbf{E} is the electric field due to the fixed point charges. (We ignore the effects of radiation due to accelerating charges.) Incorporate the relevant subroutines from **Program fieldline** to visualize the electric field.
2. Assume that \mathbf{E} is due to a charge $q(1) = 1.5$ fixed at the origin. Simulate the motion of a charged particle of mass $m = 0.1$ and charge $q = 1$ initially at $x = 1, y = 0$. Consider the following initial conditions for its velocity: (i) $v_x = 0, v_y = 0$; (ii) $v_x = 1, v_y = 0$; (iii) $v_x = 0, v_y = 1$; and (iv) $v_x = -1, v_y = 0$. Draw electric field lines beginning at the initial values of (x, y) and a line representing the particle's trajectory. Why does the trajectory of the particle not always follow a field line?
3. Assume that the electric field is due to two fixed point charges: $q(1) = 1$ at $x(1) = 2, y(1) = 0$ and $q(2) = -1$ at $x(2) = -2, y(2) = 0$. Place a charged particle of unit mass and unit positive charge at the point $x = 0.05, y = 0$. What do you expect the motion of this charge to be? Do the simulation and determine the qualitative nature of the motion.
4. Consider the motion of a charged particle in the vicinity of the electric dipole defined in part (c). Choose the initial position to be five times the separation of the charges in the dipole. Do you find any bound orbits? Can you find any closed orbits or do all orbits show some precession?

We know that it is often easier to analyze the behavior of a system using energy rather than force concepts. We define the electric potential $V(\mathbf{r})$ by the relation

$$V(\mathbf{r}_2) - V(\mathbf{r}_1) = - \int_{\mathbf{r}_1}^{\mathbf{r}_2} \mathbf{E} \cdot d\mathbf{r} \quad (10.5)$$

or

$$\mathbf{E}(\mathbf{r}) = -\nabla V(\mathbf{r}). \quad (10.6)$$

Only differences in the potential between two points have physical significance. The gradient operator ∇ is given in Cartesian coordinates by

$$\nabla = \frac{\partial}{\partial x} \hat{\mathbf{x}} + \frac{\partial}{\partial y} \hat{\mathbf{y}} + \frac{\partial}{\partial z} \hat{\mathbf{z}}, \quad (10.7)$$

where the vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ are unit vectors along the x , y , and z axes respectively. If V depends only on the magnitude of \mathbf{r} , then (9.6) becomes $E(r) = -dV(r)/dr$. Recall that $V(r)$ for a point charge q relative to a zero potential at infinity is given by

$$V(r) = \frac{q}{r}. \quad (\text{Gaussian units}) \quad (10.8)$$

The surface on which the electric potential has an equal value everywhere is called an *equipotential surface* (curve in two dimensions). Because \mathbf{E} is in the direction in which the electric potential decreases most rapidly, the electric field lines are orthogonal to the equipotential surfaces at any point. We can use this relation between the electric field lines and the equipotential lines to modify **Program fieldline** so that it draws the latter. Because the components of the line segment $\Delta\mathbf{s}$ parallel to the electric field line are given by $\Delta x = \Delta s(E_x/E)$ and $\Delta y = \Delta s(E_y/E)$, the components of the line segment perpendicular to \mathbf{E} , and hence parallel to the equipotential line, are given by $\Delta x = -\Delta s(E_y/E)$ and $\Delta y = \Delta s(E_x/E)$. It is unimportant whether the minus sign is assigned to the x or y component, because the only difference would be the direction that the equipotential lines are drawn.

Problem 10.4. Equipotential lines

1. Modify **Program fieldline** to draw some of the equipotential lines for the charge distributions considered in Problem 9.1. One way to do so is to add a subroutine that uses a mouse click to determine the initial position of an equipotential line. The following code determines this position from a mouse click.

```
DO
  GET MOUSE x0,y0,s
LOOP until s <> 0
```

The variable s has the value 0 until the mouse is clicked. After a mouse click, the subroutine should draw the equipotential line starting from x_0, y_0 and ending when the line closes on itself.

2. What would a higher density of equipotential lines mean if we drew lines such that each adjacent line differed from a neighboring one by a fixed potential difference?
3. Explain why equipotential surfaces never cross.

Problem 10.5. The electric potential due to a finite sheet of charge

Consider a uniformly charged dielectric plate of total charge Q and linear dimension L centered at $(0, 0, 0)$ in the x - y plane. In the limit $L \rightarrow \infty$ with the charge density $\sigma = Q/L^2$ a constant, we know that the electric field is normal to the sheet and is given by $E_n = 2\pi\sigma$ (Gaussian units). What is the electric field due to a finite sheet of charge? A simple method is to divide the plate into a grid of p cells on a side such that each cell is small enough to be approximated by a point charge of magnitude $q = Q/p^2$. Because the potential is a scalar quantity, it is easier to compute the total potential rather than the total electric field from the $N = p^2$ point charges. Use the relation (9.8) for the potential from a point charge and write a program to compute $V(z)$ and hence $E_z = -\partial V(z)/\partial z$ for points along the z -axis and perpendicular to the sheet. Take $L = 1$, $Q = 1$, and $p = 10$ for your initial calculations. Increase p until your results for $V(z)$ do not change significantly. Plot $V(z)$ and E_z as a function of z and compare their z -dependence to their infinite sheet counterparts.

**Problem 10.6.* Electrostatic shielding

We know that the (static) electric field is zero inside a conductor, all excess charges reside on the surface of the conductor, and the surface charge density is greatest at the points of greatest curvature. Although these properties are plausible, it is instructive to do a simulation to see how these properties follow from Coulomb's law. For simplicity, consider the conductor to be two-dimensional so that the potential energy is proportional to $\ln r$ rather than $1/r$ (see Problem 9.2). It also is convenient to choose the surface of the conductor to be an ellipse.

1. If we are interested only in the final distribution of the charges and not in the dynamics of the system, we can use a Monte Carlo method. Our goal is to find the minimum energy configuration beginning with the N charges randomly placed within the ellipse. The method is to choose a charge i at random, and make a trial change in the position of the charge. The trial position should be no more than d_{\max} from the old position and still within the ellipse. The parameter d_{\max} should be chosen to be approximately $b/10$, where b is the semiminor axis of the ellipse. Compute the change in the total potential energy given by (in arbitrary units)

$$\Delta U = - \sum_j [\ln r_{ij}^{(\text{new})} - \ln r_{ij}^{(\text{old})}]. \quad (10.9)$$

The sum is over all charges in the system not including i . If $\Delta U > 0$, then reject the trial move, otherwise accept it. Repeat this procedure many times until very few trial moves are accepted. Write a program to implement this Monte Carlo algorithm. Run the simulation for $N \geq 20$ charges inside a circle and then repeat the simulation for an ellipse. How are the charges distributed in the (approximately) minimum energy distribution? Which parts of the ellipse have a higher charge density?

2. Repeat part (a) for a two-dimensional conductor, but assume that the potential energy $U \sim 1/r$. Do the charges move to the surface? Is it sufficient that the interaction be repulsive?

3. Repeat part (a) with the added condition that there is a fixed positive charge of magnitude $N/2$ located outside the ellipse. How does this fixed charge effect the charge distribution? Are the excess free charges still at the surface? Try different positions for the fixed charge.
4. Repeat parts (a) and (b) for $N = 50$ charges located within an ellipsoid in three dimensions.

10.2 Numerical Solutions of Laplace's Equation

In Section ?? we found the electric fields and potentials due to a fixed distribution of charges. Suppose that we do not know the positions of the charges and instead know only the potential on a set of boundaries surrounding a charge-free region. This information is sufficient to determine the potential $V(\mathbf{r})$ at any point within the charge-free region.

The direct method of solving for $V(x, y, z)$ is based on Laplace's equation which can be expressed in Cartesian coordinates as

$$\nabla^2 V(x, y, z) \equiv \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} = 0. \quad (10.10)$$

The problem is to find the function $V(x, y, z)$ that satisfies (9.10) and the specified boundary conditions. This type of problem is an example of a *boundary value* problem. Because analytical methods for regions of arbitrary shape do not exist, the only general approach is to use numerical methods.

Laplace's equation is not a new law of physics, but can be derived directly from (9.6) and the relation $\nabla \cdot \mathbf{E} = 0$ or indirectly from Coulomb's law in regions of space where there is no charge. For simplicity, we consider only two-dimensional boundary value problems for $V(x, y)$. We use a finite difference method and divide space into a discrete grid of points. In the following, we show that in the absence of a charge at (x, y) , the discrete form of Laplace's equation satisfies the relation

$$V(x, y) \approx \frac{1}{4}[V(x + \Delta x, y) + V(x - \Delta x, y) + V(x, y + \Delta y) + V(x, y - \Delta y)]. \quad (\text{two dimensions}) \quad (10.11)$$

That is, $V(x, y)$ is the average of the potential at the four nearest neighbor points. This remarkable property of $V(x, y)$ can be derived by approximating the partial derivatives in (9.10) by finite differences (see Problem 9.7b).

In Problem 9.7a we verify (9.12) by calculating the potential due to a point charge at a point in space selected by the user and at the four nearest neighbors. As the form of (9.12) implies, the average of the potential at the four neighboring points should equal the potential at the center point. We assume the form (9.8) for the potential $V(r)$ due to a point charge, a form that satisfies Laplace's equation for $r \neq 0$. The program is listed in the following.

```
PROGRAM verify
! verify the discrete form of Laplace's equation
! for a point charge on a square lattice
LIBRARY "csgraphics"
```

```

CALL initial(L,dx,dy,#1,#2)
CALL draw_grid(L,dx,dy,#3)
DO
    CALL potential(L,dx,dy,stop_plot$,#1,#2,#3)
LOOP until stop_plot$ = "yes"
END

SUB initial(L,dx,dy,#1,#2)
    INPUT prompt "lattice dimension = ": L
    INPUT prompt "grid spacing in x direction = ": dx
    INPUT prompt "grid spacing in y direction = ": dy
    OPEN #1: screen 0,0.25,0.8,1
    SET WINDOW -1.2,2,-1,2
    OPEN #2: screen 0,0.25,0,0.6
END SUB

SUB draw_grid(L,dx,dy,#1)
    OPEN #1: screen 0.25,1,0,1
    CALL compute_aspect_ratio(L,xwin,ywin)
    SET WINDOW -xwin,xwin,-ywin,ywin
    LET a = 0.2*dx           ! "radius" of visual image of charge
    SET COLOR "blue"
    BOX CIRCLE -a,a,-a,a
    FLOOD 0,0
    SET COLOR "black"
    FOR y = -L to L step dy
        FOR x = -L to L step dx
            PLOT POINTS: x,y
        NEXT x
    NEXT y
    BOX LINES -L,L,-L,L
END SUB

SUB potential(L,dx,dy,stop_plot$,#1,#2,#3)
    WINDOW #3
    SET CURSOR 1,20
    PRINT "click on mouse to choose site"
    LET s = 0
    DO
        GET MOUSE xs,ys,s
    LOOP until s = 2
    WINDOW #1
    CLEAR
    LET stop_plot$ = ""
    IF abs(xs) <= L and abs(ys) <= L then
        CALL showpotential(xs,ys,0,0,V0)

```

```

CALL showpotential(xs+dx,ys,1,0,V1)
CALL showpotential(xs-dx,ys,-1,0,V2)
CALL showpotential(xs,ys+dy,0,1,V3)
CALL showpotential(xs,ys-dy,0,-1,V4)
WINDOW #2
SET CURSOR 1,1
PRINT " average potential"
PRINT " of four neighbors:"
PRINT truncate(0.25*(V1+V2+V3+V4),4)
ELSE
  LET stop_plot$ = "yes"
END IF
END SUB

SUB showpotential(x,y,xp,yp,V)
  LET V = 1/sqr(x*x + y*y)      ! potential of a point charge
  LET V = truncate(V,4)
  PLOT TEXT, AT xp,yp: str$(V)
END SUB

```

Problem 10.7. Verification of the difference equation for the potential

1. Choose reasonable values for the grid spacings Δx and Δy and consider a point that is not too close to the source charge. Compare the computed potential at a point to the average of the potential at its four nearest neighbor points. Do similar measurements for other points. Does the relative agreement with (9.12) depend on the distance of the point to the source charge? Choose smaller values of Δx and Δy and determine if results are in better agreement with (9.12). Does it matter whether Δx and Δy have the same value?
2. Derive the finite difference equation (9.12) for $V(x, y)$ using the second-order Taylor expansion:

$$V(x + \Delta x, y) = V(x, y) + \Delta x \frac{\partial V(x, y)}{\partial x} + \frac{1}{2}(\Delta x)^2 \frac{\partial^2 V(x, y)}{\partial x^2} + \dots \quad (10.12)$$

$$V(x, y + \Delta y) = V(x, y) + \Delta y \frac{\partial V(x, y)}{\partial y} + \frac{1}{2}(\Delta y)^2 \frac{\partial^2 V(x, y)}{\partial y^2} + \dots \quad (10.13)$$

The effect of including higher derivatives is discussed by MacDonald (see references).

Now that we have found that (9.12), a finite difference form of Laplace's equation, is consistent with Coulomb's law, we adopt (9.12) as the basis for computing the potential for systems for which we cannot calculate the potential directly. In particular, we consider problems where the potential is specified on a closed surface that divides space into interior and exterior regions in which the potential is independently determined. For simplicity, we consider only two-dimensional geometries. The approach, known as the *relaxation method*, is based on the following algorithm:

1. Divide the region of interest into a rectangular grid of points spanning the region. The region is enclosed by a surface (curve in two dimensions) with specified values of the potential along the curve.
2. Assign to a boundary point the potential of the boundary nearest the point.
3. Assign all interior points an arbitrary potential (preferably a reasonable guess).
4. Compute new values for the potential V for each interior point. Each new value is obtained by finding the average of the previous values of the potential at the four nearest neighbor points.
5. Repeat step (4) using the values of V obtained in the previous iteration. This iterative process is continued until the potential at each interior point is computed to the desired accuracy.

In Problems 9.8–9.10 we use this method and its variants to compute the potential for various geometries.

```

PROGRAM laplace
! implementation of Jacobi relaxation method to solve
! Laplace's equation in rectangular geometry
DIM V(0:100,0:100)
CALL assign(V(,),nx,ny,min_change)
CALL iterate(V(,),nx,ny,min_change,iterations)
END

SUB assign(V(,),nx,ny,min_change)
! linear dimension of rectangular region
LET nx = 9      ! number of interior points in x-direction
LET ny = 9      ! number of interior points in y-direction
LET V0 = 10      ! boundary potential of rectangle
INPUT prompt "percentage change = ": min_change
LET min_change = min_change/100
! fix potential on boundary of rectangle
FOR x = 0 to nx + 1
    LET V(x,0) = V0
    LET V(x,ny+1) = V0
NEXT x
FOR y = 0 to ny + 1
    LET V(0,y) = V0
    LET V(nx+1,y) = V0
NEXT y
! guess initial values of potentials of interior sites
FOR y = 1 to ny
    FOR x = 1 to nx
        LET V(x,y) = 0.9*V0

```

```

NEXT x
NEXT y
CALL show_output(V(,),nx,ny,0)
END SUB

SUB iterate(V(,),nx,ny,min_change,iterations)
  DIM Vave(100,100)
  LET iterations = 0
  DO
    ! maximum difference of iterated potential at any site
    LET change = 0
    FOR y = 1 to ny
      FOR x = 1 to nx
        ! average of potential of neighboring cells
        LET Vave(x,y) = V(x+1,y) + V(x-1,y)
        LET Vave(x,y) = Vave(x,y) + V(x,y+1) + V(x,y-1)
        LET Vave(x,y) = 0.25*Vave(x,y)
        ! compute percentage change in potential
        IF Vave(x,y) <> 0 then
          LET dV = abs((V(x,y) - Vave(x,y))/Vave(x,y))
          IF dV > change then LET change = dV
        END IF
      NEXT x
    NEXT y
    FOR y = 1 to ny      ! update potential at each site
      FOR x = 1 to nx
        LET V(x,y) = Vave(x,y)
      NEXT x
    NEXT y
    LET iterations = iterations + 1
    CALL show_output(V(,),nx,ny,iterations)
  LOOP until change <= min_change
END SUB

SUB show_output(V(,),nx,ny,iterations)      ! print potential
  PRINT
  PRINT "iteration ="; iterations
  FOR y = 0 to ny + 1
    FOR x = nx + 1 to 0 step - 1
      PRINT using "###.##": V(x,y);
    NEXT x
    PRINT
  NEXT y
END SUB

```

Problem 10.8. Numerical solution of the potential within a rectangular region

1. Use **Program laplace** to determine the potential $V(x, y)$ in a square region with linear dimension $L = 9$. The boundary of the square is at a potential $V = 10$. Let the number of interior grid points in the horizontal and vertical directions be $\text{nx} = \text{ny} = 9$, respectively. Before you run the program, guess the exact form of $V(x, y)$ and set the initial values of the interior potential close to the exact answer. How many iterations are necessary to achieve 1% accuracy? Increase both Δx and Δy by a factor of two, and determine the number of iterations that are now necessary to achieve 1% accuracy.
2. Consider the same geometry as in part (a), but set the initial potential at the interior points equal to zero except for the center point whose potential is set equal to four. Does the potential distribution evolve to the same values as in part (a)? What is the effect of a poor initial guess? Are the final results independent of your initial guess?
3. Modify **SUB assign** in **Program laplace** so that the value of the potential at the four sides is 5, 10, 5, and 10, respectively (see Figure 9.1). Sketch the equipotential surfaces. What happens if the potential is 10 on three sides and 0 on the fourth? Start with a reasonable guess for the initial values of the potential at the interior points and iterate until 1% accuracy is obtained.
4. Consider the same initial choice of the potential as in part (b) and focus your attention on the potential at the points near the center of the square. If there were four random walkers at the central point corresponding to an initial potential of four, how many walkers would there be at the nearest neighbor points after the first iteration? Follow the distribution of the “walkers” as a function of the number of iterations and verify that the nature of the relaxation of the potential to its correct distribution is closely related to *diffusion* (see Chapters 7 and ??). It would be helpful to increase the number of points in the grid and the initial value of the potential at the central point to see the nature of the relaxation more clearly.

In Problem 9.8, we implemented a simple version of the relaxation method known as the Jacobi method. In particular, the new potential of each point is found based on the values of the potentials at the neighboring points at the previous iteration. After the entire lattice was visited, the potential at each point was updated *simultaneously*. The difficulty with this relaxation method is that it converges very slowly. The use of more general relaxation methods is discussed in many texts (cf. Koonin and Meredith or Press et al.). In Problem 9.9 we consider a method known as Gauss-Seidel relaxation.

Problem 10.9. Gauss-Seidel relaxation

1. Modify the program that you used in Problem 9.8 so that the potential at each point is updated sequentially. That is, after the average potential of the nearest neighbor points of point i is computed, update the potential at i immediately. In this way the new potential of the next point is computed using the most recently computed values of its nearest neighbor potentials. Are your results better, worse, or about the same as the simple relaxation method?
2. Imagine coloring the alternate points of a grid red and black, so that the grid resembles a checkerboard. Modify the program so that all the red points are updated first, and then all the black points are updated. This ordering is repeated for each iteration. Do your results converge any more quickly than in part (a)?

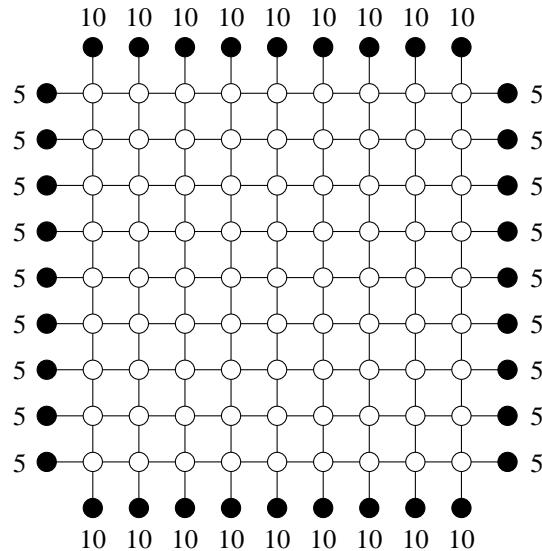


Figure 10.1: Potential distribution considered in Problem 9.8c. The number of interior points in each direction is nine.

3. The slow convergence of the relaxation methods we have explored is due to the fact that it takes a long time for a change in the potential at one point to effect changes further away. We can improve the Gauss-Seidel method by using an overrelaxation method which updates the new potential as follows:

```
LET V(x,y) = w*Vave(x,y) + (1-w)*V(x,y)
```

The overrelaxation parameter w is in the range $1 < w < 2$. The effect of w is to cause the potential to change by a greater amount than in the simple relaxation procedure. Explore the dependence of the rate of convergence on w . A relaxation method that does increase the rate of convergence is explored in Project ??.

Problem 10.10. The capacitance of concentric squares

1. Use a relaxation method to compute the potential distribution between the two concentric square cylinders shown in Figure 9.2. The potential of the outer square conductor is $V_{\text{out}} = 10$ and the potential of the inner square conductor is $V_{\text{in}} = 5$. The linear dimensions of the exterior and interior squares are $L_{\text{out}} = 25$ and $L_{\text{in}} = 5$, respectively. Modify your program so that the potential of the interior square is fixed. Sketch the equipotential surfaces.
2. A system of two conductors with charge Q and $-Q$ respectively has a capacitance C that is defined as the ratio of Q to the potential difference ΔV between the two conductors. Determine the capacitance per unit length of the concentric cylinders considered in part (a). In this case $\Delta V = 5$. The charge Q can be determined from the fact that near a conducting

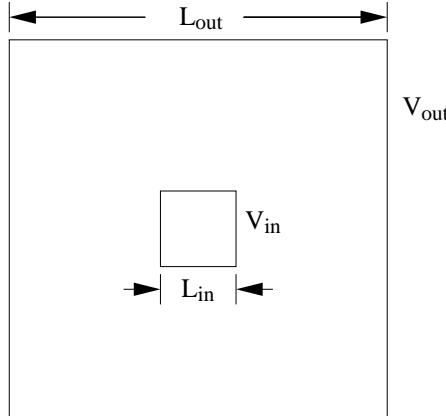


Figure 10.2: The geometry of the two concentric squares considered in Problem 9.10.

surface, the surface charge density σ is given by $\sigma = E_n / 4\pi$, where E_n is the magnitude of the electric field normal to the surface. E_n can be approximated by the relation $-\delta V/\delta r$, where δV is the potential difference between a boundary point and an adjacent interior point a distance δr away. Use the result of part (a) to compute δV for each point adjacent to the two square surfaces. Use this information to determine E_n for the two surfaces and the charge per unit length on each conductor. Are the charges equal and opposite in sign? Compare your numerical result to the capacitance per unit length, $1/2 \ln r_{\text{out}}/r_{\text{in}}$, of a system of two concentric circular cylinders of radii r_{out} and r_{in} . Assume that the circumference of each cylinder equals the perimeter of the corresponding square, i.e., $2\pi r_{\text{out}} = 4L_{\text{out}}$ and $2\pi r_{\text{in}} = 4L_{\text{in}}$.

3. Move the inner square 1 cm off center and repeat the calculations of parts (a) and (b). How do the potential surfaces change? Is there any qualitative difference if we set the inner conductor potential equal to -5 statvolt instead of $+5$ statvolt?

Laplace's equation holds only in charge-free regions. If there is a charge density $\rho(x, y, z)$ in the region, we need to use *Poisson's* equation which can be written as

$$\nabla^2 V(\mathbf{r}) = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} = -4\pi\rho(\mathbf{r}), \quad (10.14)$$

where $\rho(\mathbf{r})$ is the charge density. The difference form of Poisson's equation is given in two dimensions by

$$\begin{aligned} V(x, y) \approx & \frac{1}{4} [V(x + \Delta x, y) + V(x - \Delta x, y) + V(x, y + \Delta y) + V(x, y - \Delta y)] \\ & + \frac{1}{4} \Delta x \Delta y 4\pi\rho(x, y). \end{aligned} \quad (10.15)$$

Note that the product $\rho(x, y)\Delta x\Delta y$ is the total charge in the cell centered at (x, y) .

Problem 10.11. Numerical solution of Poisson's equation

1. Consider a square of linear dimension $L = 25$ whose boundary is fixed at a potential equal to $V = 10$. Assume that each interior cell has a uniform charge density ρ such that the total charge is $Q = 1$. Use a modification of **Program laplace** to compute the potential distribution for this case. Compare the equipotential surfaces obtained for this case to that found in Problem 9.10.
2. Find the potential distribution if the charge distribution of part (a) is restricted to a 5×5 square at the center.
3. Find the potential distribution if the charge distribution of part (a) is restricted to a 1×1 square at the center. How does the potential compare to that of a point charge without the boundary?

10.3 Random Walk Solution of Laplace's Equation

In Section 9.2 we found that the solution to Laplace's equation in two dimensions at the point (x, y) is given by

$$V(x, y) = \frac{1}{4} \sum_{i=1}^4 V(i), \quad (10.16)$$

where $V(i)$ is the value of the potential at the i th neighbor. A generalization of this result is that the potential at any point equals the average of the potential on a circle (or sphere in three dimensions) centered about that point.

The relation (9.18) can be given a probabilistic interpretation in terms of random walks (see Problem 9.8d). Suppose that many random walkers are at the point (x, y) and each walker "jumps" to one of its four neighbors (on a square grid) with equal probability $p = 1/4$. From (9.18) we see that the average potential found by the walkers after jumping one step is the potential at (x, y) . This relation generalizes to walkers that visit a point on a closed surface with fixed potential. The random walk algorithm for computing the solution to Laplace's equation can be stated as:

1. Begin at a point (x, y) where the value of the potential is desired, and take a step in a random direction.
2. Continue taking steps until the walker reaches the surface. Record $V_b(i)$, the potential at the boundary point i . A typical walk is shown in Figure 9.3.
3. Repeat steps (1) and (2) n times and sum the potential found at the surface each time.
4. The value of the potential at the point (x, y) is estimated by

$$V(x, y) = \frac{1}{n} \sum_{i=1}^n V_b(i) \quad (10.17)$$

where n is the total number of random walkers.

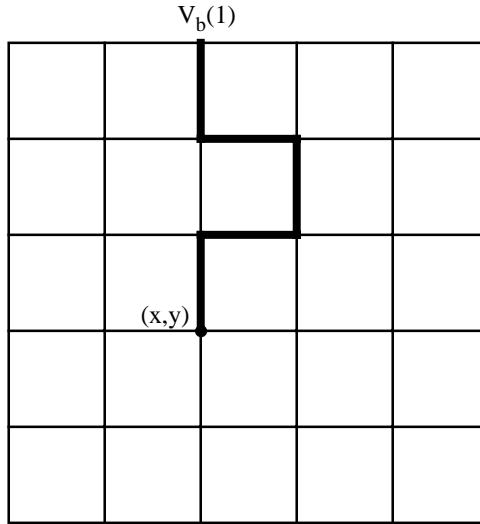


Figure 10.3: A random walk on a 6×6 grid starting at the point $(x, y) = (3, 3)$ and ending at the boundary point $V_b(3, 6)$ where the potential is recorded.

Problem 10.12. Random walk solution of Laplace's equation

1. Consider the square region shown in Figure 9.1 and compare the results of the random walk method with the results of the relaxation method (see Problem 9.8c). Try $n = 100$ and $n = 1000$ walkers, and choose a point near the center of the square.
2. Repeat part (a) for other points within the square. Do you need more or less walkers when the potential near the surface is desired? How quickly do your answers converge as a function of n ?

The disadvantage of the random walk method is that it requires many walkers to obtain a good estimate of the potential at each point. However, if the potential is needed at only a small number of points, then the random walk method might be more appropriate than the relaxation method which requires the potential to be computed at all points within the region. Another case where the random walk method is appropriate is when the geometry of the boundary is fixed, but the potential in the interior for a variety of different boundary potentials is needed. In this case the quantity of interest is $G(x, y, x_b, y_b)$, the number of times that a walker from the point (x, y) lands at the boundary (x_b, y_b) . The random walk algorithm is equivalent to the relation

$$V(x, y) = \frac{1}{n} \sum_b G(x, y, x_b, y_b) V(x_b, y_b), \quad (10.18)$$

where the sum is over all points on the boundary. We can use the same function G for different distributions of the potential on a given boundary. G is an example of a Green's function, a function that you will encounter in advanced treatments of electrodynamics and quantum mechanics

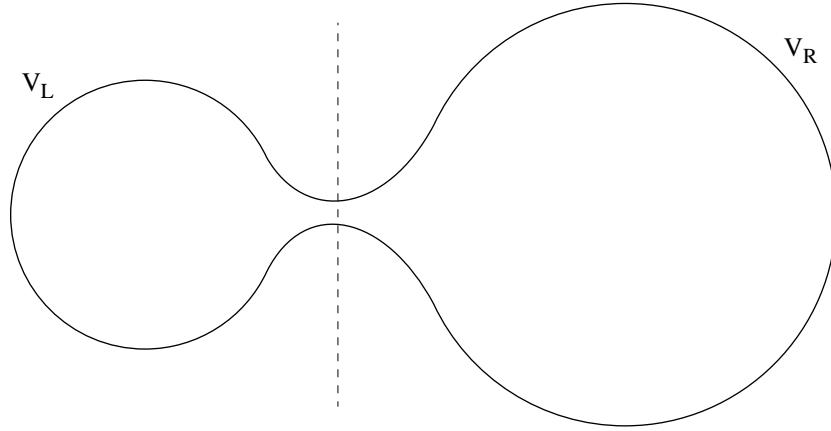


Figure 10.4: Two regions of space connected by a narrow neck. The boundary of the left region has a potential V_L , and the boundary of the right region has a potential V_R .

(cf. Section ??). Of course, if we change the geometry of the boundary, we have to recompute the function G .

Problem 10.13. Green's function solution of Laplace's equation

1. Compute the Green's function $G(x, y, x_b, y_b)$ for the same geometry considered in Problem 9.12. Use at least 200 walkers at each interior point to estimate G . Because of the symmetry of the geometry, you can determine some of the values of G from other values without doing an additional calculation. Store your results for G in a file.
2. Use your results for G found in part (a) to determine the potential at each interior point when the boundary potential is the same as in part (a) except for five boundary points which are held at $V = 20$. Find the locations of the five boundary points that maximize the potential at the interior point located at $(3, 5)$. Repeat the calculation to maximize the potential at $(5, 3)$. Use trial and error guided by your physical intuition.

The random walk algorithm can help us gain additional insight into the nature of Laplace's equation. Suppose that you have a boundary similar to the one shown in Figure 9.4. The potentials on the left and right boundaries are V_L and V_R , respectively. If the neck between the two sides is narrow, it is clear that a random walker starting on the left side has a low probability of reaching the other side. Hence, we can conclude that the potential in the interior of the left side is approximately V_L except very near the neck.

Poisson's equation also can be solved using the random walk method. In this case, the potential is given by

$$V(x, y) = \frac{1}{n} \sum_{\alpha} V(\alpha) + \frac{\pi \Delta x \Delta y}{n} \sum_{i, \alpha} \rho(x_{i, \alpha}, y_{i, \alpha}), \quad (10.19)$$

where α labels the walker, and i labels the point visited by the walker. That is, each time a walker is at a point i , we add the charge density at that point to the second sum in (9.21).

10.4 *Fields Due to Moving Charges

The fact that accelerating charges radiate electromagnetic waves is one of the most important results in the history of physics. In this section we discuss a numerical algorithm for computing the electric and magnetic fields due to the motion of charged particles. The algorithm is very general, but requires some care in its application.

To understand the algorithm, we need a few results that can be found conveniently in Feynman's lectures. We begin with the fact that the scalar potential at the observation point \mathbf{R} due to a stationary particle of charge q is

$$V(\mathbf{R}) = \frac{q}{|\mathbf{R} - \mathbf{r}|}, \quad (10.20)$$

where \mathbf{r} is the position of the charged particle. The electric field is given by

$$\mathbf{E}(\mathbf{R}) = -\frac{\partial V(\mathbf{R})}{\partial \mathbf{R}}, \quad (10.21)$$

where $\partial V(\mathbf{R})/\partial \mathbf{R}$ is the gradient with respect to the coordinates of the observation point. (Note that our notation for the observation point differs from that used in other sections in this chapter.) How do the relations (9.22) and (9.23) change when the particle is moving? We might guess that because it takes a finite time for the disturbance due to a charge to reach the point of observation, we should modify (9.22) by writing

$$V(\mathbf{R}) \stackrel{?}{=} \frac{q}{r_{\text{ret}}}, \quad (10.22)$$

where

$$r_{\text{ret}} = |\mathbf{R} - \mathbf{r}(t_{\text{ret}})|. \quad (10.23)$$

The quantity r_{ret} is the separation of the charged particle from the observation point \mathbf{R} at the retarded time t_{ret} . The latter is the time at which the particle was at $\mathbf{r}(t_{\text{ret}})$ such that a disturbance starting at $\mathbf{r}(t_{\text{ret}})$ and traveling at the speed of light would reach \mathbf{R} at time t ; t_{ret} is given by the implicit equation

$$t_{\text{ret}} = t - \frac{r_{\text{ret}}(t_{\text{ret}})}{c}, \quad (10.24)$$

where t is the observation time and c is the speed of light.

Although the above reasoning is plausible, the relation (9.24) is not quite correct (cf. Feynman et al. for a derivation of the correct result). We need to take into account that the potential due to the charge is a maximum if the particle is moving toward the observation point and a minimum if it is moving away. The correct result can be written as

$$V(\mathbf{R}, t) = \frac{q}{r_{\text{ret}}(1 - \hat{\mathbf{r}}_{\text{ret}} \cdot \mathbf{v}_{\text{ret}}/c)}, \quad (10.25)$$

where

$$\mathbf{v}_{\text{ret}} = \frac{d\mathbf{r}(t)}{dt} \Big|_{t=t_{\text{ret}}}. \quad (10.26)$$

To find the electric field of a moving charge, we recall that the electric field is related to the time rate of change of the magnetic flux. Hence, we expect that the total electric field at the observation point \mathbf{R} has a contribution due to the magnetic field created by the motion of the charge. We know that the magnetic field due to a moving charge is given by

$$\mathbf{B} = \frac{1}{c} \frac{q\mathbf{v} \times \mathbf{r}}{r^3}. \quad (10.27)$$

If we define the vector potential \mathbf{a} as

$$\mathbf{a} = \frac{q}{r} \frac{\mathbf{v}}{c}, \quad (10.28)$$

we can express \mathbf{B} in terms of \mathbf{a} as

$$\mathbf{B} = \nabla \times \mathbf{a}. \quad (10.29)$$

As we did for the scalar potential V , we argue that the correct formula for \mathbf{a} is

$$\mathbf{a}(\mathbf{R}, t) = q \frac{\mathbf{v}_{\text{ret}}/c}{r_{\text{ret}}(1 - \hat{\mathbf{r}}_{\text{ret}} \cdot \mathbf{v}_{\text{ret}}/c)}. \quad (10.30)$$

Equations (9.27) and (9.32) are known as the Liénard-Wiechert potentials.

The contribution to the electric field \mathbf{E} from V and \mathbf{a} is given by

$$\mathbf{E} = -\nabla V - \frac{1}{c} \frac{\partial \mathbf{a}}{\partial t}. \quad (10.31)$$

The derivatives in (9.33) are with respect to the observation coordinates. The difficulty associated with calculating these derivatives is that the potentials depend on t_{ret} which in turn depends on \mathbf{R} , \mathbf{r} , and t . The result can be expressed as

$$\mathbf{E}(\mathbf{R}, t) = \frac{qr_{\text{ret}}}{(\mathbf{r}_{\text{ret}} \cdot \mathbf{u}_{\text{ret}})^3} [\mathbf{u}_{\text{ret}}(c^2 - v_{\text{ret}}^2) + \mathbf{r}_{\text{ret}} \times (\mathbf{u}_{\text{ret}} \times \mathbf{a}_{\text{ret}})], \quad (10.32)$$

where

$$\mathbf{u}_{\text{ret}} \equiv c\hat{\mathbf{r}}_{\text{ret}} - \mathbf{v}_{\text{ret}}. \quad (10.33)$$

The acceleration of the particle $\mathbf{a}_{\text{ret}} = d\mathbf{v}(t)/dt|_{t=t_{\text{ret}}}$. We also can show using (9.31) that the magnetic field \mathbf{B} is given by

$$\mathbf{B} = \hat{\mathbf{r}}_{\text{ret}} \times \mathbf{E}. \quad (10.34)$$

The above discussion is not rigorous, but we can accept (9.34) and (9.36) in the same spirit as we accept Coulomb's law and the Biot-Savart law. All of classical electrodynamics can be reduced

to (9.34) and (9.36), if we assume that the sources of all fields are charges, and all electric currents are due to the motion of charged particles. Note that (9.34) and (9.36) are consistent with the special theory of relativity and reduce to known results in the limit of stationary charges and steady currents.

Although (9.34) and (9.36) are deceptively simple (we do not even have to solve any differential equations), it is difficult to calculate the fields analytically even if the position of a charged particle is an analytic function of time. The difficulty is that we must find the retarded time t_{ret} from (9.26) for each observation position \mathbf{R} and time t . For example, consider a charged particle whose motion is sinusoidal, i.e., $x(t_{\text{ret}}) = A \cos \omega t_{\text{ret}}$. To calculate the fields at the position $\mathbf{R} = (X, Y, Z)$ at time t , we need to solve the following transcendental equation for t_{ret} :

$$t_{\text{ret}} = t - \frac{r_{\text{ret}}}{c} = t - \frac{1}{c} \sqrt{(X - A \cos^2 \omega t_{\text{ret}})^2 + Y^2 + Z^2}. \quad (10.35)$$

The solution of (9.37) can be expressed as a root finding problem for which we need to find the zero of the function $f(t_{\text{ret}})$:

$$f(t_{\text{ret}}) = t - t_{\text{ret}} - \frac{r_{\text{ret}}}{c}. \quad (10.36)$$

One way to find the root is to use a bracketing method. The idea is to first find a value t_a such that $f(t_a) > 0$, and another value t_b such that $f(t_b) < 0$. Because $f(t_{\text{ret}})$ is continuous, there is a value of t_{ret} in the interval $t_a < t_{\text{ret}} < t_b$ such that $f(t_{\text{ret}}) = 0$. There are various ways of guessing the solution and reducing the size of the interval. In the *bisection method* (see Section 6.6), we use the midpoint $t_m = (t_a + t_b)/2$ as the guess, and compute $f(t_m)$. If $f(t_m) > 0$, find the next midpoint with $t_a = t_m$; otherwise, find the next midpoint with $t_b = t_m$. Continue until $f(t_m) = 0$ to the desired accuracy.

In the *method of false position*, we find the intersection of the line connecting the points $(t_a, f(t_a))$ and $(t_b, f(t_b))$ with the t_{ret} axis. The value of t_{ret} at the intersection, t_1 , gives the first estimate for the zero of (9.37). If $f(t_1) > 0$, repeat the calculation with $t_a = t_1$, else repeat the calculation with $t_b = t_1$. This procedure is continued until the desired level of accuracy is achieved.

The bisection method and the method of false position are guaranteed to find a root. *Newton's method*, which uses the slope $df(t_{\text{ret}})/dt_{\text{ret}}$ to estimate the solution, is much faster, but does not always give the correct answer and must be used with care. In this method the guess for t_{ret} at the n th iteration is given by

$$t_n = t_{n-1} - \frac{f(t_{n-1})}{df(t_{\text{ret}})/dt_{\text{ret}}|_{t_{\text{ret}}=t_{n-1}}}. \quad (10.37)$$

We use Newton's method unless the interval between two guesses is increasing. In that case we use the bisection method to decrease the interval. **Program radiation** uses this strategy to calculate the electric field due to an oscillating charge. We choose units such that the velocity is measured in terms of the speed of light c .

```
PROGRAM radiation
! compute electric fields due to accelerating charge and
! plot electric field lines
```

```

LIBRARY "csgraphics"
DIM snapshot$(100)
CALL initial(L,nsnap, radius, lpc, dt, dtheta)
FOR isnap = 1 to nsnap
    CALL draw_field(L, isnap, radius, dt, dtheta, snapshot$())
NEXT isnap
CALL animate(L, nsnap, snapshot$())
END

SUB initial(L,nsnap,a,lpc,dt,dtheta)
    LET L = 20
    CALL compute_aspect_ratio(2*L,xwin,ywin)
    SET WINDOW -xwin,xwin,-ywin,ywin
    LET a = 0.3           ! "radius" of visual image of charge
    LET lpc = 10          ! number of field lines
    LET dt = 0.4          ! time between plots
    LET dtheta = 2*pi/lpc
    LET nsnap = 15         ! number of snapshots
END SUB

SUB draw_field(L,isnap,a,dt,dtheta,snapshot$())
    ! adopted from Program fieldline
    DIM E(3),R(3),r_ret(3),v_ret(3),a_ret(3)
    DECLARE DEF dotproduct
    LET ds = 1
    LET t = isnap*dt           ! observation time
    LET R(1) = 0
    LET R(2) = 0
    CALL motion(R(),t,r_ret(),v_ret(),a_ret())
    ! find charge position at time t
    LET x = R(1) - r_ret(1)
    LET y = R(2) - r_ret(2)
    BOX CIRCLE x-a,x+a,y-a,y+a
    FOR theta = 0 to 2*pi step dtheta
        LET xline = x + a*cos(theta)
        LET yline = y + a*sin(theta)
        PLOT xline,yline;
        DO
            LET R(1) = xline
            LET R(2) = yline
            CALL field(R(),t,E()) ! compute fields
            LET Emag = sqr(dotproduct(E(),E()))
            ! new position on field line
            LET xline = xline + ds*E(1)/Emag
            LET yline = yline + ds*E(2)/Emag
            PLOT xline,yline;
    
```



```

! insure that f(ta) > 0 and f(tb) < 0
DO
    LET ta = ta*1.6
    CALL fanddf(fa,dfdt,R(),t,ta)
LOOP until fa > 0
LET t_ret = 0.5*(ta + tb)
CALL fanddf(f,dfdt,R(),t,t_ret)
CALL zero_of_f(f,dfdt,R(),t,t_ret,ta,tb)
END SUB

SUB fanddf(f,dfdt,R(),t,t_ret)
    ! calculate f and df/dt_r
    DIM r_ret(3),v_ret(3),a_ret(3)
    DECLARE DEF dotproduct
    CALL motion(R(),t_ret,r_ret(),v_ret(),a_ret())
    LET dist_ret = sqr(dotproduct(r_ret(),r_ret()))
    LET f = t - t_ret - dist_ret
    ! derivative evaluated at retarded time
    LET dfdt = -1 + dotproduct(r_ret(),v_ret())/dist_ret
END SUB

SUB zero_of_f(f,dfdt,R(),t,t_ret,ta,tb)
    ! do no more than 100 iterations to find the value of the reduced
    ! time t_ret such that f = t - t_ret - dist_ret = 0
    LET eps = 1e-6
    LET dt_r = tb - ta
    FOR j = 1 to 100
        ! Newton difference between successive t_ret estimates
        LET dt = f/dfdt
        LET sign = ((t_ret - ta) - dt)*((t_ret - tb) - dt)
        LET dt_old = dt_r
        IF sign >= 0 or abs(2*dt) > abs(dt_old) then
            ! use bisection method if next Newton iteration is not
            ! between ta and tb, or if dt is not less than half
            ! of the old value of dt_r
            LET dt_r = 0.5*(ta - tb)
            LET t_ret = tb + dt_r
        ELSE
            ! use Newton's method
            LET dt_r = dt
            LET t_ret = t_ret - dt_r
        END IF
        IF abs(dt_r) < eps then EXIT SUB      ! convergence test
        CALL fanddf(f,dfdt,R(),t,t_ret)
        IF f < 0 then
            LET tb = t_ret
        ELSE

```

```

        LET ta = t_ret
    END IF
NEXT j
PRINT "too many iterations"
STOP
END SUB

SUB animate(L,nsnap,snapshot$())
    FOR isnap = 1 to nsnap      ! show motion picture
        BOX SHOW snapshot$(isnap) at -L,-L
        PAUSE 1
    NEXT isnap
END SUB

SUB crossproduct(a(),b(),c())
    LET c(1) = a(2)*b(3) - a(3)*b(2)
    LET c(2) = a(3)*b(1) - a(1)*b(3)
    LET c(3) = a(1)*b(2) - a(2)*b(1)
END SUB

DEF dotproduct(a(),b())
    LET dotproduct = a(1)*b(1) + a(2)*b(2) + a(3)*b(3)
END DEF

```

Problem 10.14. Field lines from an accelerating charge

1. Read **Program radiation** carefully to understand the correspondence between the program and the calculation discussed in the text of the electric field lines due to an accelerating point charge. Describe qualitatively the nature of the electric field lines from an oscillating point charge.
2. Use **Program radiation** to calculate **E** due to a positively charged particle oscillating about the origin according to $x(t') = 0.2 \cos t'$. The program draws field lines in the x - y plane starting from a small circle surrounding the origin. Let the observation time be $t = 1$ and stop drawing each field line when $|x| > 20$ or $|y| > 20$. How do the field lines differ from those of a static charge at the origin?
3. What happens to the field lines as you increase the observation time t ?
4. One way of visualizing how the field lines change with time is by drawing the field lines at successive times, capturing the screen image each time. In True BASIC we can use the **BOX KEEP** statement and then show the images sequentially using the **BOX SHOW** statement. What new information does this mode of presentation convey?
5. Repeat the above observations for a charge moving with uniform circular motion about the origin.

Problem 10.15. Spatial dependence of radiating fields

- As waves propagate from an accelerating point source, the total power that passes through a spherical surface of radius R remains constant. Because the surface area is proportional to R^2 , the power per unit area or intensity is proportional to $1/R^2$. Also, because the intensity is proportional to E^2 , we expect that $E \propto 1/R$ far from the source. Modify *Program radiation* to verify this result for a charge that is oscillating along the x -axis according to $x(t') = 0.2 \cos t'$. Plot $|E|$ as a function of the observation time t for a fixed position such as $\mathbf{R} = (10, 10, 0)$. The field should oscillate in time. Find the amplitude of this oscillation. Next double the distance of the observation point from the origin. How does the amplitude depend on R ?
- Repeat part (a) for several directions and distances. Generate a polar diagram showing the amplitude as a function of angle in the x - y plane. Is the radiation greatest along the line in which the charge oscillates?

Problem 10.16. Field lines from a charge moving at constant velocity

- Use *Program radiation* to calculate \mathbf{E} due to a charged particle moving at constant velocity toward the origin, i.e., $x(t_{\text{ret}}) = 1 - 2t_{\text{ret}}$. Take a snapshot at $t = 0.5$ and compare the field lines with those you expect from a stationary charge.
- Modify *SUB motion* so that $x(t_{\text{ret}}) = 1 - 2t_{\text{ret}}$ for $t_{\text{ret}} < 0.5$ and $x(t_{\text{ret}}) = 0$ for $t_{\text{ret}} > 0.5$. Describe the field lines for $t > 0.5$. Does the particle accelerate at any time? Is there any radiation?

Problem 10.17. Frequency dependence of an oscillating charge

- The radiated power at any point in space is proportional to E^2 . Plot $|E|$ versus time at a fixed observation point (e.g., $X = 10, Y = Z = 0$), and calculate the frequency dependence of the amplitude of $|E|$ due to a charge oscillating at the frequency ω . It is shown in standard textbooks that the power associated with radiation from an oscillating dipole is proportional to ω^4 . How does the ω -dependence that you measured compare to that for dipole radiation? Repeat for a much bigger value of R , and explain any differences.
- Repeat part (a) for a charge moving in a circle. Are there any qualitative differences?

10.5 *Maxwell's Equations

In Section 9.4 we found that accelerating charges produce electric and magnetic fields which depend on position and time. We now investigate the direct relation between changes in \mathbf{E} and \mathbf{B} given by the differential form of Maxwell's equations:

$$\frac{\partial \mathbf{B}}{\partial t} = -\frac{1}{c} \nabla \times \mathbf{E} \quad (10.38)$$

$$\frac{\partial \mathbf{E}}{\partial t} = c \nabla \times \mathbf{B} - 4\pi \mathbf{j}, \quad (10.39)$$

where \mathbf{j} is the electric current density. We can regard (9.40) and (9.41) as the basis of electrodynamics. In addition to (9.40) and (9.41), we need the relation between \mathbf{j} and the charge density ρ that expresses the conservation of charge:

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot \mathbf{j}. \quad (10.40)$$

A complete description of electrodynamics requires (9.40), (9.41), and (??) and the initial values of all currents and fields.

For completeness, we obtain the Maxwell's equations that involve $\nabla \cdot \mathbf{B}$ and $\nabla \cdot \mathbf{E}$ by taking the divergence of (9.40) and (9.41), substituting (??) for $\nabla \cdot \mathbf{j}$, and then integrating over time. If the initial fields are zero, we obtain (using the relation $\nabla \cdot (\nabla \times \mathbf{a}) = 0$ for any vector \mathbf{a}):

$$\nabla \cdot \mathbf{E} = 4\pi\rho \quad (10.41)$$

$$\nabla \cdot \mathbf{B} = 0. \quad (10.42)$$

If we introduce the electric and magnetic potentials, it is possible to convert the first-order equations (9.40) and (9.41) to second-order differential equations. However, the familiar first-order equations are better suited for numerical analysis. To solve (9.40) and (9.41) numerically, we need to interpret the curl and divergence of a vector. As its name implies, the curl of a vector measures how much the vector twists around a point. A coordinate free definition of the curl of an arbitrary vector \mathbf{W} is

$$(\nabla \times \mathbf{W}) \cdot \hat{\mathbf{S}} = \lim_{S \rightarrow 0} \frac{1}{S} \oint_C \mathbf{W} \cdot d\mathbf{l}, \quad (10.43)$$

where \mathbf{S} is the area of any surface bordered by the closed curve C , and $\hat{\mathbf{S}}$ is a unit vector normal to the surface S .

Equation (??) gives the component of $\nabla \times \mathbf{W}$ in the direction of $\hat{\mathbf{S}}$ and suggests a way of computing the curl numerically. We divide space into cubes of linear dimension Δl . The rectangular components of \mathbf{W} can be defined either on the edges or on the faces of the cubes. We compute the curl using both definitions. We first consider a vector \mathbf{B} that is defined on the edges of the cubes so that the curl of \mathbf{B} is defined on the faces. (We use the notation \mathbf{B} because we will find that it is convenient to define the magnetic field in this way.) Associated with each cube is one edge vector and one face vector. We label the cube by the coordinates corresponding to its lower left front corner (see Figure ??a). The three components of \mathbf{B} associated with this cube are shown in Figure ??a. The other edges of the cube are associated with B vectors defined at neighboring cubes.

The discrete version of (??) for the component of $\nabla \times \mathbf{B}$ defined on the front face of the cube (i, j, k) is

$$(\nabla \times \mathbf{B}) \cdot \hat{\mathbf{S}} = \frac{1}{(\Delta l)^2} \sum_{i=1}^4 B_i \Delta l_i, \quad (10.44)$$

where $S = (\Delta l)^2$, and B_i and l_i are shown in Figures ??b and ??c, respectively. Note that two of the components of \mathbf{B} are associated with neighboring cubes.

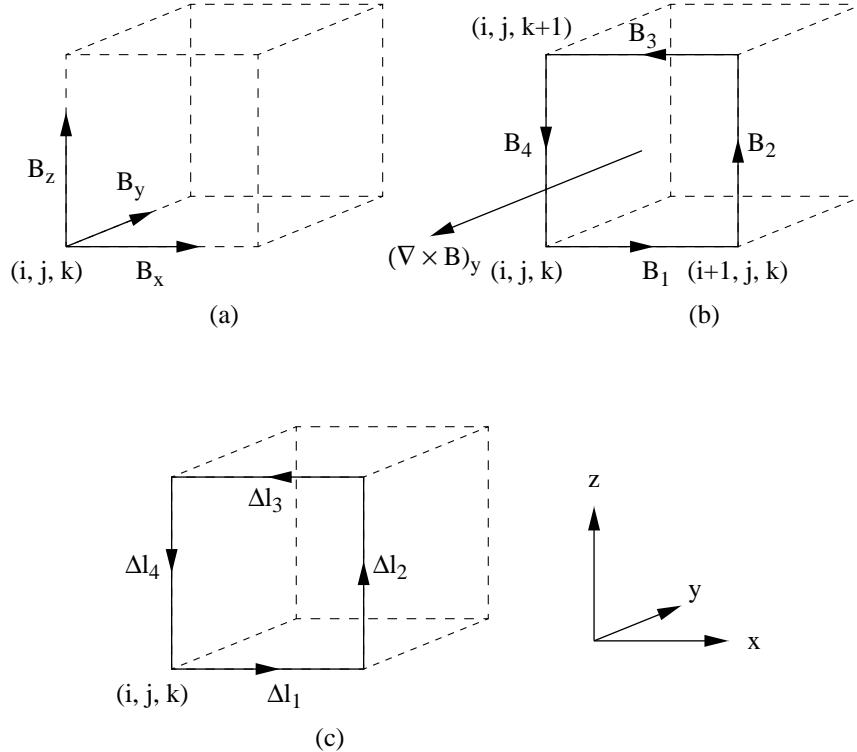


Figure 10.5: Calculation of the curl of \mathbf{B} defined on the edges of a cube. (a) The edge vector \mathbf{B} associated with cube (i, j, k) . (b) The components B_i along the edges of the front face of the cube. $B_1 = B_x(i, j, k)$, $B_2 = B_z(i+1, j, k)$, $B_3 = -B_x(i, j, k+1)$, and $B_4 = -B_z(i, j, k)$. (c) The vector components Δl_i on the edges of the front face. (The y -component of $\nabla \times \mathbf{B}$ defined on the face points in the negative y direction.)

The components of a vector also can be defined on the faces of the cubes. We call this vector \mathbf{E} because it will be convenient to define the electric field in this way. In Figure ??a we show the components of \mathbf{E} associated with the cube (i, j, k) . Because \mathbf{E} is normal to a cube face, the components of $\nabla \times \mathbf{E}$ lie on the edges. The components E_i and l_i are shown in Figures ??b and ??c respectively. The form of the discrete version of $\nabla \times \mathbf{E}$ is similar to (??) with B_i replaced by E_i , where E_i and l_i are shown in Figures ??b and ??c respectively. The z -component of $\nabla \times \mathbf{E}$ is along the left edge of the front face.

A coordinate free definition of the divergence of the vector field \mathbf{W} is

$$\nabla \cdot \mathbf{W} = \lim_{V \rightarrow 0} \frac{1}{V} \oint_S \mathbf{W} \cdot d\mathbf{S}, \quad (10.45)$$

where V is the volume enclosed by the closed surface \mathbf{S} . The divergence measures the average flow of the vector through a closed surface. An example of the discrete version of (??) is given in (??).

We now discuss where to define the quantities $\rho, \mathbf{j}, \mathbf{E}$, and \mathbf{B} on the grid. It is natural to define the charge density ρ at the center of a cube. From the continuity equation (??), we see that this definition leads us to define \mathbf{j} at the faces of the cube. Hence, each face of a cube has a number associated with it corresponding to the current density flowing parallel to the outward normal to that face. Given the definition of \mathbf{j} on the grid, we see from (9.41) that the electric field \mathbf{E} and \mathbf{j} should be defined at the same places, and hence we define the electric field on the faces of the cubes. Because \mathbf{E} is defined on the faces, it is natural to define the magnetic field \mathbf{B} on the edges of the cubes. Our definitions of the vectors \mathbf{j}, \mathbf{E} , and \mathbf{B} on the grid are now complete.

We label the faces of cube c by the symbol f_c . If we use the simplest finite difference method with a discrete time step Δt and discrete spatial interval $\Delta x = \Delta y = \Delta z \equiv \Delta l$, we can write the continuity equation as:

$$[\rho(c, t + \frac{1}{2}\Delta t) - \rho(c, t - \frac{1}{2}\Delta t)] = -\frac{\Delta t}{\Delta l} \sum_{f_c=1}^6 j(f_c, t). \quad (10.46)$$

The factor of $1/\Delta l$ comes from the area of a face $(\Delta l)^2$ used in the surface integral in (??) divided by the volume $(\Delta l)^3$ of a cube. In the same spirit, the discretization of (9.41) can be written as:

$$E(f, t + \frac{1}{2}\Delta t) - E(f, t - \frac{1}{2}\Delta t) = \Delta t [\nabla \times \mathbf{B} - 4\pi j(f, t)]. \quad (10.47)$$

Note that \mathbf{E} in (??) and ρ in (??) are defined at different times than \mathbf{j} . As usual, we choose units such that $c = 1$.

We next need to define a square around which we can discretize the curl. If \mathbf{E} is defined on the faces, it is natural to use the square that is the border of the faces. As we have discussed, this choice implies that we should define the magnetic field on the edges of the cubes. We write (??) as:

$$E(f, t + \frac{1}{2}\Delta t) - E(f, t - \frac{1}{2}\Delta t) = \Delta t \left[\frac{1}{\Delta l} \sum_{e_f=1}^4 B(e_f, t) - 4\pi j(f, t) \right], \quad (10.48)$$

where the sum is over e_f , the four edges of the face f (see Figure ??b). Note that B is defined at the same time as j . In a similar way we can write the discrete form of (9.40) as:

$$B(e, t + \Delta t) - B(e, t) = -\frac{\Delta t}{\Delta l} \sum_{f_e=1}^4 E(f_e, t + \frac{1}{2}\Delta t), \quad (10.49)$$

where the sum is over f_e , the four faces that share the same edge e (see Figure ??b).

We now have a well defined algorithm for computing the spatial dependence of the electric and magnetic field, the charge density, and the current density as a function of time. This algorithm was developed by Yee, an electrical engineer, in 1966, and independently by Visscher, a physicist, in 1988 who also showed that all of the integral relations and other theorems that are satisfied by the continuum fields are also satisfied for the discrete fields.

Perhaps the most difficult part of the method is specifying the initial conditions since we cannot simply place a charge somewhere. The reason is that the initial fields appropriate for this

charge would not be present. Indeed, our rules for updating the fields and the charge densities reflect the fact that the electric and magnetic fields do not appear instantaneously at all positions in space when a charge appears, but instead evolve from the initial appearance of a charge. Of course, charges do not appear out of nowhere, but appear by disassociating the charges from neutral objects. Conceptually, the simplest initial condition corresponds to two charges of opposite sign moving oppositely to each other. This condition corresponds to an initial current on one face. From this current, a charge density and electric field appears using (??) and (??), respectively, and a magnetic field appears using (??).

Because we cannot compute the fields for an infinite lattice, we need to specify the boundary conditions. The easiest method is to use fixed boundary conditions such that the fields vanish at the edges of the lattice. If the lattice is sufficiently large, fixed boundary conditions are a reasonable approximation. However, fixed boundary conditions usually lead to nonphysical reflections off the edges, and a variety of approaches have been used including boundary conditions equivalent to a conducting medium that gradually absorbs the fields. In some cases physically motivated boundary conditions can be employed. For example, in simulations of microwave cavity resonators (see Problem ??), the appropriate boundary conditions are that the tangential component of **E** and the normal component of **B** vanish at the boundary.

As we have noted, **E** and ρ are defined at different times than **B** and **j**. This “half-step” approach leads to well behaved equations that are stable over a range of parameters. An analysis of the stability requirement for the Yee-Visscher algorithm shows that the time step Δt must be smaller than the spatial grid Δl by:

$$c\Delta t \leq \frac{\Delta l}{\sqrt{3}}. \quad (\text{stability requirement}) \quad (10.50)$$

Your understanding of the Yee-Visscher algorithm for finding solutions to Maxwell’s equations will be enhanced by carefully reading the program listing for Program **maxwell** given in the following. The program uses a special True BASIC graphics subroutine **PICTURE arrow** that is called by a **DRAW** statement. Such a subroutine has arguments just like any other subroutine. Its utility is that it creates a graphical image that can then be rotated or shifted in space.

```

PROGRAM maxwell
! implementation of Yee-Visscher algorithm
LIBRARY "csgraphics"
PUBLIC E(0 to 21,0 to 21,0 to 21,3),B(0 to 21,0 to 21,0 to 21,3)
PUBLIC j(0 to 21,0 to 21,0 to 21,3)
PUBLIC n(3),dt,d1
PUBLIC mid,fpi,d1_1
CALL initial(escale,bscale,jscale)
CALL screen(#1,#2,#3)
DO
  CALL current(t)
  CALL newE
  CALL newB
  CALL plotfields(escale,bscale,jscale,#1,#2,#3)
DO

```

```

LOOP until key input
GET KEY k
LOOP until k = ord("s")
END

SUB initial(escale,bscale,jscale)
  DECLARE PUBLIC E(,,,),B(,,,)
  DECLARE PUBLIC n(),dt
  DECLARE PUBLIC mid,fpi,dl_1
  LET dt = 0.03
  LET n(1) = 8
  LET n(2) = 8
  LET n(3) = 8
  LET mid = n(1)/2
  LET dl = 0.1
  LET dl_1 = 1/dl
  LET fpi = 4*pi
  LET escale = dl/(4*pi*dt)
  LET bscale = escale*dl/dt
  LET jscale = 1
  FOR x = 1 to n(1)
    FOR y = 1 to n(2)
      FOR z = 1 to n(3)
        FOR comp = 1 to 3
          LET E(x,y,z,comp) = 0
          LET B(x,y,z,comp) = 0
        NEXT comp
      NEXT z
    NEXT y
  NEXT x
END SUB

SUB current(t)
  DECLARE PUBLIC j(,,,),n(),mid
  ! steady current loop in x-y plane turned on at t = 0 and left on
  LET j(mid,mid,mid,2) = 1
  LET j(mid,mid,mid,1) = -1
  LET j(mid-1,mid,mid,2) = -1
  LET j(mid,mid-1,mid,1) = 1
END SUB

SUB newE
  ! E defined at the faces
  DECLARE PUBLIC E(,,,),B(,,,),j(,,,)
  DECLARE PUBLIC n(),dt,dl_1,fpi
  FOR x = 1 to n(1)

```

```

FOR y = 1 to n(2)
  FOR z = 1 to n(3)
    LET curlBx = B(x,y,z,2)+B(x,y+1,z,3)-B(x,y,z+1,2)-B(x,y,z,3)
    LET curlBx = curlBx*dl_1
    LET E(x,y,z,1) = E(x,y,z,1) + dt*(curlBx - fpi*j(x,y,z,1))
    LET curlBy = B(x,y,z,3)-B(x,y,z,1)+B(x,y,z+1,1)-B(x+1,y,z,3)
    LET curlBy = curlBy*dl_1
    LET E(x,y,z,2) = E(x,y,z,2) + dt*(curlBy - fpi*j(x,y,z,2))
    LET curlBz = B(x,y,z,1)+B(x+1,y,z,2)-B(x,y+1,z,1)-B(x,y,z,2)
    LET curlBz = curlBz*dl_1
    LET E(x,y,z,3) = E(x,y,z,3) + dt*(curlBz - fpi*j(x,y,z,3))
    NEXT z
  NEXT y
NEXT x
END SUB

SUB newB
  ! B defined at the edges
  DECLARE PUBLIC E(,,,),B(,,,)
  DECLARE PUBLIC n(),dt,dl_1
  FOR x = 1 to n(1)
    FOR y = 1 to n(2)
      FOR z = 1 to n(3)
        LET curlEx = E(x,y,z,3)-E(x,y,z,2)-E(x,y-1,z,3)+E(x,y,z-1,2)
        LET curlEx = curlEx*dl_1
        LET B(x,y,z,1) = B(x,y,z,1) - curlEx*dt
        LET curlEy = E(x,y,z,1)-E(x,y,z,3)-E(x,y,z-1,1)+E(x-1,y,z,3)
        LET curlEy = curlEy*dl_1
        LET B(x,y,z,2) = B(x,y,z,2) - curlEy*dt
        LET curlEz = E(x,y,z,2)-E(x,y,z,1)-E(x-1,y,z,2)+E(x,y-1,z,1)
        LET curlEz = curlEz*dl_1
        LET B(x,y,z,3) = B(x,y,z,3) - curlEz*dt
      NEXT z
    NEXT y
  NEXT x
END SUB

SUB screen(#1,#2,#3)
  DECLARE PUBLIC n()
  LET L = n(1)
  CALL compute_aspect_ratio(L+1,xwin,ywin)
  SET BACKGROUND COLOR "black"
  OPEN #1: screen 0,.5,0,.5
  SET WINDOW 0,xwin,0,ywin
  SET COLOR "white"
  OPEN #2: screen 0.5,1,0.5,1

```

```

SET WINDOW 0,xwin,0,ywin
SET COLOR "white"
OPEN #3: screen 0.5,1,0,0.5
SET WINDOW 0,xwin,0,ywin
SET COLOR "white"
OPEN #4: screen 0,0.5,0.5,1
SET COLOR "white"
SET CURSOR 1,1
PRINT "Type s to stop"
PRINT
PRINT "Type any other key for next time step"
END SUB

SUB plotfields(escale,bscale,jscale,#1,#2,#3)
DECLARE PUBLIC E(,,,),B(,,,),j(,,,)
DECLARE PUBLIC n(),dt,mid
WINDOW #1
CLEAR
PRINT "E(x,y)"
FOR x = 1 to n(1)
  FOR y = 1 to n(2)
    CALL plotarrow(E(x,y,mid,1),x,y,escale,0,0.5,0,pi)
    CALL plotarrow(E(x,y,mid,2),x,y,escale,0.5,0,pi/2,3*pi/2)
  NEXT y
NEXT x
WINDOW #2
CLEAR
PRINT "B(x,z)"
FOR x = 1 to n(1)
  FOR z = 1 to n(3)
    CALL plotarrow(B(x,mid,z,1),x,z,bscale,0.5,0,0,pi)
    CALL plotarrow(B(x,mid,z,3),x,z,bscale,0,0.5,pi/2,3*pi/2)
  NEXT z
NEXT x
WINDOW #3
CLEAR
PRINT "j(x,y)"
FOR x = 1 to n(1)
  FOR y = 1 to n(2)
    CALL plotarrow(j(x,y,mid,1),x,y,jscale,0,0.5,0,pi)
    CALL plotarrow(j(x,y,mid,2),x,y,jscale,0.5,0,pi/2,3*pi/2)
  NEXT y
NEXT x
END SUB

SUB plotarrow(V,x,y,scale,shiftx,shifty,angle1,angle2)

```

```

IF V > 0 then
    DRAW arrow(V/scale) with rotate(angle1)*shift(x+shiftx,y+shifty)
ELSE IF V < 0 then
    DRAW arrow(-V/scale) with rotate(angle2)*shift(x+shiftx,y+shifty)
END IF
END SUB

PICTURE arrow(x)
SET COLOR "yellow"
PLOT LINES: -0.25*x,0;0.25*x,0;0.12*x,0.12*x
PLOT LINES: 0.25*x,0;0.12*x,-0.12*x
SET COLOR "white"
END PICTURE

```

Problem 10.18. Fields from a current loop

1. Program `maxwell` shows the electric field in the x - y plane and the magnetic field in the x - z plane in separate windows. The fields are represented by arrows, whose length is proportional to the field magnitude at each position where the field is defined. A steady current loop in the middle of the x - y plane is turned on at $t = 0$ and left on for all time (see `SUB current`). Before running the program, predict what you expect to see. Compare your expectations with the results of the simulation. Use $\Delta t = 0.03$, $\Delta l = 0.1$, and take the number of cubes in each direction to be $n(1) = n(2) = n(3) = 8$.
2. Verify the stability requirement (??), by running your program with $\Delta t = 0.1$ and $\Delta l = 0.1$. Then try $\Delta t = 0.05$ and $\Delta l = \Delta t\sqrt{3}$. What happens to the results in part (a) if the stability requirement is not satisfied?
3. Modify the current density in part (a) so that \mathbf{j} is nonzero only for one time step. What happens to the electric and magnetic field vectors?
4. The amplitude of the fields far from the current loop should be characteristic of radiation fields for which the amplitude falls off as $1/r$, where r is the distance from the current loop to the observation point. Try to detect this dependence (if you have sufficient patience or computer resources).

Problem 10.19. Microwave cavity resonators

1. Cavity resonators are a practical way of storing energy in the form of oscillating electric and magnetic fields without losing as much energy as would be dissipated in a resonant LC circuit. Consider a cubical resonator of linear dimension L whose walls are made of a perfectly conducting material. The tangential components of \mathbf{E} and the normal component of \mathbf{B} vanish at the walls. Standing microwaves can be set up in the box of the form (cf. Reitz et al.)

$$E_x = E_{x0} \cos k_x x \sin k_y y \sin k_z z e^{i\omega t} \quad (10.51a)$$

$$E_y = E_{y0} \cos k_y y \sin k_x x \sin k_z z e^{i\omega t} \quad (10.51b)$$

$$E_z = E_{z0} \cos k_z z \sin k_x x \sin k_y y e^{i\omega t}. \quad (10.51c)$$

The wave vector $\mathbf{k} = (k_x, k_y, k_z) = (m_x\pi/L, m_y\pi/L, m_z\pi/L)$, where m_x , m_y , and m_z are integers. A particular mode is labeled by the integers (m_x, m_y, m_z) . The initial electric field is perpendicular to \mathbf{k} , and $\omega = ck$. Implement the boundary conditions at $(x = 0, y = 0, z = 0)$ and $(x = L, y = L, z = L)$. Set $\Delta t = 0.05$, $\Delta l = 0.1$, and $L = 1$. At $t = 0$, set $\mathbf{B} = 0$, $\mathbf{j} = 0$ (there are no currents within the cavity), and use (??) with $(m_x, m_y, m_z) = (0, 1, 1)$, and $E_{x0} = 1$. Plot the field components at specific positions as a function of t and find the resonant frequency ω . Compare your computed value of ω with the analytical result. Do the magnetic fields change with time? Are they perpendicular to \mathbf{k} and \mathbf{E} ?

2. Repeat part (a) for two other modes.
3. Repeat part (a) with a uniform random noise added to the initial field at all positions. Assume the amplitude of the noise is δ and describe the resulting fields for $\delta = 0.1$. Are they similar to those without noise? What happens for $\delta = 0.5$? More quantitative results can be found by computing the power spectrum $|E(\omega)|^2$ for the electric field at a few positions. What is the order of magnitude of δ for which the maximum of $|E(\omega)|^2$ at the standing wave frequency is swamped by the noise?
4. Change the shape of the container slightly by removing a 0.1×0.1 cubical box from each of the corners of the original resonator. Do the standing wave frequencies change? Determine the standing wave frequency by adding noise to the initial fields and looking at the power spectrum. How do the standing wave patterns change?
5. Change the shape of the container slightly by adding a 0.1×0.1 cubical box at the center of one of the faces of the original resonator. Do the standing wave frequencies change? How do the standing wave patterns change?
6. Cut a 0.2×0.2 square hole in a face in the y - z plane, and double the computational region in the x direction. Begin with a $(0, 1, 1)$ standing wave, and observe how the fields “leak” out of the hole.

Problem 10.20. Billiard microwave cavity resonators

1. Repeat part (a) of Problem ?? for $L_x = L_y = 2$, $L_z = 0.2$, $\Delta l = 0.1$, and $\Delta t = 0.05$. Indicate the magnitude of the electric field in the $L_z = 0.1$ plane by a color code. Choose an initial normal mode field distribution and describe the pattern that you obtain. Then repeat your calculation for a random initial field distribution.
2. Place an approximately circular conductor in the middle of the cavity of radius $r = 0.4$. Describe the patterns that you see. Such a geometry leads to chaotic trajectories for particles moving within such a cavity (see Project 6.24). Is there any evidence of chaotic behavior in the field pattern?
3. Repeat part (b) with the circular conductor placed off center.

10.6 Project

Much of the difficulty in understanding electromagnetic phenomena is visualizing its three-dimensional character. Although True BASIC has an excellent three-dimensional graphics toolkit, we have not

used it here because of the difficulty of translating its statements to other programming languages. Many interesting problems can be posed based on the simple, but nontrivial question of how the electromagnetic fields can best be represented visually in various contexts.

Many of the techniques used in this chapter, e.g., the random walk method and the relaxation method for solving Laplace's equation, have applications in other fields, especially problems in fluid flow and transport. Similarly, the multigrid method, discussed below, has far reaching applications.

Project 10.21. Multigrid method

In general, the relaxation method for solving Laplace's equation is very slow even using overrelaxation. The reason is that the local updates of the relaxation method cannot quickly take into account effects at very large length scales. The *multigrid method* greatly improves performance by using relaxation at many length scales. The important idea is to use a relaxation method to find the values of the potential on coarser and coarser grids, and then use the coarse grid values to determine the fine grid values. The fine grid relaxation updates take into account effects at short length scales. If we define the initial grid by a lattice spacing $b = 1$, then the coarser grids are characterized by $b = 2^n$, where n is the grid level. We need to decide how to use the fine grid values of the potential to assign values to a coarser grid, and then how to use a coarse grid to assign values to a finer grid. The first step is called prolongation and the second step is called restriction. There is some flexibility on how to do these two operations. We discuss one approach.

We define the points of the coarse grid as every other point of the fine grid. That is, if the set $\{i, j\}$ represents the positions of the points of the fine grid, then $\{2i, 2j\}$ represents the positions of the coarse grid points. The fine grid points that are at the same position as a coarse grid point are assigned the value of the potential of the corresponding coarse grid point. The fine grid points that have two coarse grid points as nearest neighbors are assigned the average value of these two coarse grid points. The other fine grid points have four coarse grid points as next nearest neighbors and are assigned the average value of these four coarse grid points. This prescription defines the restriction of the coarse grid to the fine grid.

In the full weighting prolongation method, each coarse grid point receives one fourth of the potential of the fine grid point at the same position, one eighth of the potential for the four nearest neighbor points of the fine grid, and one sixteenth of the potential for the four next nearest neighbor points of the fine grid. Note that the sum of these fractions, $1/4 + 4(1/8) + 4(1/16)$, adds up to unity. An alternative procedure, known as half weighting, ignores the next nearest neighbors and uses one half of the potential of the fine grid point at the same position as the coarse grid point.

1. Write a program that implements the multigrid method using Gauss-Seidel relaxation on a checkerboard lattice (see Problem 9.9b). In its simplest form the program should allow the user to intervene and decide whether to go to a finer or coarser grid, or to remain at the same level for the next relaxation step. Also have the program print the potential at each point of the current level after each relaxation step. Test your program on a 4×4 grid whose boundary points are all equal to unity, and whose initial internal points are set to zero. Make sure that the boundary points of the coarse grids also are set to unity.
2. The exact solution for part (a) gives a potential of unity at each point. How many relaxation steps does it take to reach unity within 0.1% at every point by simply using the 4×4 grid? How many steps does it take if you use one coarse grid and continue until the coarse grid values are within 0.1% of unity? Is it necessary to carry out any fine grid relaxation steps to

reach the desired accuracy on the fine grid? Next start with the coarsest scale, which is just one point. How many relaxation steps does it take now?

3. Repeat part (b), but change the boundary so that one side of the boundary is held at a potential of 0.5. Experiment with different sequences of prolongation, restriction, and relaxation.
4. Assume that the boundary points alternate between zero and unity, and repeat part (b). Does the multigrid method work? Should one go up and down in levels many times instead of staying at the coarsest level and then going down to the finest level?

References and Suggestions for Further Reading

- Forman S. Acton, *Numerical Methods That Work*, Harper & Row (1970); corrected edition, Mathematical Association of America (1990). Chapter 18 discusses solutions to Laplace's equation using the relaxation method and alternative approaches.
- Charles K. Birdsall and A. Bruce Langdon, *Plasma Physics via Computer Simulation*, McGraw-Hill (1985).
- D. H. Choi and W. J. R. Hoefer, "The finite-difference-time-domain method and its application to eigenvalue problems," *IEEE Trans. Microwave Theory and Techniques* **34**, 1464 (1986). The authors use Yee's algorithm to model microwave cavity resonators.
- David M. Cook, *The Theory of the Electromagnetic Field*, Prentice Hall (1975). One of the first books to introduce numerical methods in the context of electromagnetism.
- Robert Ehrlich, Jaroslaw Tuszyński, Lyle Roelofs, and Ronald Stoner, *Electricity and Magnetism Simulations: The Consortium for Upper-Level Physics Software*, John Wiley (1995).
- Richard P. Feynman, Robert B. Leighton, and Matthew Sands, *The Feynman Lectures on Physics*, Vol. 2, Addison-Wesley (1963).
- T. E. Freeman, "One-, two- or three-dimensional fields?," *Am. J. Phys.* **63**, 273 (1995).
- Robert M. Eisberg and Lawrence S. Lerner, *Physics*, Vol. 2, McGraw-Hill (1981). An introductory level text that uses numerical methods to find the electric field lines and solutions to Laplace's equation.
- R. L. Gibbs, Charles W. Beason, and James D. Beason, "Solutions to boundary value problems of the potential type by random walk method," *Am. J. Phys.* **43**, 782 (1975).
- R. H. Good, "Dipole radiation: Simulation using a microcomputer," *Am. J. Phys.* **52**, 1150 (1984). The author reports on a graphical simulation of dipole radiation.
- R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*, McGraw-Hill (1981).
- Steven E. Koonin and Dawn C. Meredith, *Computational Physics*, Addison-Wesley (1990). See Chapter 6 for a discussion of the numerical solution of elliptic partial differential equations of which Laplace's and Poisson's equations are examples.

William M. MacDonald, “Discretization and truncation errors in a numerical solution of Laplace’s equation,” *Amer. J. Phys.* **62**, 169 (1994).

William H. Press and Saul A. Teukolsky, “Multigrid Methods for Boundary Value Problems I,” *Computers in Physics* **5**(5), 514 (1991).

Edward M. Purcell, *Electricity and Magnetism*, second edition, Berkeley Physics Course, Vol. 2, McGraw-Hill (1985). A well known text that discusses the relaxation method.

John R. Reitz, Frederick J. Milford, and Robert W. Christy, *Foundations of Electromagnetic Theory*, third edition, Addison-Wesley (1979). This text discusses microwave cavity resonators.

Matthew N. O. Sadiku, *Numerical Techniques in Electromagnetics*, CRC Press (1992).

A. Taflove and M. E. Brodwin, “Numerical solution of steady state electromagnetic scattering problems using the time dependent Maxwell equations,” *IEEE Trans. Microwave Theory and Techniques* **23**, 623 (1975). The authors derive the stability conditions for the Yee algorithm.

P. B. Visscher, *Fields and Electrodynamics*, John Wiley & Sons (1988). An intermediate level text that incorporates computer simulations and analysis into its development.

P. J. Walker and I. D. Johnston, “Computer model clarifies spontaneous charge distribution in conductors,” *Computers in Physics* **9**, 42 (1995).

Gregg Williams, “An introduction to relaxation methods,” *Byte* **12**(1), 111 (1987). The author discusses the application of relaxation methods to the solution of the two-dimensional Poisson’s equation.

K. S. Yee, *IEEE Trans. Antennas and Propagation* **14**, 302 (1966). Yee uses the discretized Maxwell’s equations to model the scattering of electromagnetic waves off a perfectly conducting rectangular obstacle.

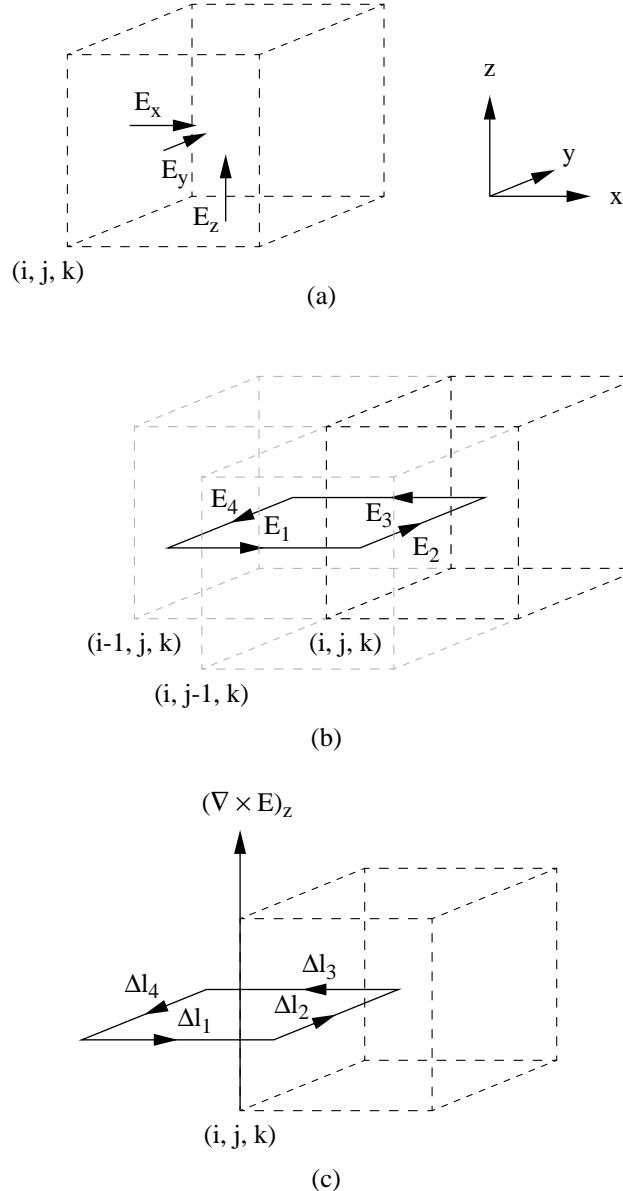


Figure 10.6: Calculation of the curl of the vector \mathbf{E} defined on the faces of a cube. (a) The face vector \mathbf{E} associated with the cube (i, j, k) . The components associated with the left, front, and bottom faces are $E_x(i, j, k)$, $E_y(i, j, k)$, $E_z(i, j, k)$ respectively. (b) The components E_i on the faces that share the front left edge of the cube (i, j, k) . $E_1 = E_x(i, j - 1, k)$, $E_2 = E_y(i, j, k)$, $E_3 = -E_x(i, j, k)$, and $E_4 = -E_y(i - 1, j, k)$. The cubes associated with E_1 and E_4 also are shown. (c) The vector components Δl_i on the faces that share the left front edge of the cube. (The z -component of the curl of \mathbf{E} defined on the left edge points in the positive z direction.)

Chapter 11

Numerical Integration and Monte Carlo Methods

©2001 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
10 April 2001

Simple classical and Monte Carlo methods are illustrated in the context of the numerical evaluation of definite integrals.

11.1 Numerical Integration Methods in One Dimension

Monte Carlo methods were introduced in Chapter 7 in the context of systems that are intrinsically random. In this chapter we will find that we can use sequences of random numbers to estimate definite integrals, a problem that seemingly has nothing to do with randomness. To place the Monte Carlo numerical integration methods in perspective, we will first discuss several common classical methods of determining the numerical value of definite integrals. We will see that these classical methods, although usually preferable in low dimensions, are impractical for multidimensional integrals and that Monte Carlo methods are essential for the evaluation of the latter if the number of dimensions is sufficiently high.

Consider the one-dimensional definite integral of the form

$$F = \int_a^b f(x) dx. \quad (11.1)$$

For some choices of the integrand $f(x)$, the integration in (11.1) can be done analytically, found in tables of integrals, or evaluated as a series. However, there are relatively few functions that can be evaluated analytically and most functions must be integrated numerically.

The classical methods of numerical integration are based on the geometrical interpretation of the integral (11.1) as the area under the curve of the function $f(x)$ from $x = a$ to $x = b$ (see

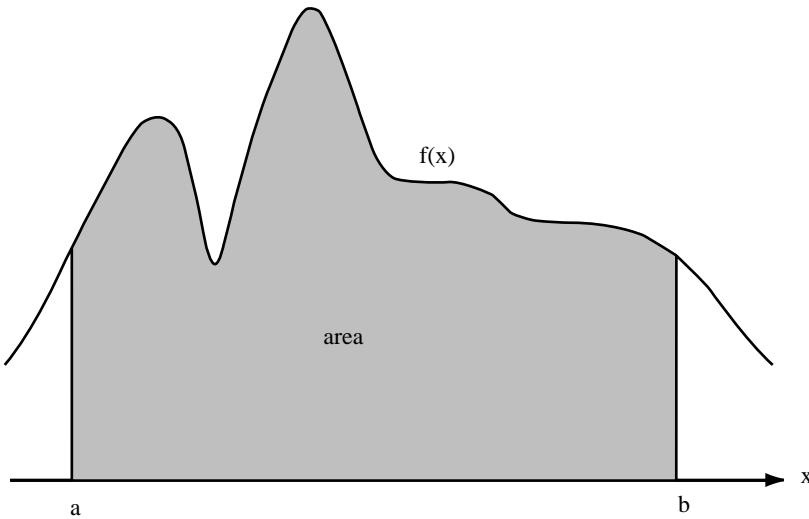


Figure 11.1: The integral F equals the area under the curve $f(x)$.

Figure 11.1). In these methods the x -axis is divided into n equal intervals of width Δx , where Δx is given by

$$\Delta x = \frac{b - a}{n}, \quad (11.2a)$$

and

$$x_n = x_0 + n \Delta x. \quad (11.2b)$$

In the above, $x_0 = a$ and $x_n = b$.

The simplest approximation of the area under the curve $f(x)$ is the sum of rectangles shown in Figure 11.2. In the *rectangular* approximation, $f(x)$ is evaluated at the *beginning* of the interval, and the approximate F_n of the integral is given by

$$F_n = \sum_{i=0}^{n-1} f(x_i) \Delta x. \quad (\text{rectangular approximation}) \quad (11.3)$$

In the *trapezoidal* approximation or rule the integral is approximated by computing the area under a trapezoid with one side equal to $f(x)$ at the beginning of the interval and the other side equal to $f(x)$ at the end of the interval. This approximation is equivalent to replacing the function by a straight line connecting the values of $f(x)$ at the beginning and the end of each interval. Because the approximate area under the curve from x_i to x_{i+1} is given by $\frac{1}{2}[f(x_{i+1}) + f(x_i)]\Delta x$, the total area F_n is given by

$$F_n = \left[\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(x_n) \right] \Delta x. \quad (\text{trapezoidal rule}) \quad (11.4)$$

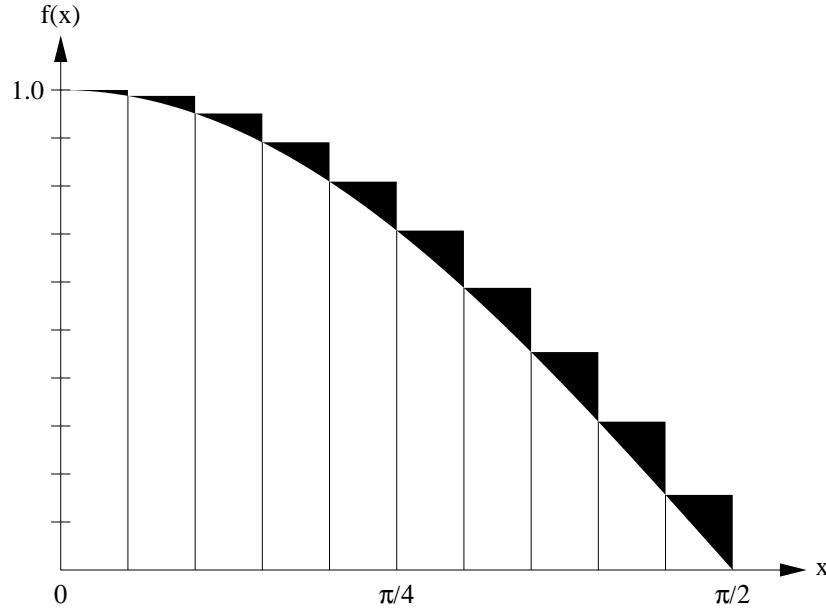


Figure 11.2: The rectangular approximation for $f(x) = \cos x$ for $0 \leq x \leq \pi/2$. The error in the rectangular approximation is shaded. Numerical values of the error for various values of the number of intervals n are given in Table 11.1.

A generally more accurate method is to use a quadratic or parabolic interpolation procedure through adjacent triplets of points. For example, the equation of the second-order polynomial that passes through the points (x_0, y_0) , (x_1, y_1) , and (x_2, y_2) can be written as

$$\begin{aligned} y(x) &= y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \\ &\quad + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}. \end{aligned} \quad (11.5)$$

What is the value of $y(x)$ at $x = x_1$? The area under the parabola $y(x)$ between x_0 and x_2 can be found by simple integration and is given by

$$F_0 = \frac{1}{3} (y_0 + 4y_1 + y_2) \Delta x, \quad (11.6)$$

where $\Delta x = x_1 - x_0 = x_2 - x_1$. The total area under all the parabolic segments yields the parabolic approximation for the total area:

$$\begin{aligned} F_n &= \frac{1}{3} [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \dots \\ &\quad + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)] \Delta x. \end{aligned} \quad (\text{Simpson's rule}) \quad (11.7)$$

This approximation is known as *Simpson's rule*. Note that Simpson's rule requires that n be even.

In practice, Simpson's rule is adequate for functions $f(x)$ that are reasonably well behaved, that is, functions that can be adequately represented by a polynomial. If $f(x)$ is such a function, we can adopt the strategy of evaluating the area for a given number of intervals n and then doubling the number of intervals and evaluating the area again. If the two evaluations are sufficiently close to one another, we may stop. Otherwise, we again double n until we achieve the desired accuracy. Of course, this strategy will fail if $f(x)$ is not well-behaved. An example of a poorly-behaved function is $f(x) = x^{-1/3}$ at $x = 0$, where $f(x)$ diverges. Another example where this strategy might fail is when a limit of integration is equal to $\pm\infty$. In many cases we can eliminate the problem by a change of variables.

We first list classes `NumericalIntegration` and `RectangularApproximator`. What are the main features of `NumericalIntegration`?

```
// numerical integration of f(x) from x = a to x = b
package edu.clarku.sip.chapter11;
import edu.clarku.sip.templates.*;
import edu.clarku.sip.graphics.*;
import java.awt.*;
import java.awt.geom.*;
import java.text.NumberFormat;

public class NumericalIntegration implements Model
{
    private Function cosineFunction;
    private FunctionDrawer functionDrawer;           // draw function
    // compute sum and draw rectangles
    private RectangularApproximator rectangularApproximator;
    private Control myControl = new SControl(this);
    private World2D myWorld = new World2D();
    private NumberFormat numberFormat = NumberFormat.getInstance();

    public NumericalIntegration()
    {
        cosineFunction = new CosineFunction();
        rectangularApproximator = new RectangularApproximator(cosineFunction);
        functionDrawer = new FunctionDrawer(cosineFunction);
        myWorld.addDrawable(functionDrawer);
        myWorld.addDrawable(rectangularApproximator);
        numberFormat.setMaximumFractionDigits(4);
    }

    public void reset()
    {
        myControl.setValue("lower limit a", 0);
        myControl.setValue("upper limit b", numberFormat.format(Math.PI/2));
    }
}
```

```

        myControl.setValue("n", 4);
    }

    public void calculate()
    {
        double a = myControl.getValue("lower limit a");
        double b = myControl.getValue("upper limit b");
        int n = (int) myControl.getValue("n");      // number of intervals
        double dx = (b - a)/n; // width of interval
        functionDrawer.initialize(a, b, 0.01, false);
        double ymin = 0;
        double ymax = 1;
        myWorld.setXYMinMax(a, b, ymin, ymax);
        double areaUnderCurve = rectangularApproximator.computeArea(a, b, dx);
        myWorld.repaint();
        myControl.println("area under curve = " + areaUnderCurve);
    }

    public static void main(String[] args)
    {
        NumericalIntegration integ = new NumericalIntegration();
        integ.reset();
    }
}

```

The new classes that we have used in class `RectangularApproximator` are `Vector` and `Rectangle2D`. The `Vector` class is part of the `java.util` package which provides various container classes for storing and managing objects. Vectors can be thought of as dynamic arrays that can change size while the program is running. They are used in the following to store the rectangles that will be drawn. Vectors are less efficient than arrays and the elements in a vector must be objects. The simplest way to store an object in a `Vector` is to use the `add()` method.

```

package edu.clarku.sip.chapter11;
import java.awt.*;
import java.awt.geom.*;
import java.util.Vector;
import edu.clarku.sip.graphics.*;

public class RectangularApproximator implements Drawable
{
    private Function function;
    private Vector rectangles = new Vector();

    public RectangularApproximator(Function f)
    {
        function = f;
    }
}

```

```

}

public double computeArea(double a, double b, double dx)
{
    rectangles.clear();
    double x = a;
    double sum = function.evaluate(x);
    while (x < b)
    {
        double ytop = function.evaluate(x);
        double xleft = x;
        double height = ytop;
        double width = dx;
        // Rectangle is (already defined) Shape in Java 2D
        /* coordinates of rectangle are world coordinates; must convert
           them to pixel coordinates before drawing */
        Rectangle2D.Double r = new Rectangle2D.Double(xleft, ytop, width, height);
        rectangles.add(r);      // add rectangle to vector of rectangles
        x = x + dx;
        sum = sum + function.evaluate(x);
    }
    return sum*dx;
}

public void draw(World2D myWorld, Graphics g)
{
    Graphics2D g2 = (Graphics2D) g;
    g2.setColor(Color.red);
    Rectangle2D.Double worldRect = null; // Rectangle in world coordinates
    Rectangle2D.Double pixRect = null;   // Rectangle in pixel coordinates
    // draw rectangles
    for (int i = 0; i < rectangles.size(); i++)
    {
        worldRect = (Rectangle2D.Double) rectangles.get(i); // get Rectangle from Vector
        double px = myWorld.xToPix(worldRect.x);
        double py = myWorld.yToPix(worldRect.y);
        double pwidth = myWorld.xToPix(worldRect.x + worldRect.width) - px;
        double pheight = myWorld.yToPix(0) - myWorld.yToPix(worldRect.height);
        pixRect = new Rectangle2D.Double(px, py, pwidth, pheight);
        g2.draw(pixRect);
    }
}
}

```

The class `FunctionDrawer` defines a object that can draw itself using an object of type `GeneralPath`. The latter represents a geometric path constructed from straight lines, quadratic, and cubic curves.

What is the function of each method that was used? An affine transform is used to transform the coordinate system.

```
package edu.clarku.sip.chapter11;
import edu.clarku.sip.graphics.*;
import java.awt.*;
import java.awt.geom.*;
// class draws a function from xmin to xmax

public class FunctionDrawer implements Drawable
{
    private double ymin, ymax; // y min and max values of function in specified range
    // GeneralPath class represents a geometric path constructed from straight
    // lines, quadratic, and cubic curves
    private GeneralPath path = new GeneralPath();
    private Function function;
    private boolean filled = false; // fill area under curve with drawing color
    public Color color = Color.black;

    public FunctionDrawer(Function f)
    {
        function = f;
    }

    public double getYMin(){return ymin;}

    public double getYMax(){return ymax;}

    public void initialize(double xmin, double xmax, double stepSize, boolean _filled)
    {
        filled = _filled;
        // reset path to empty
        path.reset();
        path.moveTo((float) xmin, (float) function.evaluate(xmin));
        ymin = function.evaluate(xmin);
        ymax = ymin; // starting values for ymin and ymax
        if (filled)
        {
            path.moveTo((float) xmin, 0);
            path.lineTo((float) xmin, (float) ymin);
        }
        else
            path.moveTo((float)xmin, (float)ymin);
        for (double x = xmin + stepSize; x <= xmax; x = x + stepSize)
        {
            // add point to path by drawing straight line from current to specified coordinates
        }
    }
}
```

```

        double y = function.evaluate(x);
        path.lineTo((float) x,(float) y);
        ymin = Math.min(ymin, y);
        ymax = Math.max(ymax, y);
    }
    if (filled)
    {
        path.lineTo((float)xmax, 0);
        path.closePath();
    }
}

public void draw(World2D myWorld, Graphics g)
{
    Graphics2D g2 = (Graphics2D) g;
    g2.setColor(color);
    /*AffineTransform class performs linear mapping from 2D coordinates
     * to other 2D coordinates preserving straightness and parallelness of lines */
    /*AffineTransform(double m00,double m10,double m01,double m11,double m02,double m12)
     [x']  [m00  m01  m02] [x]  [m00x + m01y + m02]
     [y']  [m10  m11  m12] [y]  [m10x + m11y + m12]
     [1 ]  [ 0    0    1 ] [1]  [           1           ] */
    double m00 = myWorld.getXPixPerUnit();
    double m11 = -myWorld.getYPixPerUnit();
    double m02 = myWorld.getXOffset();
    double m12 = myWorld.getSize().height - myWorld.getYOffset();
    AffineTransform transform = new AffineTransform(m00, 0, 0, m11, m02, m12);
    /* return new Shape object defined by geometry of specified Shape after
     * it has been transformed */
    Shape s = path.createTransformedShape(transform);
    if (filled)
        g2.fill(s);
    else
        g2.draw(s);
}
}

package edu.clarku.sip.chapter11;
public class CosineFunction implements Function
{
    public double evaluate(double x)
    {
        return Math.cos(x);
    }
}

```

```

package edu.clarku.sip.chapter11;
public interface Function
{
    public double evaluate(double x);
}

```

Let us consider the accuracy of the rectangular approximation for the integral of $f(x) = \cos x$ from $x = 0$ to $x = \pi/2$ by comparing the numerical results shown in Table 11.1 with the exact answer of unity. Note that the error decreases as n^{-1} . This observed n dependence of the error is consistent with the analytical derivation of the n dependence of the error obtained in Appendix 11A. We explore the n dependence of the error associated with other numerical integration methods in Problems 11.1 and 11.2.

n	F_n	Δ_n
2	1.34076	0.34076
4	1.18347	0.18347
8	1.09496	0.09496
16	1.04828	0.04828
32	1.02434	0.02434
64	1.01222	0.01222
128	1.00612	0.00612
256	1.00306	0.00306
512	1.00153	0.00153
1024	1.00077	0.00077

Table 11.1: Rectangular approximations of the integral of $\cos x$ from $x = 0$ to $x = \pi/2$ as a function of n , the number of intervals. The error Δ_n is the difference between the rectangular approximation and the exact result of unity. Note that the error Δ_n decreases approximately as n^{-1} , that is, if n is increased by a factor of 2, Δ_n decreases by a factor 2.

Problem 11.1. The rectangular and midpoint approximations

- Test the above program by reproducing the results in Table 11.1.
- Use the rectangular approximation to determine numerical approximations for the definite integrals of $f(x) = 2x + 3x^2 + 4x^3$ and $f(x) = e^{-x}$ for $0 \leq x \leq 1$ and $f(x) = 1/x$ for $a \leq x \leq 2$. What is the approximate n dependence of the error in each case?
- A straightforward modification of the rectangular approximation is to evaluate $f(x)$ at the *midpoint* of each interval. Define a `MidpointApproximator` class by making the necessary modifications and approximate the integral of $f(x) = \cos x$ in the interval $0 \leq x \leq \pi/2$. How does the magnitude of the error compare with the results shown in Table 11.1? What is the approximate dependence of the error on n ?
- Use the midpoint approximation to determine the definite integrals considered in part (b). What is the approximate n dependence of the error in each case? Given that our goal is to compute

the integral as accurately as possible with the smallest number of function evaluations of the integrand, which approximation would you choose?

Problem 11.2. The trapezoidal and Simpson's rule

- Modify your program so that the trapezoidal rule is computed and determine the n -dependence of the error for the same functions as in Problem (11.1b).
- It is possible to double the number of intervals without losing the benefit of the previous calculations. For $n = 1$, the trapezoidal rule is proportional to the average of $f(a)$ and $f(b)$. In the next approximation the value of f at the midpoint is added to this average. The next refinement adds the values of f at the $1/4$ and $3/4$ points. Modify your program so that the number of intervals is doubled each time and the results of previous calculations are used. The following pseudocode should be helpful:

```

if (n == 1) then
    sum = 0.5*(b - a)*(f(a) + f(b));
else
{
    int nadd = (int) Math.pow(2.0, n); // additional intervals
    double delta = (b - a)/nadd;
    double x = a + 0.5*delta;
    double sumit = 0.0; // intermediate sum
    for (int i = 1; i <= nadd; i++)
    {
        sumit = sumit + f(x);
        x = x + delta;
    }
    sum = 0.5*(sum + (b - a)*sum/nadd);
}

```

- Use the trapezoidal approximation to approximate the integrals of $f(x) = \cos x$ for $0 \leq x \leq \pi/2$ and $f(x) = 2x + 3x^2 + 4x^3$ and $f(x) = e^{-x}$ for $0 \leq x \leq 1$. What is the approximate n dependence of the error in each case? Which approximation yields the best results for the same computation time?
- Either adapt your program so that it uses Simpson's rule directly or convince yourself that the result of Simpson's rule can be expressed as

$$S_n = (4T_{2n} - T_n)/3, \quad (11.8)$$

where T_n is the result from the trapezoidal rule for n intervals. Determine the same integrals as in part (c) and discuss the relative merits of the various approximations.

- Use Simpson's rule to approximate the integral of $f(x) = (2\pi)^{-1/2} e^{-x^2}$ for $-1 \leq x \leq 1$. Do you obtain the same result by choosing the interval $[0, 1]$ and then multiplying by two? Why or why not?

- f. Evaluate the integral of the function $f(x) = 4\sqrt{1-x^2}$ for $-1 \leq x \leq 1$. What value of n is needed for four decimal accuracy? The reason for the slow convergence can be understood by reading Appendix 11A.
- g. So far, our strategy for numerically estimating the value of definite integrals has been to choose one or more of the classical integration formulae and to compute F_n and F_{2n} for reasonable values of n . If the difference $|F_{2n} - F_n|$ is too large, then we double n until the desired accuracy is reached. The success of this strategy is based on the implicit assumption that the sequence F_n, F_{2n}, \dots converges to the true integral F . Is there a way of extrapolating this sequence to the limit? Let us explore this idea by using the trapezoidal approximation. Because the error for this approximation decreases approximately as n^{-2} , we can write $F = F_n + Cn^{-2}$, and plot F_n as a function of n^{-2} to obtain the extrapolated result F . Apply this procedure to the integrals considered in some of the above problems and compare your results to those found from the trapezoidal approximation and Simpson's rule alone. A more sophisticated application of this idea is known as *Romberg* integration (cf. Press et al.).

11.2 Simple Monte Carlo Evaluation of Integrals

We now explore a totally different method of estimating integrals. Let us introduce this method by asking, ‘Suppose the pond is in the middle of a field of known area A . One way to estimate the area of the pond is to throw the stones so that they land at random within the boundary of the field and count the number of splashes that occur when a stone lands in a pond. The area of the pond is approximately the area of the field times the fraction of stones that make a splash. This simple procedure is an example of a *Monte Carlo* method.

More explicitly, imagine a rectangle of height h , width $(b-a)$, and area $A = h(b-a)$ such that the function $f(x)$ is within the boundaries of the rectangle (see Figure 11.3). Compute n pairs of random numbers x_i and y_i with $a \leq x_i \leq b$ and $0 \leq y_i \leq h$. The fraction of points x_i, y_i that satisfy the condition $y_i \leq f(x_i)$ is an estimate of the ratio of the integral of $f(x)$ to the area of the rectangle. Hence, the estimate F_n in the *hit or miss* method is given by

$$F_n = A \frac{n_s}{n}, \quad (\text{hit or miss method}) \quad (11.9)$$

where n_s is the number of “splashes” or points below the curve, and n is the total number of points. The number of *trials* n in (11.9) should not be confused with the number of intervals used in the numerical methods discussed in Section 11.1.

Another Monte Carlo integration method is based on the mean-value theorem of calculus, which states that the definite integral (11.1) is determined by the average value of the integrand $f(x)$ in the range $a \leq x \leq b$. To determine this average, we choose the x_i at random instead of at regular intervals and *sample* the value of $f(x)$. For the one-dimensional integral (11.1), the estimate F_n of the integral in the *sample mean* method is given by

$$F_n = (b-a) \langle f \rangle = (b-a) \frac{1}{n} \sum_{i=1}^n f(x_i). \quad (\text{sample mean method}) \quad (11.10)$$

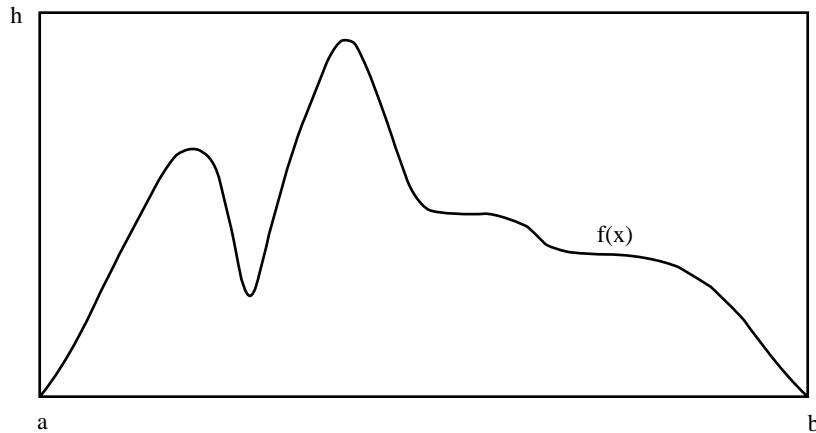


Figure 11.3: The function $f(x)$ is in the domain determined by the rectangle of height H and width $(b - a)$.

The x_i are random numbers distributed uniformly in the interval $a \leq x_i \leq b$, and n is the number of trials. Note that the forms of (11.3) and (11.10) are formally identical except that the n points are chosen with equal spacing in (11.3) and with random spacing in (11.10). We will find that for low dimensional integrals (11.3) is more accurate, but for higher dimensional integrals (11.10) does better.

A simple program that implements the hit or miss method is given below. Note the use of the `Random` class and the methods `setSeed` and `nextDouble()`. The primary reason that it is desirable to specify the seed rather than to choose it more or less at random from the time (as done by `Math.random()`) is that it is convenient to use the same random number sequence when testing a Monte Carlo program. In Section ??.

```
package edu.clarku.sip.chapter11;
import edu.clarku.sip.templates.*;
import java.util.Random;

public class MonteCarloEstimation implements Model
{
    private Control myControl = new SControl(this);
    private Random rnd = new Random();
    private int n;                      // number of trials
    private long seed;

    public double evaluate(double x)
    {
        return Math.sqrt(1 - x*x);
    }
}
```

```

public void calculate()
{
    double ymax = 1.0;
    double a = 0;
    double b = 1.0;
    n = (int) myControl.getValue("n");
    seed = (long) myControl.getValue("seed");
    rnd.setSeed(seed);
    long hits = 0;
    double x, y;

    for (int i = 0; i < n; i++)
    {
        // nextDouble returns random double between 0 (inclusive) and 1 (exclusive)
        x = rnd.nextDouble()*(b-a);
        y = rnd.nextDouble()*ymax;
        if ( y <= evaluate(x) )
            hits++;
    }

    double estimatedArea = (hits*(b-a)*ymax)/n;
    myControl.println("estimated area = " + estimatedArea);
}

public void reset()
{
    myControl.setValue("n", 1000);
    myControl.setValue("seed", 1379);
}

public static void main(String[] args)
{
    new MonteCarloEstimation().reset();
}
}

```

Problem 11.3. Monte Carlo integration in one dimension

- Use the hit or miss Monte Carlo method to estimate F_n , the integral of $f(x) = 4\sqrt{1 - x^2}$ in the interval $0 \leq x \leq 1$ as a function of n . Choose $a = 0$, $b = 1$, $h = 1$, and compute the mean value of the function $\sqrt{1 - x^2}$. Multiply the estimate by 4 to determine F_n . Calculate the difference between F_n and the exact result of π . This difference is a measure of the error associated with the Monte Carlo estimate. Make a log-log plot of the error as a function of n . What is the approximate functional dependence of the error on n for large n , for example, $n \geq 10^4$?
- Estimate the same integral using the sample mean Monte Carlo method (11.10) and compute

the error as a function of the number of trials n for $n \geq 10^4$. How many trials are needed to determine F_n to two decimal places? What is the approximate functional dependence of the error on n for large n ?

- c. Determine the computational time per trial using the two Monte Carlo methods. Which Monte Carlo method is preferable?

11.3 *Numerical Integration of Multidimensional Integrals

Many problems in physics involve averaging over many variables. For example, suppose we know the position and velocity dependence of the total energy of ten interacting particles. In three dimensions each particle has three velocity components and three position components. Hence the total energy is a function of 60 variables, and a calculation of the average energy per particle involves computing a $d = 60$ dimensional integral. (More accurately, the total energy is a function of $60 - 6 = 54$ variables if we use center of mass and relative coordinates.) If we divide each coordinate into p intervals, there would be p^{60} points to sum. Clearly, standard numerical methods such as Simpson's rule would be impractical for this example.

A discussion of the n dependence of the error associated with the standard numerical methods for d -dimensional integrals is given in Appendix 11A. We show that if the error decreases as n^{-a} for $d = 1$, then the error decreases as $n^{-a/d}$ in d dimensions. In contrast, we find (see Section 11.4) that the error for all Monte Carlo integration methods decreases as $n^{-1/2}$ independently of the dimension of the integral. Because the computational time is roughly proportional to n in both the classical and Monte Carlo methods, we conclude that for low dimensions, the classical numerical methods such as Simpson's rule are preferable to Monte Carlo methods unless the domain of integration is very complicated. However, the error in the conventional numerical methods increases with dimension (see Appendix 11A), and Monte Carlo methods are essential for higher dimensional integrals.

To illustrate the general method for evaluating multidimensional integrals, we consider the two-dimensional integral

$$F = \int_R f(x, y) dx dy, \quad (11.11)$$

where R denotes the region of integration. The extension to higher dimensions is straightforward, but tedious. Form a rectangle that encloses the region R , and divide this rectangle into squares of length h . Assume that the rectangle runs from x_a to x_b in the x direction and from y_a to y_b in the y direction. The total number of squares is $n_x n_y$, where $n_x = (x_b - x_a)/h$ and $n_y = (y_b - y_a)/h$. If we use the midpoint approximation, the integral F is estimated by

$$F \approx \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} f(x_i, y_j) H(x_i, y_j) h^2, \quad (11.12)$$

where $x_i = x_a + (i - \frac{1}{2})h$, $y_j = y_a + (j - \frac{1}{2})h$, and the function $H(x, y)$ equals unity if (x, y) is in R and is zero otherwise.

A simple Monte Carlo method for evaluating a two-dimensional integral uses the same rectangular region as in the above, but the n points (x_i, y_i) are chosen at random within the rectangle. The estimate for the integral is then

$$F_n = \frac{A}{n} \sum_{i=1}^n f(x_i, y_i) H(x_i, y_i), \quad (11.13)$$

where A is the area of the rectangle. Note that if $f(x, y) = 1$ everywhere, then (11.13) is equivalent to the hit or miss method of calculating the area of the region R . In general, (11.13) represents the area of the region R multiplied by the average value of $f(x, y)$ in R . In Section 11.7 we discuss a more efficient Monte Carlo method for evaluating definite integrals.

Problem 11.4. Two-dimensional numerical integration

- Write a program to implement the midpoint approximation in two dimensions and integrate the function $f(x, y) = x^2 + 6xy + y^2$ over the region defined by the condition $x^2 + y^2 \leq 1$. Use $h = 0.1, 0.05, 0.025$, and if time permits 0.0125 . Display n , the number of squares, and the estimate for the integral.
- Repeat part (a) using a Monte Carlo method and the same number of points n . For each value of n repeat the calculation several times to obtain a crude estimate of the random error.

Problem 11.5. Volume of a hypersphere

- The interior of a d -dimensional hypersphere of unit radius is defined by the condition $x_1^2 + x_2^2 + \dots + x_d^2 \leq 1$. Write a program that finds the volume of a hypersphere using the midpoint approximation. If you are clever, you can write a program that does any dimension using recursive subroutines. Test your program for $d = 2$ and $d = 3$, and then find the volume for $d = 4$ and $d = 5$. Begin with $h = 0.2$, and decrease h until your results do not change by more than 1%, or until you run out of patience or resources.
- Repeat part (a) using a Monte Carlo technique. For each value of n , repeat the calculation several times to obtain a rough estimate of the random error. Is a program valid for any d easier to write in this case than in part (a)?

11.4 Monte Carlo Error Analysis

Both the classical numerical integration methods and the Monte Carlo methods yield approximate answers whose accuracy depends on the number of intervals or on the number of trials respectively. So far, we have used our knowledge of the exact value of various integrals to determine that the error in the Monte Carlo method approaches zero as approximately $n^{-1/2}$ for large n , where n is the number of trials. In the following, we will find how to estimate the error when the exact answer is unknown. Our main result is that the n dependence of the error is independent of the nature of the integrand and, most importantly, independent of the number of dimensions.

Because the appropriate measure of the error in Monte Carlo calculations is subtle, we first determine the error for an explicit example. Consider the Monte Carlo evaluation of the integral

of $f(x) = 4\sqrt{1 - x^2}$ in the interval $[0, 1]$ (see Problem 11.3). Our result for a particular sequence of $n = 10^4$ random numbers using the sample mean method is $F_n = 3.1489$. How does this result for F_n compare with your result found in Problem 11.3 for the same value of n ? By comparing F_n to the exact result of $F = \pi \approx 3.1416$, we find that the error associated with $n = 10^4$ trials is approximately 0.0073.

How can we estimate the error if the exact result is unknown? How can we know if $n = 10^4$ trials is sufficient to achieve the desired accuracy? Of course, we cannot answer these questions definitively because if the actual error in F_n were known, we could correct F_n by the required amount and obtain F . The best we can do is to calculate the *probability* that the true value F is within a certain range centered on F_n .

If the integrand were a constant, then the error would be zero, that is, F_n would equal F for any n . Why? This limiting behavior suggests that a possible measure of the error is the *variance* σ^2 defined by

$$\sigma^2 = \langle f^2 \rangle - \langle f \rangle^2, \quad (11.14)$$

where

$$\langle f \rangle = \frac{1}{n} \sum_{i=1}^n f(x_i), \quad (11.15a)$$

and

$$\langle f^2 \rangle = \frac{1}{n} \sum_{i=1}^n f(x_i)^2. \quad (11.15b)$$

From the definition of the *standard deviation* σ , we see that if f is independent of x , σ is zero. For our example and the same sequence of random numbers used to obtain $F_n = 3.1489$, we obtain $\sigma_n = 0.8850$. Because this value of σ is two orders of magnitude larger than the actual error, we conclude that σ cannot be a direct measure of the error. Instead, σ is a measure of how much the function $f(x)$ varies in the interval of interest.

Another clue to finding an appropriate measure of the error can be found by increasing n and seeing how the actual error decreases as n increases. In Table 11.2 we see that as n goes from 10^2 to 10^4 , the actual error decreases by a factor of 10, that is, as $\sim 1/n^{1/2}$. However, we also see that σ_n is roughly constant and is much larger than the actual error.

n	F_n	actual error	σ_n
10^2	3.0692	0.0724	0.8550
10^3	3.1704	0.0288	0.8790
10^4	3.1489	0.0073	0.8850

Table 11.2: Examples of Monte Carlo measurements of the mean value of $f(x) = 4\sqrt{1 - x^2}$ in the interval $[0, 1]$. The actual error is given by the difference $|F_n - \pi|$. The standard deviation σ_n is found using (11.14).

One way to obtain an estimate for the error is to make additional runs of n trials each. Each run of n trials yields a mean or *measurement* that we denote as M_α . In general, these

run	α	M_α	actual error
1		3.1489	0.0073
2		3.1326	0.0090
3		3.1404	0.0012
4		3.1460	0.0044
5		3.1526	0.0110
6		3.1397	0.0019
7		3.1311	0.0105
8		3.1358	0.0058
9		3.1344	0.0072
10		3.1405	0.0011

Table 11.3: Examples of Monte Carlo measurements of the mean value of $f(x) = 4\sqrt{1 - x^2}$ in the interval $[0, 1]$. A total of 10 measurements of $n = 10^4$ trials each were made. The mean value M_α and the actual error $|M_\alpha - \pi|$ for each measurement are shown.

measurements are not equal because each measurement uses a different finite sequence of random numbers. Table 11.3 shows the results of ten separate measurements of $n = 10^4$ trials each. We see that the actual error varies from measurement to measurement. Qualitatively, the magnitude of the differences between the measurements is similar to the actual errors, and hence these differences are a measure of the error associated with a single measurement. To obtain a quantitative measure of this error, we determine the differences of these measurements using the *standard deviation of the means* σ_m which is defined as

$$\sigma_m^2 = \langle M^2 \rangle - \langle M \rangle^2, \quad (11.16)$$

where

$$\langle M \rangle = \frac{1}{m} \sum_{\alpha=1}^m M_\alpha, \quad (11.17a)$$

and

$$\langle M^2 \rangle = \frac{1}{m} \sum_{\alpha=1}^m M_\alpha^2. \quad (11.17b)$$

From the values of M_α in Table 11.3 and the relation (11.16), we find that $\sigma_m = 0.0068$. This value of σ_m is consistent with the results for the actual errors shown in Table 11.3 which we see vary from 0.00112 to 0.01098. Hence we conclude that σ_m , the standard deviation of the means, is a measure of the error for a single measurement. The more precise interpretation of σ_m is that a single measurement has a 68% chance of being within σ_m of the “true” mean. Hence the probable error associated with our first measurement of F_n with $n = 10^4$ is 3.149 ± 0.007 .

Although σ_m gives an estimate of the probable error, our method of obtaining σ_m by making additional measurements is impractical because we could have combined the additional measure-

ments to make a better estimate. In Appendix 11.8 we derive the relation

$$\sigma_m = \frac{\sigma}{\sqrt{n-1}} \quad (11.18a)$$

$$\approx \frac{\sigma}{\sqrt{n}}. \quad (11.18b)$$

The reason for the expression $1/\sqrt{n-1}$ in (11.18a) rather than $1/\sqrt{n}$ is similar to the reason for the expression $1/\sqrt{n-2}$ in the error estimates of the least squares fits (see (7.27c)). The idea is that to compute σ , we need to use n trials to compute the mean, $\langle f(x) \rangle$, and, loosely speaking, we have only $n-1$ independent trials remaining to calculate σ . Because we almost always make a large number of trials, we will use the relation (11.18b) and consider only this limit in Appendix 11A. Note that (11.18) implies that the most probable error decreases with the square root of the number of trials. For our example we find that the most probable error of our initial measurement is approximately $0.8850/100 \approx 0.009$, an estimate consistent with the known error of 0.007 and with our estimated value of $\sigma_m \approx 0.007$.

subset k	S_k
1	3.14326
2	3.15633
3	3.10940
4	3.15337
5	3.15352
6	3.11506
7	3.17989
8	3.12398
9	3.17565
10	3.17878

Table 11.4: The values of S_k for $f(x) = 4\sqrt{1-x^2}$ for $0 \leq x \leq 1$ is shown for 10 subsets of 10^3 trials each. The average value of $f(x)$ over the 10 subsets is 3.14892, in agreement with the result for F_n for the first measurement shown in Table 11.3.

One way to verify the relation (11.18) is to divide the initial measurement of n trials into s subsets. This procedure does not require additional measurements. We denote the mean value of $f(x_i)$ in the k th subset by S_k . As an example, we divide the 10^4 trials of the first measurement into $s = 10$ subsets of $n/s = 10^3$ trials each. The results for S_k are shown in Table 11.4. As expected, the mean values of $f(x)$ for each subset k are not equal. A reasonable candidate for a measure of the error is the standard deviation of the means of each subset. We denote this quantity as σ_s where

$$\sigma_s^2 = \langle S^2 \rangle - \langle S \rangle^2, \quad (11.19)$$

where the averages are over the subsets. From Table 11.4 we obtain $\sigma_s = 0.025$, a result that is approximately three times larger than our estimate of 0.007 for σ_m . However, we would like to define an error estimate that is independent of how we subdivide the data. This quantity is not σ_s ,

but the ratio σ_s/\sqrt{s} , which for our example is approximately $0.025/\sqrt{10} \approx 0.008$. This value is consistent with both σ_m and the ratio σ/\sqrt{n} . We conclude that we can interpret the n trials either as a single measurement or as a collection of s measurements with n/s trials each. In the former interpretation, the probable error is given by the standard deviation of the n trials divided by the square root of the number of trials. In the same spirit, the latter interpretation implies that the probable error is given by the standard deviation of the s measurements of the subsets divided by the square root of the number of measurements.

We can make the error as small as we wish by either increasing the number of trials or by increasing the efficiency of the individual trials and thereby reducing the standard deviation σ . Several *reduction of variance* methods are introduced in Sections 11.7 and 11.8.

Problem 11.6. Estimate of the Monte Carlo error

- a. Estimate the integral of $f(x) = e^{-x}$ in the interval $0 \leq x \leq 1$ using the sample mean Monte Carlo method with $n = 10^2$, $n = 10^3$, and $n = 10^4$. Compute the standard deviation σ as defined by (11.14). Does your estimate of σ change significantly as n is increased? Determine the exact answer analytically and estimate the n dependence of the error. How does your estimated error compare with the error estimate obtained from the relation (11.18)?
- b. Generate nineteen additional measurements of the integral each with $n = 10^3$ trials. Compute σ_m , the standard deviation of the twenty measurements. Is the magnitude of σ_m consistent with your estimate of the error obtained in part (a)? Will your estimate of σ_m change significantly if more measurements are made?
- c. Divide your first measurement of $n = 10^3$ trials into $s = 20$ subsets of 50 trials each. Compute the standard deviation of the subsets σ_s . Is the magnitude σ_s/\sqrt{s} consistent with your previous error estimates?
- d. Divide your first measurement into $s = 10$ subsets of 100 trials each and again compute the standard deviation of the subsets. How does the value of σ_s compare to what you found in part (c)? What is the value of σ_s/\sqrt{s} in this case? How does the standard deviation of the subsets compare using the two different divisions of the data?
- e. Estimate the integral

$$\int_0^1 e^{-x^2} dx \quad (11.20)$$

to two decimal places using σ_n/\sqrt{n} as an estimate of the probable error.

**Problem 11.7.* Importance of randomness

We will learn in Chapter ?? that the random number generator included with many programming languages is based on the linear congruential method. In this method each term in the sequence can be found from the preceding one by the relation

$$x_{n+1} = (ax_n + c) \bmod m, \quad (11.21)$$

where x_0 is the seed, and a , c , and m are nonnegative integers. The random numbers r in the unit interval $0 \leq r < 1$ are given by $r_n = x_n/m$. The notation $y = x \bmod m$ means that if x exceeds m ,

then the *modulus* m is subtracted from x as many times as necessary until $0 \leq y < m$. Eventually, the sequence of numbers generated by (11.21) will repeat itself, yielding a *period* for the random number generator. To examine the effect of a poor random number generator, we choose values of x_0 , m , a , and c such that (11.21) has poor statistical properties, for example, a short period. What is the period for $x_0 = 1$, $a = 5$, $c = 0$, and $m = 32$? Estimate the integral in Problem a by making a single measurement of $n = 10^3$ trials using the “random number” generator (11.21) with the above values of x_0 , a , c , and m . Analyze your measurement in the same way as before, that is, calculate the mean, the mean of each of the twenty subsets, and the standard deviation of the means of the subsets. Then divide your data into ten subsets and calculate the same quantities. Are the standard deviations of the subsets related as before? If not, why?

11.5 Nonuniform Probability Distributions

In the previous two sections we learned how uniformly distributed random numbers can be used to estimate definite integrals. We will find that it is desirable to sample the integrand $f(x)$ more often in regions of x where the magnitude of $f(x)$ is large or rapidly varying. Because such *importance sampling* methods require nonuniform probability distributions, we first consider several methods for generating random numbers that are not distributed uniformly before considering importance sampling in Section 11.7. In the following, we will denote r as a member of a uniform random number sequence in the unit interval $0 \leq r < 1$.

Suppose that two discrete events occur with probabilities p_1 and p_2 such that $p_1 + p_2 = 1$. How can we choose the two events with the correct probabilities using a uniform probability distribution? For this simple case, it is obvious that we choose event 1 if $r < p_1$; otherwise, we choose event 2. If there are three events with probabilities p_1 , p_2 , and p_3 , then if $r < p_1$ we choose event 1; else if $r < p_1 + p_2$, we choose event 2; else we choose event 3. We can visualize these choices by dividing a line segment of unit length into three pieces whose lengths are as shown in Figure 11.4. A random point r on the line segment will land in the i th segment with a probability equal to p_i .

Now consider n discrete events. How do we determine which event, i , to choose given the value of r ? The generalization of the procedure we have followed for $n = 2$ and 3 is to find the value of i that satisfies the condition

$$\sum_{j=0}^{i-1} p_j \leq r \leq \sum_{j=0}^i p_j, \quad (11.22)$$

where we have defined $p_0 \equiv 0$. Check that (11.22) reduces to the correct procedure for $n = 2$ and $n = 3$.

Now let us consider a continuous nonuniform probability distribution. One way to generate such a distribution is to take the limit of (11.22) and associate p_i with $p(x) dx$, the *probability density* $p(x)$, is defined such that $p(x) dx$ is the probability that the event x is in the interval between x and $x + dx$. The probability density $p(x)$ is normalized such that

$$\int_{-\infty}^{+\infty} p(x) dx = 1. \quad (11.23)$$

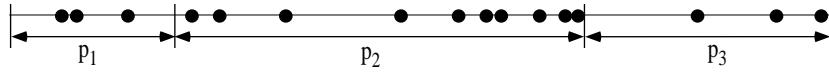


Figure 11.4: The unit interval is divided into three segments of lengths $p_1 = 0.2$, $p_2 = 0.5$, and $p_3 = 0.3$. Sixteen random numbers are represented by the filled circles uniformly distributed on the unit interval. The fraction of circles within each segment is approximately equal to the value of p_i for that segment.

In the continuum limit the two sums in (11.22) become the same integral and the inequalities become equalities. Hence we can write

$$P(x) \equiv \int_{-\infty}^x p(x') dx' = r. \quad (11.24)$$

From (11.24) we see that the uniform random number r corresponds to the *cumulative probability distribution* function $P(x)$, which is the probability of choosing a value less than or equal to x . The function $P(x)$ should not be confused with the probability density $p(x)$ or the probability $p(x) dx$. In many applications the meaningful range of values of x is positive. In that case, we have $p(x) = 0$ for $x < 0$.

The relation (11.24) leads to the *inverse transform* method for generating random numbers distributed according to the function $p(x)$. This method involves generating a random number r and solving (11.24) for the corresponding value of x . As an example of the method, we use (11.24) to generate a random number sequence according to the uniform probability distribution on the interval $a \leq x \leq b$. The desired probability density $p(x)$ is

$$p(x) = \begin{cases} (1/(b-a)), & a \leq x \leq b \\ 0, & \text{otherwise.} \end{cases} \quad (11.25)$$

The cumulative probability distribution function $P(x)$ for $a \leq x \leq b$ can be found by substituting (11.25) into (11.24) and performing the integral. The result is

$$P(x) = \frac{x-a}{b-a}. \quad (11.26)$$

If we substitute the form (11.26) for $P(x)$ into (11.24) and solve for x , we find the desired relation

$$x = a + (b-a)r. \quad (11.27)$$

The variable x given by (11.27) is distributed according to the probability distribution $p(x)$ given by (11.25). Of course, the relation (11.27) is rather obvious, and we already have used (11.27) in our Monte Carlo programs.

We next apply the inverse transform method to the probability density function

$$p(x) = \begin{cases} (1/\lambda) e^{-x/\lambda}, & \text{if } 0 \leq x \leq \infty \\ 0, & x < 0. \end{cases} \quad (11.28)$$

In Section 11.6 we will use this probability density to find the distance between scattering events of a particle whose mean free path is λ . If we substitute (11.28) into (11.24) and integrate, we find the relation

$$r = P(x) = 1 - e^{-x/\lambda}. \quad (11.29)$$

The solution of (11.29) for x yields $x = -\lambda \ln(1 - r)$. Because $1 - r$ is distributed in the same way as r , we can write

$$x = -\lambda \ln r. \quad (11.30)$$

The variable x found from (11.30) is distributed according to the probability density $p(x)$ given by (11.28). On many computers the computation of the natural logarithm in (11.30) is relatively slow, and hence the inverse transform method might not necessarily be the most efficient method to use.

From the above examples, we see that two conditions must be satisfied in order to apply the inverse transform method. Specifically, the form of $p(x)$ must allow the integral in (11.24) to be performed analytically, and it must be feasible to invert the relation $P(x) = r$ for x . The Gaussian probability density

$$p(x) = \frac{1}{(2\pi\sigma^2)^{1/2}} e^{-x^2/2\sigma^2} \quad (11.31)$$

is an example of a probability density for which the cumulative distribution $P(x)$ cannot be obtained analytically. However, we can generate the two-dimensional probability $p(x, y) dx dy$ given by

$$p(x, y) dx dy = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} dx dy. \quad (11.32)$$

First, we make a change of variables to polar coordinates:

$$r = (x^2 + y^2)^{1/2} \quad \theta = \tan^{-1} \frac{y}{x}. \quad (11.33)$$

Let $\rho = r^2/2$, and write the two-dimensional probability as

$$p(\rho, \theta) d\rho d\theta = \frac{1}{2\pi} e^{-\rho} d\rho d\theta, \quad (11.34)$$

where we have set $\sigma = 1$. If we generate ρ according to the exponential distribution (11.28) and generate θ uniformly in the interval $0 \leq \theta < 2\pi$, then the variables

$$x = (2\rho)^{1/2} \cos \theta \quad \text{and} \quad y = (2\rho)^{1/2} \sin \theta \quad (\text{Box-Muller method}) \quad (11.35)$$

will each be generated according to (11.31) with zero mean and $\sigma = 1$. (Note that the two-dimensional density (11.32) is the product of two independent one-dimensional Gaussian distributions.) This way of generating a Gaussian distribution is known as the *Box-Muller* method. We discuss other methods for generating the Gaussian distribution in Problem 11.12 and Appendix 11C.

Problem 11.8. Nonuniform probability densities

- Write a program to simulate the simultaneous rolling of two dice. In this case the events are discrete and occur with nonuniform probability. You might wish to revisit Problem 7.12 and simulate the game of craps.
- Write a program to verify that the sequence of random numbers $\{x_i\}$ generated by (11.30) is distributed according to the exponential distribution (11.28).
- Generate random variables according to the probability density function

$$p(x) = \begin{cases} 2(1-x), & \text{if } 0 \leq x \leq 1; \\ 0, & \text{otherwise.} \end{cases} \quad (11.36)$$

- Verify that the variables x and y in (11.35) are distributed according to the Gaussian distribution. What is the mean value and the standard deviation of x and of y ?
- How can you use the relations (11.35) to generate a Gaussian distribution with arbitrary mean and standard deviation?

11.6 *Neutron Transport

We now consider the application of a nonuniform probability distribution to the simulation of the transmission of neutrons through bulk matter, one of the original applications of a Monte Carlo method. Suppose that a neutron is incident on a plate of thickness t . We assume that the plate is infinite in the x and y directions and that the z axis is normal to the plate. At any point within the plate, the neutron can either be captured with probability p_c or scattered with probability p_s . These probabilities are proportional to the capture cross section and scattering cross section, respectively. If the neutron is scattered, we need to find its new direction as specified by the polar angle θ (see Figure 11.5). Because we are not interested in how far the neutron moves in the x or y direction, the value of the azimuthal angle ϕ is irrelevant.

If the neutrons are scattered equally in all directions, then the probability $p(\theta, \phi) d\theta d\phi$ equals $d\Omega/4\pi$, where $d\Omega$ is an infinitesimal solid angle and 4π is the total solid angle. Because $d\Omega = \sin \theta d\theta d\phi$, we have

$$p(\theta, \phi) = \frac{\sin \theta}{4\pi}. \quad (11.37)$$

We can find the probability density for θ and ϕ separately by integrating over the other angle. For example,

$$p(\theta) = \int_0^{2\pi} p(\theta, \phi) d\phi = \frac{1}{2} \sin \theta, \quad (11.38)$$

and

$$p(\phi) = \int_0^\pi p(\theta, \phi) d\theta = \frac{1}{2\pi}. \quad (11.39)$$

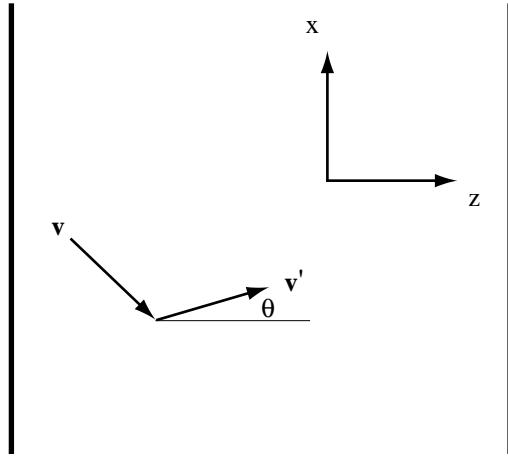


Figure 11.5: The definition of the scattering angle θ . The velocity before scattering is \mathbf{v} and the velocity after scattering is \mathbf{v}' . The scattering angle θ is independent of \mathbf{v} and is defined relative to the z axis.

Because the point probability $p(\theta, \phi)$ is the product of the probabilities $p(\theta)$ and $p(\phi)$, θ and ϕ are independent variables. Although we do not need to generate a random angle ϕ , we note that since $p(\phi)$ is a constant, ϕ can be found from the relation

$$\phi = 2\pi r. \quad (11.40)$$

To find θ according to the distribution (11.38), we substitute (11.38) in (11.24) and obtain

$$r = \frac{1}{2} \int_0^\theta \sin x \, dx \quad (11.41)$$

If we do the integration in (11.41), we find

$$\cos \theta = 1 - 2r. \quad (11.42)$$

Note that (11.40) implies that ϕ is uniformly distributed between 0 and 2π and (11.42) implies that $\cos \theta$ is uniformly distributed between -1 and $+1$.

We could invert the cosine in (11.42) to solve for θ . However, to find the z component of the path of the neutron through the plate, we need to multiply $\cos \theta$ by the path length ℓ , and hence we need $\cos \theta$ rather than θ . The path length, which is the distance traveled between subsequent scattering events, is obtained from the exponential probability density, $p(\ell) \propto e^{-\ell/\lambda}$ (see (11.28)). From (11.30), we have

$$\ell = -\lambda \ln r, \quad (11.43)$$

where λ is the mean free path.

Now we have all the necessary ingredients for calculating the probabilities for a neutron to pass through the plate, be reflected off the plate, or be captured and absorbed in the plate. The input parameters are the thickness of the plate t , the capture and scattering probabilities p_c and p_s , and the mean free path λ . We begin with $z = 0$, and implement the following steps:

1. Determine if the neutron is captured or scattered. If it is captured, then add one to the number of captured neutrons, and go to step 5.
2. If the neutron is scattered, compute $\cos \theta$ from (11.42) and ℓ from (11.43). Change the z coordinate of the neutron by $\ell \cos \theta$.
3. If $z < 0$, add one to the number of reflected neutrons. If $z > t$, add one to the number of transmitted neutrons. In either case, skip to step 5 below.
4. Repeat steps 1–3 until the fate of the neutron has been determined.
5. Repeat steps 1–4 with additional incident neutrons until sufficient data has been obtained.

Problem 11.9. Elastic neutron scattering

- a. Write a program to implement the above algorithm for neutron scattering through a plate. Assume $t = 1$ and $p_c = p_s/2$. Find the transmission, reflection, and absorption probabilities for the mean free path λ equal to 0.01, 0.05, 0.1, 1, and 10. Begin with 100 incident neutrons, and increase this number until satisfactory statistics are obtained. Give a qualitative explanation of your results.
- b. Choose $t = 1$, $p_c = p_s$, and $\lambda = 0.05$, and compare your results with the analogous case considered in part (a).
- c. Repeat part (b) with $t = 2$ and $\lambda = 0.1$. Do the various probabilities depend on λ and t separately or only on their ratio? Answer this question before doing the simulation.
- d. Draw some typical paths of the neutrons. From the nature of these paths, explain the results in parts (a)–(c). For example, how does the number of scattering events change as the absorption probability changes?

Problem 11.10. Inelastic neutron scattering

- a. In Problem 11.9 we assumed elastic scattering, that is, no energy is lost during scattering. Here we assume that some of the neutron energy E is lost and that the mean free path is proportional to the speed and hence to \sqrt{E} . Modify your program so that a neutron loses a fraction f of its energy at each scattering event, and assume that $\lambda = \sqrt{E}$. Consider $f = 0.05, 0.1$, and 0.5 , and compare your results with those found in Problem 11.9a.
- b. Make a histogram for the path lengths between scattering events and plot the path length distribution function for $f = 0.1, 0.5$, and 0 (elastic scattering).

The above procedure for simulating neutron scattering and absorption is more computer intensive than necessary. Instead of considering a single neutron at a time, we can consider a collection of neutrons at each position. Then instead of determining whether one neutron is captured or scattered, we determine the fraction that is captured and the fraction that is scattered. For example, at the first scattering site, a fraction p_c of the neutrons are captured and a fraction p_s are scattered. We accumulate the fraction p_c for the captured neutrons. We also assume that all the scattered neutrons move in the same direction with the same path length, both of which are generated at random as before. At the next scattering site there are p_s^2 scattered neutrons and $p_s p_c$ captured neutrons. At the end of m steps, the fraction of neutrons remaining is $w = p_s^m$ and the total fraction of captured neutrons is $p_c + p_c p_s + p_c p_s^2 + \dots + p_c p_s^{m-1}$. If the new position at the m th step is at $z < 0$, we add w to the sum for the reflected neutrons; if $z > t$, we add w to the neutrons transmitted. When the neutrons are reflected or absorbed, we start over again at $z = 0$ with another collection of neutrons.

Problem 11.11. Improved neutron scattering method Apply the improved Monte Carlo method to neutron transmission through a plate. Repeat the simulations suggested in Problem a and compare your new and previous results. Also compare the computational times for the two approaches to obtain comparable statistics.

The power of the Monte Carlo method becomes apparent when the geometry of the material is complicated or when the material is spatially nonuniform so that the cross sections vary from point to point. A difficult problem of current interest is the absorption of various forms of radiation in the human body.

Problem 11.12. Transmission through layered materials Consider two plates with the same thickness $t = 1$ that are stacked on top of one another with no space between them. For one plate, $p_c = p_s$, and for the other, $p_c = 2p_s$, that is, the top plate is a better absorber. Assume that $\lambda = 1$ in both plates. Find the transmission, reflection, and absorption probabilities for elastic scattering. Does it matter which plate receives the incident neutrons?

11.7 Importance Sampling

In Section 11.4 we found that the error associated with a Monte Carlo estimate is proportional to σ and inversely proportional to the square root of the number of trials. Hence, there are only two ways of reducing the error in a Monte Carlo estimate – either increase the number of trials or reduce the variance. Clearly the latter choice is desirable because it not require much more computer time. In this section we introduce *importance sampling* techniques that reduce σ and improve the efficiency of each trial.

In the context of numerical integration, we introduce a positive function $p(x)$ such that

$$\int_a^b p(x) dx = 1, \quad (11.44)$$

and rewrite the integral (11.1) as

$$F = \int_a^b \left[\frac{f(x)}{p(x)} \right] p(x) dx. \quad (11.45)$$

We can evaluate the integral (11.45) by sampling according to the probability distribution $p(x)$ and constructing the sum

$$F_n = \frac{1}{n} \sum_{i=1}^n \frac{f(x_i)}{p(x_i)}. \quad (11.46)$$

The sum (11.46) reduces to (11.10) for the uniform case $p(x) = 1/(b-a)$.

We wish to choose a form for $p(x)$ that minimizes the variance of the integrand $f(x)/p(x)$. Because we cannot evaluate σ analytically in general, we determine σ *a posteriori* and choose a form of $p(x)$ that mimics $f(x)$ as much as possible, particularly where $f(x)$ is large. A suitable choice of $p(x)$ would make the integrand $f(x)/p(x)$ slowly varying, and hence the variance will be reduced. As an example, we again consider the integral (see Problem ee)

$$F = \int_0^1 e^{-x^2} dx. \quad (11.47)$$

The estimate of F with $p(x) = 1$ for $0 \leq x \leq 1$ is shown in the first column of Table 11.5. A reasonable choice of a weight function is $p(x) = Ae^{-x}$, where A is chosen such that $p(x)$ is normalized on the unit interval. Note that this choice of $p(x)$ is positive definite and is qualitatively similar to $f(x)$. The results are shown in the second column of Table 11.5. We see that although the computation time per trial for the nonuniform case is larger, the smaller value of σ makes the use of the nonuniform probability distribution more efficient.

	$p(x) = 1$	$p(x) = Ae^{-x}$
n (trials)	4×10^5	8×10^3
F_n	0.7471	0.7469
σ	0.2010	0.0550
σ/\sqrt{n}	3×10^{-4}	6×10^{-4}
Total CPU time (s)	35	1.35
CPU time per trial (s)	10^{-4}	2×10^{-4}

Table 11.5: Comparison of the Monte Carlo estimates of the integral (11.47) using the uniform probability density $p(x) = 1$ and the nonuniform probability density $p(x) = Ae^{-x}$. The normalization constant A is chosen such that $p(x)$ is normalized on the unit interval. The value of the integral to four decimal places is 0.7468. The estimates F_n , variance σ , and the probable error $\sigma/n^{1/2}$ are shown. The CPU time (seconds) is shown for comparison only.

Problem 11.13. Importance sampling

- Choose $f(x) = \sqrt{1-x^2}$ as in the text and consider $p(x) = A(1-x)$ for $x \geq 0$. What is the value of A that normalizes $p(x)$ in the interval $[0, 1]$? What is the relation for the random variable x in terms of r assuming this form of the probability density $p(x)$? What is the variance of $f(x)/p(x)$ in the unit interval?
- Choose the importance function $p(x) = Ae^{-x}$ and evaluate the integral

$$\int_0^3 x^{3/2} e^{-x} dx. \quad (11.48)$$

- c. Choose $p(x) = Ae^{-\lambda x}$ and estimate the integral

$$\int_0^\pi \frac{1}{x^2 + \cos^2 x} dx. \quad (11.49)$$

Determine the value of λ that minimizes the variance of the integral.

An alternative approach is to use the known values of $f(x)$ at regular intervals to sample more often where $f(x)$ is relatively large. Because the idea is to use $f(x)$ itself to determine the probability of sampling, we only consider integrands $f(x)$ that are non-negative. To compute a rough estimate of the relative values of $f(x)$, we first compute its average value by taking k equally spaced points s_i and computing

$$S = \sum_{i=1}^k f(s_i) \quad (11.50)$$

This sum divided by k gives the average value of f in the interval. The approximate value of the integral is given by $F \approx Sh$, where $h = (b - a)/k$. This approximation of the integral is equivalent to the rectangular or mid-point approximation depending on where we compute the values of $f(x)$.

We then generate n random samples as follows. The probability of choosing subinterval i is given by the probability

$$p_i = \frac{f(s_i)}{S}. \quad (11.51)$$

Note that the sum over all subintervals of p_i is normalized to unity. To choose a subinterval with the desired probability, we generate a random number r uniformly in the interval $[a, b]$ and determine the subinterval i that satisfies the inequality (11.22). Now that the subinterval has been chosen with the desired probability, we generate a random number x_i in the subinterval $[s_i, s_i + h]$ and compute the ratio $f(x_i)/p(x_i)$.

The estimate of the integral is given by the following considerations. The probability p_i in (11.51) is the probability of choosing the subinterval i , not the probability of choosing a value of x between x and $x + \Delta x$. The probability $p(x)\Delta x$ is p_i times the probability of picking the particular value of x in subinterval i :

$$p(x_i)\Delta x = p_i \times \frac{\Delta x}{h}. \quad (11.52)$$

Hence, we have that

$$F_n = \frac{1}{n} \sum_{i=1}^n \frac{f(x_i)}{p(x_i)} = \frac{h}{n} \sum_{i=1}^n \frac{f(x_i)}{p_i}. \quad (11.53)$$

Problem 11.14. Apply the above method to estimate the integral of $f(x) = \sqrt{1 - x^2}$ in the unit interval.

11.8 Metropolis Algorithm

Another way of generating an arbitrary nonuniform probability distribution was introduced by Metropolis, Rosenbluth, Rosenbluth, Teller and Teller in 1953. The *Metropolis* algorithm is a special case of an importance sampling procedure in which certain possible sampling attempts are rejected (see Appendix 11C). The Metropolis method is useful for computing averages of the form

$$\langle f \rangle = \frac{\int p(x) f(x) dx}{\int p(x) dx}, \quad (11.54)$$

where $p(x)$ is an arbitrary probability distribution that need not be normalized. In Chapter ?? we will discuss the application of the Metropolis algorithm to problems in statistical mechanics.

For simplicity, we introduce the Metropolis algorithm in the context of estimating one-dimensional definite integrals. Suppose that we wish to use importance sampling to generate random variables according to an arbitrary probability density $p(x)$. The Metropolis algorithm produces a random walk of points $\{x_i\}$ whose asymptotic probability distribution approaches $p(x)$ after a large number of steps. The random walk is defined by specifying a *transition probability* $T(x_i \rightarrow x_j)$ from one value x_i to another value x_j such that the distribution of points x_0, x_1, x_2, \dots converges to $p(x)$. It can be shown that it is sufficient (but not necessary) to satisfy the “detailed balance” condition

$$p(x_i)T(x_i \rightarrow x_j) = p(x_j)T(x_j \rightarrow x_i). \quad (11.55)$$

The relation (11.55) does not specify $T(x_i \rightarrow x_j)$ uniquely. A simple choice of $T(x_i \rightarrow x_j)$ that is consistent with (11.55) is

$$T(x_i \rightarrow x_j) = \min \left[1, \frac{p(x_j)}{p(x_i)} \right]. \quad (11.56)$$

If the “walker” is at position x_i and we wish to generate x_{i+1} , we can implement this choice of $T(x_i \rightarrow x_j)$ by the following steps:

1. Choose a trial position $x_{\text{trial}} = x_i + \delta_i$, where δ_i is a random number in the interval $[-\delta, \delta]$.
2. Calculate $w = p(x_{\text{trial}})/p(x_i)$.
3. If $w \geq 1$, accept the change and let $x_{i+1} = x_{\text{trial}}$.
4. If $w < 1$, generate a random number r .
5. If $r \leq w$, accept the change and let $x_{i+1} = x_{\text{trial}}$.
6. If the trial change is not accepted, then let $x_{i+1} = x_i$.

It is necessary to sample many points of the random walk before the asymptotic probability distribution $p(x)$ is attained. How do we choose the maximum “step size” δ ? If δ is too large, only a small percentage of trial steps will be accepted and the sampling of $p(x)$ will be inefficient. On the other hand, if δ is too small, a large percentage of trial steps will be accepted, but again the sampling of $p(x)$ will be inefficient. A rough criterion for the magnitude of δ is that approximately

one third to one half of the trial steps should be accepted. We also wish to choose the value of x_0 such that the distribution $\{x_i\}$ will approach the asymptotic distribution as quickly as possible. An obvious choice is to begin the random walk at a value of x at which $p(x)$ is a maximum.

Pseudocode that implements the Metropolis algorithm is given below.

```
double xtrial = x + (2*rnd.nextDouble() - 1.0)*delta;
double w = p(xtrial)/p(x);
if (rnd <= w)
{
    x = xtrial
    naccept++;           // number of acceptances
}
```

Problem 11.15. The Gaussian distribution

- Write a program using the Metropolis algorithm to generate the Gaussian distribution, $p(x) = Ae^{-x^2/2}$. Is the value of the normalization constant A relevant? Determine the qualitative dependence of the acceptance ratio and the equilibration time on the maximum step size δ . One possible criterion for equilibrium is that $\langle x^2 \rangle \approx 1$. What is a reasonable choice for δ ? How many trials are needed to reach equilibrium for your choice of δ ?
- Modify your program so that it plots the asymptotic probability distribution generated by the Metropolis algorithm.
- Calculate the autocorrelation function $C(j)$ defined by

$$C(j) = \frac{\langle x_{i+j} x_i \rangle - \langle x_i \rangle^2}{\langle x_i^2 \rangle - \langle x_i \rangle^2}, \quad (11.57)$$

where $\langle \dots \rangle$ indicates an average over the random walk. What is the value of $C(j=0)$? What would be the value of $C(j \neq 0)$ if x_i were completely random? Calculate $C(j)$ for different values of j and determine the value of j for which $C(j)$ is essentially zero.

Problem 11.16. Application of the Metropolis algorithm

- Although the Metropolis algorithm is not the most efficient method in this case, write a program to estimate the average

$$\langle x \rangle = \frac{\int_0^\infty x e^{-x} dx}{\int_0^\infty e^{-x} dx}, \quad (11.58)$$

with $p(x) = Ae^{-x}$ for $x \geq 0$ and $p(x) = 0$ for $x < 0$. Incorporate into the program a computation of the histogram $H(x)$ showing the fraction of points in the random walk in the region x to $x + \Delta x$, with $\Delta x = 0.2$. Begin with $n = 1000$ and maximum step size $\delta = 1$. Allow the system to equilibrate for 200 steps before computing averages. Is the integrand sampled uniformly? If not, what is the approximate region of x where the integrand is sampled more often?

- Calculate analytically the exact value of $\langle x \rangle$. How do your Monte Carlo results compare with the exact value for $n = 100$ and $n = 1000$ with $\delta = 0.1, 1$, and 10 ? Estimate the standard error of the mean. Does this error give a reasonable estimate of the error? If not, why?

- c. In part (b) you should have found that the estimated error is much smaller than the actual error. The reason is that the $\{x_i\}$ are not statistically independent. The Metropolis algorithm produces a random walk whose points are correlated with each other over short times (measured in the number of Monte Carlo steps). The correlation of the points decays exponentially with time. If τ is the characteristic time for this decay, then only points separated by approximately 2 to 3τ can be considered statistically independent. Rerun your program with the data grouped into 20 sets of 50 points each and 10 sets of 100 points each. If the sets of 50 points each are statistically independent (that is, if τ is significantly smaller than 50), then your estimate of the error for the two groupings should be approximately the same.

Appendix 11A: Error Estimates for Numerical Integration

We derive the dependence of the truncation error estimates on the number of intervals for the numerical integration methods considered in Sections 11.1 and 11.3. These estimates are based on the assumed adequacy of the Taylor series expansion of the integrand $f(x)$:

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2}f''(x_i)(x - x_i)^2 + \dots, \quad (11.59)$$

and the integration of (11.1) in the interval $x_i \leq x \leq x_{i+1}$:

$$\int_{x_i}^{x_{i+1}} f(x) dx = f(x_i)\Delta x + \frac{1}{2}f'(x_i)(\Delta x)^2 + \frac{1}{6}f''(x_i)(\Delta x)^3 + \dots \quad (11.60)$$

We first estimate the error associated with the rectangular approximation with $f(x)$ evaluated at the left side of each interval. The error Δ_i in the interval $[x_i, x_{i+1}]$ is the difference between (11.60) and the estimate $f(x_i)\Delta x$:

$$\Delta_i = \left[\int_{x_i}^{x_{i+1}} f(x) dx \right] - f(x_i)\Delta x \approx \frac{1}{2}f'(x_i)(\Delta x)^2. \quad (11.61)$$

We see that to leading order in Δx , the error in each interval is order $(\Delta x)^2$. Because there are a total of n intervals and $\Delta x = (b - a)/n$, the total error associated with the rectangular approximation is $n\Delta_i \sim n(\Delta x)^2 \sim n^{-1}$.

The estimated error associated with the trapezoidal approximation can be found in the same way. The error in the interval $[x_i, x_{i+1}]$ is the difference between the exact integral and the estimate, $\frac{1}{2}[f(x_i) + f(x_{i+1})]\Delta x$:

$$\Delta_i = \left[\int_{x_i}^{x_{i+1}} f(x) dx \right] - \frac{1}{2}[f(x_i) + f(x_{i+1})]\Delta x. \quad (11.62)$$

If we use (11.60) to estimate the integral and (11.59) to estimate $f(x_{i+1})$ in (11.62), we find that the term proportional to f' cancels and that the error associated with one interval is order $(\Delta x)^3$. Hence, the total error in the interval $[a, b]$ associated with the trapezoidal approximation is order n^{-2} .

Because Simpson's rule is based on fitting $f(x)$ in the interval $[x_{i-1}, x_{i+1}]$ to a parabola, error terms proportional to f'' cancel. We might expect that error terms of order $f'''(x_i)(\Delta x)^4$ contribute, but these terms cancel by virtue of their symmetry. Hence the $(\Delta x)^4$ term of the Taylor expansion of $f(x)$ is adequately represented by Simpson's rule. If we retain the $(\Delta x)^4$ term in the Taylor series of $f(x)$, we find that the error in the interval $[x_i, x_{i+1}]$ is of order $f''''(x_i)(\Delta x)^5$ and that the total error in the interval $[a, b]$ associated with Simpson's rule is $O(n^{-4})$.

The error estimates can be extended to two dimensions in a similar manner. The two-dimensional integral of $f(x, y)$ is the volume under the surface determined by $f(x, y)$. In the “rectangular” approximation, the integral is written as a sum of the volumes of parallelograms with cross sectional area $\Delta x \Delta y$ and a height determined by $f(x, y)$ at one corner. To determine the error, we expand $f(x, y)$ in a Taylor series

$$f(x, y) = f(x_i, y_i) + \frac{\partial f(x_i, y_i)}{\partial x}(x - x_i) + \frac{\partial f(x_i, y_i)}{\partial y}(y - y_i) + \dots, \quad (11.63)$$

and write the error as

$$\Delta_i = \left[\iint f(x, y) dx dy \right] - f(x_i, y_i) \Delta x \Delta y. \quad (11.64)$$

If we substitute (11.63) into (11.64) and integrate each term, we find that the term proportional to f cancels and the integral of $(x - x_i) dx$ yields $\frac{1}{2}(\Delta x)^2$. The integral of this term with respect to dy gives another factor of Δy . The integral of the term proportional to $(y - y_i)$ yields a similar contribution. Because Δy also is order Δx , the error associated with the intervals $[x_i, x_{i+1}]$ and $[y_i, y_{i+1}]$ is to leading order in Δx :

$$\Delta_i \approx \frac{1}{2}[f'_x(x_i, y_i) + f'_y(x_i, y_i)](\Delta x)^3. \quad (11.65)$$

We see that the error associated with one parallelogram is order $(\Delta x)^3$. Because there are n parallelograms, the total error is order $n(\Delta x)^3$. However in two dimensions, $n = A/(\Delta x)^2$, and hence the total error is order $n^{-1/2}$. In contrast, the total error in one dimension is order n^{-1} , as we saw earlier.

The corresponding error estimates for the two-dimensional generalizations of the trapezoidal approximation and Simpson's rule are order n^{-1} and n^{-2} respectively. In general, if the error goes as order n^{-a} in one dimension, then the error in d dimensions goes as $n^{-a/d}$. In contrast, Monte Carlo errors vary as order $n^{-1/2}$ independent of d . Hence for large enough d , Monte Carlo integration methods will lead to smaller errors for the same choice of n .

Appendix 11B: The Standard Deviation of the Mean

In Section 11.4 we gave empirical reasons for the claim that the error associated with a single measurement consisting of n trials equals σ/\sqrt{n} , where σ is the standard deviation in a single measurement. We now present an analytical derivation of this relation.

The quantity of experimental interest is denoted as x . Consider m sets of measurements each with n trials for a total of mn trials. We use the index α to denote a particular measurement

and the index i to designate the i th trial within a measurement. We denote $x_{\alpha,i}$ as trial i in the measurement α . The value of a measurement is given by

$$M_\alpha = \frac{1}{n} \sum_{i=1}^n x_{\alpha,i}. \quad (11.66)$$

The mean \overline{M} of the *total* mn individual trials is given by

$$\overline{M} = \frac{1}{m} \sum_{\alpha=1}^m M_\alpha = \frac{1}{nm} \sum_{\alpha=1}^m \sum_{i=1}^n x_{\alpha,i}. \quad (11.67)$$

The difference between measurement α and the mean of all the measurements is given by

$$e_\alpha = M_\alpha - \overline{M}. \quad (11.68)$$

We can write the variance of the means as

$$\sigma_m^2 = \frac{1}{m} \sum_{\alpha=1}^m e_\alpha^2. \quad (11.69)$$

We now wish to relate σ_m to the variance of the individual trials. The discrepancy $d_{\alpha,i}$ between an individual sample $x_{\alpha,i}$ and the mean is given by

$$d_{\alpha,i} = x_{\alpha,i} - \overline{M}. \quad (11.70)$$

Hence, the variance σ^2 of the nm individual trials is

$$\sigma^2 = \frac{1}{mn} \sum_{\alpha=1}^m \sum_{i=1}^n d_{\alpha,i}^2. \quad (11.71)$$

We write

$$e_\alpha = M_\alpha - \overline{M} = \frac{1}{n} \sum_{i=1}^n (x_{\alpha,i} - \overline{M}) \quad (11.72)$$

$$= \frac{1}{n} \sum_{i=1}^n d_{\alpha,i}. \quad (11.73)$$

If we substitute (11.73) into (11.69), we find

$$\sigma_m^2 = \frac{1}{m} \sum_{\alpha=1}^m \left(\frac{1}{n} \sum_{i=1}^n d_{\alpha,i} \right) \left(\frac{1}{n} \sum_{j=1}^n d_{\alpha,j} \right). \quad (11.74)$$

The sum in (11.74) over trials i and j in set α contains two kinds of terms—those with $i = j$ and those with $i \neq j$. We expect that $d_{\alpha,i}$ and $d_{\alpha,j}$ are independent and equally positive or negative on the average. Hence in the limit of a large number of measurements, we expect that only the terms with $i = j$ in (11.74) will survive, and we write

$$\sigma_m^2 = \frac{1}{mn^2} \sum_{\alpha=1}^m \sum_{i=1}^n d_{\alpha,i}^2. \quad (11.75)$$

If we combine (11.75) with (11.71), we arrive at the desired result

$$\sigma_m^2 = \frac{\sigma^2}{n}. \quad (11.76)$$

Appendix 11C: The Acceptance-Rejection Method

Although the inverse transform method discussed in Section 11.5 can in principle be used to generate any desired probability distribution, in practice the method is limited to functions for which the equation, $r = P(x)$, can be solved analytically for x or by simple numerical approximation. Another method for generating nonuniform probability distributions is the *acceptance-rejection* method due to von Neumann.

Suppose that $p(x)$ is a (normalized) probability density function that we wish to generate. For simplicity, we assume $p(x)$ is nonzero in the unit interval. Consider a positive definite *comparison function* $w(x)$ such that $w(x) > p(x)$ in the entire range of interest. A simple although not generally optimum choice of w is a constant greater than the maximum value of $p(x)$. Because the area under the curve $p(x)$ in the range x to $x + \Delta x$ is the probability of generating x in that range, we can follow a procedure similar to that used in the hit or miss method. Generate two numbers at random to define the location of a point in two dimensions which is distributed uniformly in the area under the comparison function $w(x)$. If this point is outside the area under $p(x)$, the point is rejected; if it lies inside the area, we accept it. This procedure implies that the accepted points are uniform in the area under the curve $p(x)$ and that their x values are distributed according to $p(x)$.

One procedure for generating a uniform random point (x, y) under the comparison function $w(x)$ is as follows.

1. Choose a form of $w(x)$. One choice would be to choose $w(x)$ such that the values of x distributed according to $w(x)$ can be generated by the inverse transform method. Let the total area under the curve $w(x)$ be equal to A .
2. Generate a uniform random number in the interval $[0, A]$ and use it to obtain a corresponding value of x distributed according to $w(x)$.
3. For the value of x generated in step (2), generate a uniform random number y in the interval $[0, w(x)]$. The point (x, y) is uniformly distributed in the area under the comparison function $w(x)$. If $y \leq p(x)$, then accept x as a random number distributed according to $p(x)$.

Repeat steps (2) and (3) many times.

Note that the acceptance-rejection method is efficient only if the comparison function $w(x)$ is close to $p(x)$ over the entire range of interest.

References and Suggestions for Further Reading

Forman S. Acton, *Numerical Methods That Work*, Harper & Row (1970); corrected edition, Mathematical Association of America (1990). A delightful book on numerical methods.

Isabel Beichl and Francis Sullivan, “The Importance of Importance Sampling,” *Computing in Science and Engineering* 1(#2), 71–73 (1999).

Steven E. Koonin and Dawn C. Meredith, *Computational Physics*, Addison-Wesley (1990). Chapter 8 covers much of the same material on Monte Carlo methods as discussed in this chapter.

Malvin H. Kalos and Paula A. Whitlock, *Monte Carlo Methods, Vol. 1: Basics*, John Wiley & Sons (1986). The authors are well known experts on Monte Carlo methods.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes*, second edition, Cambridge University Press (1992).

Reuven Y. Rubinstein, *Simulation and the Monte Carlo Method*, John Wiley & Sons (1981). An advanced, but clearly written treatment of Monte Carlo methods.

I. M. Sobol, *The Monte Carlo Method*, Mir Publishing (1975). A very readable short text with excellent sections on nonuniform probability densities and the neutron transport problem.

Chapter 12

Random Walks

©2001 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
24 April 2001

We explore applications of random walks to a variety of systems and the generation of random number sequences.

12.1 Introduction

We introduced the random walk problem in Section 7.2 in the context of the motion of drunken sailors and the one-dimensional motion of particles. Of course, random walks are not restricted to one dimension nor are the applications limited to the wanderings of inebrates. We already know that we can use random walk methods to estimate a definite integral (see Section 11.8). In this chapter we introduce some of the more popular random walk models and discuss several applications. In succeeding chapters we discuss other applications of random walk methods to problems that in many cases have no obvious connection to random walks.

We begin with the most elementary version of a random walk model, a walk on a one-dimensional translationally invariant lattice with steps occurring at uniform time intervals. In Section 7.2 we showed that the variance or dispersion

$$\langle \Delta x_N^2 \rangle = \langle x_N^2 \rangle - \langle x_N \rangle^2 \quad (12.1)$$

depends linearly on N , where N is the number of steps. Many applications of random walk models make use of asymptotic results for large N . For example, in many cases $\langle \Delta x_N^2 \rangle$ satisfies a power law for sufficiently large N , that is,

$$\langle \Delta x_N^2 \rangle \sim N^{2\nu}. \quad (N \gg 1) \quad (12.2)$$

In this context the symbol \sim is interpreted as “asymptotically equal to” and the relation (12.2) is an example of an asymptotic *scaling law*. For the simple one-dimensional random walk model we know that the relation (12.2) is valid for all N and that $\nu = \frac{1}{2}$. In many of the random walk

problems introduced in this chapter, we will determine if such a power law dependence exists for large N .

Although we introduced the simple one-dimensional random walk model in Section 7.2, we did not consider all its properties. Some additional properties of this model are considered in Problem 12.1.

Problem 12.1. Discrete time random walks in one dimension

- a. Suppose that the probability of moving to the right is $p = 0.7$. Compute $\langle x_N \rangle$ and $\langle x_N^2 \rangle$ for $N = 4, 8, 16$, and 32 . What is the interpretation of $\langle x_N \rangle$ in this case? What is the qualitative dependence of $\langle \Delta x_N^2 \rangle$ on N ? Does $\langle x_N^2 \rangle$ depend simply on N ?
- b. Use the error analysis discussed in Section 11.4 to estimate the number of trials needed to obtain $\langle \Delta x_N^2 \rangle$ to 1% accuracy for $N = 8$ and $N = 32$.
- c. An interesting property of random walks is the mean number $\langle D_N \rangle$ of *distinct* lattice sites visited during the course of an N step walk. Do a Monte Carlo simulation of $\langle D_N \rangle$ and determine its N dependence.

We can equally well consider either a large number of successive walks as in Problem 12.1 or a large number of similar (noninteracting) walkers moving at the same time. In Problem 12.2 we consider the motion of many random walkers moving independently of one another on a two-dimensional lattice.

Problem 12.2. A random walk in two dimensions

Consider a collection of walkers initially at the origin of a square lattice. At each unit of time, each of the walkers moves at random with equal probability in one of the four possible directions. The following program implements this algorithm and shows the sites that have been visited.

```
// 3/22/01, 11 am
package edu.clarku.sip.chapter12;
import edu.clarku.sip.plot.*;
import edu.clarku.sip.templates.*;
import edu.clarku.sip.graphics.*;
import java.awt.*;
import java.util.Random;

// random walk in two dimensions
public class RandomWalk implements AnimationModel, Runnable, Drawable
{
    private int nwalkers;           // number of walkers
    private int xpositions[];
    private int ypositions[];
    private Control myControl = new SAnimationControl(this);
    private Thread animationThread;
    private World2D world = new World2D();
    private Random random;
```

```
public RandomWalk()
{
    world.addDrawable(this);
    random = new Random();
}

public void startCalculation()
{
    nwalkers = (int) myControl.getValue("nwalkers");
    xpositions = new int[nwalkers];
    ypositions = new int[nwalkers];
    random.setSeed(7);
    animationThread = new Thread(this);
    animationThread.start();
}

public void run()
{
    while (animationThread != null)
    {
        step();
        try
        {
            Thread.sleep(100);
        }
        catch(InterruptedException e){}
    }
}

public void stopCalculation()
{
    animationThread = null;
}

public void step()
{
    move();
    world.render();
}

public void continueCalculation()
{
    animationThread = new Thread(this);
    animationThread.start();
}
```

```
public void clear(){}

public void reset()
{
    myControl.setValue("nwalkers", 200);
}

void move()
{
    for (int i = 0; i < nwalkers; i++)
        choice(i);
}

void choice(int i)
{
    double p = random.nextDouble();
    if (p <= 0.25)
        xpositions[i] = xpositions[i] + 1;
    else if (p <= 0.5)
        xpositions[i] = xpositions[i] - 1;
    else if (p <= 0.75)
        ypositions[i] = ypositions[i] - 1;
    else
        ypositions[i] = ypositions[i] + 1;
}

public void draw(World2D world, Graphics g)
{
    for (int i = 0; i < nwalkers; i++)
    {
        double x = xpositions[i];
        double y = ypositions[i];
        int size = 2;
        int px = world.xToPix(x - size/2);
        int py = world.yToPix(y - size/2);
        g.setColor(Color.red);
        g.fillRect(px, py, size, size);
    }
}

public static void main(String[] args)
{
    RandomWalk rw = new RandomWalk();
    rw.reset();
}
}
```

- Run Program RandomWalk with the number of walkers `nwalkers` ≥ 200 and the number of steps taken by each walker $N \geq 500$. If each walker represents a bee, describe the qualitative nature of the shape of the swarm of bees. Describe the qualitative nature of the surface of the swarm as a function of N . Is the surface jagged or smooth?
- Compute the quantities $\langle x_N \rangle$, $\langle y_N \rangle$, $\langle \Delta x_N^2 \rangle$, and $\langle \Delta y_N^2 \rangle$ as a function of N . The average is over the walkers. Also compute the net mean square displacement $\langle \Delta R_N^2 \rangle$ given by

$$\langle \Delta R_N^2 \rangle = \langle x_N^2 \rangle + \langle y_N^2 \rangle - \langle x_N \rangle^2 - \langle y_N \rangle^2. \quad (12.3)$$

What is the dependence of each quantity on N ?

- Enumerate all the random walks on a square lattice for $N = 4$ and obtain exact results for $\langle x_N \rangle$, $\langle y_N \rangle$ and $\langle \Delta R_N^2 \rangle$. Assume that all four directions are equally probable. Verify your program by comparing the Monte Carlo and exact enumeration results.
- Estimate $\langle \Delta R_N^2 \rangle$ for $N = 8, 16, 32$, and 64 using a reasonable number of trials for each value of N . Assume that $\langle \Delta R_N^2 \rangle$ has the asymptotic N dependence:

$$\langle \Delta R_N^2 \rangle \sim N^{2\nu}, \quad (N \gg 1) \quad (12.4)$$

and estimate the exponent ν from a log-log plot of $\langle \Delta R_N^2 \rangle$ versus N . If $\nu \approx \frac{1}{2}$, estimate the magnitude of the self-diffusion coefficient D given by

$$\langle R_N^2 \rangle \sim 2dDN. \quad (12.5)$$

The form (12.5) is similar to (8.39) with the time t in (8.39) replaced by the number of steps N .

- Estimate the quantities $\langle x_N \rangle$, $\langle y_N \rangle$, $\langle R_N^2 \rangle = \langle x_N^2 + y_N^2 \rangle$, and $\langle \Delta R_N^2 \rangle$ for the same values of N as in part (d), with the probabilities 0.4, 0.2, 0.2, 0.2, corresponding to a step to the right, left, up, and down, respectively. This choice of probabilities corresponds to a biased random walk with a drift to the right. What is the interpretation of $\langle x_N \rangle$ in this case? What is the dependence of $\langle \Delta R_N^2 \rangle$ on N ? Does $\langle R_N^2 \rangle$ depend simply on N ?
- Consider a random walk that starts at a site that is a distance $y = h$ above a horizontal line (see Figure 12.1). If the probability of a step down is greater than the probability of a step up, we expect that the walker will eventually reach a site on the horizontal line. This walk is a simple model of the fall of a rain drop in the presence of a random swirling breeze. Do a Monte Carlo simulation to determine the mean time τ for the walker to reach any site on the line $x = 0$ and find the functional dependence of τ on h . Is it possible to define a velocity in the vertical direction? Because the walker does not always move vertically, it suffers a net displacement Δx in the horizontal direction. How does $\langle \Delta x^2 \rangle$ depend on h and τ ? Reasonable values for the step probabilities are 0.1, 0.6, 0.15, 0.15, corresponding to up, down, right, and left, respectively.
- Do a Monte Carlo simulation of $\langle \Delta R_N^2 \rangle$ on the triangular lattice (see Figure 8.5) and estimate ν . Can you conclude that ν is independent of the symmetry of the lattice? Does D depend on the symmetry of the lattice? If so, give a qualitative explanation for this dependence.

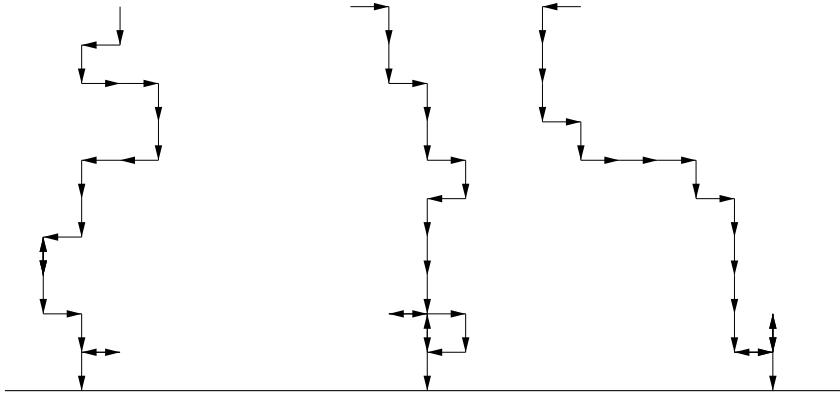


Figure 12.1: Examples of the random path of a raindrop to the ground. The step probabilities are given in Problem 12.2f.

12.2 Modified Random Walks

So far we have considered random walks on one- and two-dimensional lattices where the walker has no “memory” of the previous step. What happens if the walkers remember the nature of their previous steps? What happens if there are multiple random walkers, with the condition that no double occupancy is allowed? We explore these and other variations of the simple random walk in this section. All these variations have applications to physical systems.

Problem 12.3. A persistent random walk

- In a “persistent” random walk, the *transition* or “jump” probability depends on the previous transition. Consider a walk on a one-dimensional lattice, and suppose that step $N - 1$ has been made. Then step N is made in the same direction with probability α ; a step in the opposite direction occurs with probability $1 - \alpha$. Write a program to do a Monte Carlo simulation of the persistent random walk in one dimension. Compute $\langle x_N \rangle$, $\langle x_N^2 \rangle$, $\langle \Delta x_N^2 \rangle$, and $P_N(x)$. Note that it is necessary to specify both the initial position and an initial direction of the walker. What is the $\alpha = \frac{1}{2}$ limit of the persistent random walk?
- Consider the cases $\alpha = 0.25$ and $\alpha = 0.75$ and determine $\langle \Delta x_N^2 \rangle$ for $N = 8, 64, 256$, and 512 . Estimate the value of ν from a log-log plot of $\langle \Delta x_N^2 \rangle$ versus N for large N . Does ν depend on α ? If $\nu \approx \frac{1}{2}$, determine the self-diffusion coefficient D for $\alpha = 0.25$ and 0.75 . Give a physical argument why $D(\alpha \neq 0.5)$ is greater (smaller) than $D(\alpha = 0.5)$.
- A persistent random walk can be considered as an example of a *multistate* walk in which the state of the walk is defined by the last transition. In the above example, the walker is in one of two states; at each step the probabilities of remaining in the same state or switching states are α and $1 - \alpha$ respectively. One of the earliest applications of a two state random walk was to the study of diffusion in a chromatographic column. Suppose that a molecule in a chromatographic column can be either in a mobile phase (constant velocity v) or in a trapped

phase (zero velocity). Instead of each step changing the position by ± 1 , the position at each step changes by $+v$ or 0 . A quantity of experimental interest is the probability $P_N(x)$ that a molecule has traveled a distance x in N steps. Choose $v = 1$ and $\alpha = 0.75$ and compute the qualitative behavior of $P_N(x)$. Explain why the molecule cannot diffuse in either state, but that it is still possible to define an effective diffusion coefficient for the molecule.

- d. You might have expected that the persistent random walk yields a nonzero value for $\langle x_N \rangle$. Verify that $\langle x_N \rangle = 0$, and explain why this result is exact. How does the persistent random walk differ from the biased random walk for which $p \neq q$?

The fall of a raindrop considered in Problem 12.2f is an example of a *restricted* random walk, that is, a walk in the presence of a boundary. (Another example of a restricted random walk was considered in Problem 7.12c.) In the following problem, we discuss in a more general context the effects of various types of restrictions or boundaries on random walks. Another example of a restricted random walk is given in Problem 12.10.

Problem 12.4. Restricted random walks

- a. Consider a one-dimensional lattice with “trap” sites at $x = 0$ and $x = a$ ($a > 0$). A walker begins at site x_0 ($0 < x_0 < a$) and takes unit steps to the left and right with equal probability. When the walker arrives at a trap, it vanishes. Do a Monte Carlo simulation and verify that the mean number of steps τ for the particle to be trapped (*the first passage time*) is given by

$$\tau = (2D)^{-1}x_0(a - x_0). \quad (12.6)$$

D is the self-diffusion coefficient in the absence of the traps, and the average is over all possible walks.

- b. Random walk models in the presence of traps have had an important role in condensed matter science. For example, consider the following idealized model of energy transport in solids. The solid is represented as a lattice with two types of sites: hosts and traps. An incident photon is absorbed at a host site and excites the host molecule or atom. The excitation energy or *exciton* is transferred at random to one of the host’s nearest neighbors and the original excited molecule returns to its ground state. In this way the exciton wanders through the lattice until it reaches a trap site. The exciton is then trapped and a chemical reaction occurs.

A simple version of this energy transport model is given by a one-dimensional lattice with traps placed on a periodic sublattice. Because the traps are placed at regular intervals, we can replace the random walk on an infinite lattice by a random walk on a ring. Consider a ring of N host or nontrapping sites and one trap site. If a walker has an equal probability of starting from any host site and an equal probability of a step to each nearest neighbor site, what is the N dependence of the mean survival time τ (the mean number of steps taken before a trap site is reached)? Use the results of part (a) rather than doing another simulation.

- c. Consider a one-dimensional lattice with reflecting sites at $x = -a$ and $x = a$. For example, if a walker reaches the reflecting site at $x = a$, it is reflected at the next step to $x = a - 1$. At $t = 0$, the walker starts at $x = 0$ and steps with equal probability to nearest neighbor sites. Write a Monte Carlo program to determine $P_N(x)$, the probability that the walker is at site x after N steps. Compare the form of $P_N(x)$ with and without the presence of the reflecting “walls.” Can

you distinguish the two probability distributions if N is the order of a ? At what value of N can you first distinguish the two distributions?

Although all of the above problems involved random walks on a lattice, it was not necessary to store the positions of the lattice sites or the path of the walker. In the following problem, we consider a random walk model that requires us to store the positions of a system of random walkers for which double occupancy is excluded, that is, only one walker can occupy a site. The physical motivation of this model arises from solid state physics where the diffusing particles are thermal vacancies whose density depends on the temperature. The main physical quantity of interest is the

**Problem 12.5.* Diffusion of interacting particles

Consider a square lattice with a nonzero density ρ of particles. Each particle moves at random to *empty* nearest neighbor sites. Double occupancy of sites is excluded; otherwise the particles are noninteracting. The main physical quantity of interest is the self-diffusion coefficient D of an individual particle. The model can be summarized by the following algorithm:

- i. Occupy at random the $L \times L$ sites of a square lattice with N particles subject to the conditions that no double occupancy is allowed, and the density $\rho = N/L^2$ has the desired value. (Remember that $\rho < 1$.) Tag each particle, that is, distinguish it from the others, and record its initial position in an array.
- ii. At each step choose a particle and one of its nearest neighbor sites at random. If the neighbor site is empty, the particle is moved to this site; otherwise the particle remains in its present position. The measure of “time” in this context is arbitrary. The usual definition is that during one unit of time or one *Monte Carlo step per particle*, each particle attempts one jump *on the average*. That is, the time is advanced by $1/N$ each time a particle is chosen even if the particle does not move.

The diffusion coefficient D is obtained as the limit $t \rightarrow \infty$ of $D(t)$, where $D(t)$ is given by

$$D(t) = \frac{1}{2dt} \langle \Delta R(t)^2 \rangle, \quad (12.7)$$

and $\langle \Delta R(t)^2 \rangle$ is the net mean square displacement per tagged particle after t units of time. An example of a program that implements this algorithm is given in the following.

```
// 3/21/01 9:40 pm
package edu.clarku.sip.chapter12;
import edu.clarku.sip.templates.*;
import edu.clarku.sip.graphics.*;
import edu.clarku.sip.plot.*;
import java.awt.*;
import java.util.Random;

// simulation of particle diffusion in a lattice gas
public class LatticeGas implements AnimationModel, Lattice, Runnable
{
```

```
private int L;      // linear dimension of lattice
private double Lhalf;
private int N;      // number of particles
private final int OCCUPIED = 1;
private final int EMPTY = 0;
private Thread animationThread;
private Color occupiedColor = Color.red;
private Color emptyColor = Color.blue;
private LatticeWorld latticeWorld = new LatticeWorld();
private Control myControl = new SAnimationControl(this);
private int site[][];
private int x[]; // keeps track of the occupied sites
private int y[]; // keeps track of the occupied sites
private int x0[]; // keeps track of the occupied sites
private int y0[];
private Random random;
private int xposition;
private int yposition;
private Plot plot = new Plot("time", "R2Bar", "Lattice Gas");
private double R2bar;
private double t;

public LatticeGas()
{
    plot.setAutoUpdate(true);
}

public int getNumberOfSites()
{
    return L;
}

public void startCalculation()
{
    L = (int) myControl.getValue("L");
    N = (int) myControl.getValue("N");
    site = new int[L][L];
    x = new int[N + 1];
    y = new int[N + 1];
    x0 = new int[N + 1];
    y0 = new int[N + 1];
    latticeWorld.setLattice(this);
    Lhalf = 0.5*L;
    t = 0;
    random = new Random(13907);
    fillLattice();
}
```

```
animationThread = new Thread(this);
animationThread.start();
}

public void continueCalculation(){}
public void clear(){}

public void reset()
{
    myControl.setValue("L",40);
    myControl.setValue("N",500);
}

public void stopCalculation()
{
    animationThread = null;
}

public void run()
{
    while(animationThread != null)
    {
        step();
        try
        {
            Thread.sleep(100);
        }
        catch(InterruptedException e){}
    }
}

public void step()
{
    move();
    computeR2Bar();
    t++;
    plot.addPoint(0, t, R2bar);
    latticeWorld.render();
}

// fill the lattice with N particles
void fillLattice()
{
    int i = 0;
    while (i != N)
    {
```

```

        int xadd = (int)(L*random.nextDouble());
        int yadd = (int)(L*random.nextDouble());
        if (site[xadd][yadd] == EMPTY)
        {
            i = i + 1;           // number of particles added
            site[xadd][yadd] = OCCUPIED;    // site occupied
            x[i] = xadd;         // x[0] is ignored
            y[i] = yadd;
            x0[i] = x[i];       // x-coordinate at t = 0
            y0[i] = y[i];
        }
    }
}

void move()
{
    for (int particle = 1; particle <= N; particle++)
    {
        int i = (int)(N*random.nextDouble());
        xposition = x[i];
        yposition = y[i];
        chooseDirection(); // move left, right, up, or down
        if (site[xposition][yposition] == EMPTY)      // if new site is unoccupied
        {
            site[x[i]][y[i]] = EMPTY; // place where it was is now empty
            x[i] = xposition; // assigns new positions
            y[i] = yposition;
            site[x[i]][y[i]] = OCCUPIED; // new site occupied
        }
    }
}

// choose random direction and use periodic boundary conditions
void chooseDirection()
{
    int dir = (int)(4*random.nextDouble()) + 1;
    switch(dir)
    {
        case 1:
            xposition = xposition + 1;
            if (xposition >= L)
                xposition = 0;
            break;
        case 2:
            xposition = xposition - 1;
            if (xposition < 0)

```

```

        xposition = L - 1;
        break;
    case 3:
        yposition = yposition + 1;
        if (yposition >= L)
            yposition = 0;
        break;
    case 4:
        yposition = yposition - 1;
        if (yposition < 0)
            yposition = L - 1;
        break;
    }
}

void computeR2Bar()
{
    double R2 = 0;
    for (int i = 1; i <= N; i++)
    {
        double dx = x[i] - x0[i];
        double dy = y[i] - y0[i];
        // use periodic boundary conditions to determine separation
        if (Math.abs(dx) > Lhalf)
            dx = dx - sgn(dx)*L;
        if (Math.abs(dy) > Lhalf)
            dy = dy - sgn(dy)*L;
        R2 = R2 + dx*dx + dy*dy;
    }
    R2bar = R2/N;
}

public static int sgn(double d)
{
    if (d < 0)
        return -1;
    else if (d > 0)
        return 1;
    else
        return 0;
}

public void draw(LatticeWorld lw, Graphics g)
{
    for (int i = 0; i < site.length; i++)
        for (int j = 0; j < site.length; j++)

```

```
        drawSite(lw, g, i, j);
    }

public void drawSite(LatticeWorld lw, Graphics g, int x, int y)
{
    if (site[x][y] == OCCUPIED)
        g.setColor(occupiedColor);
    else
        g.setColor(emptyColor);
    int px = lw.xToPix(x);
    int py = lw.yToPix(y);
    int size = lw.getCellSize();
    g.fillRect(px, py, size, size);
}

public static void main(String[] args)
{
    LatticeGas lg = new LatticeGas();
    lg.reset();
}
}
```

- Do a Monte Carlo simulation to determine D on a square lattice for $\rho = 0.1, 0.2, 0.3, 0.5$, and 0.7 . Choose $L \geq 40$. Although D is defined as the limit $t \rightarrow \infty$ of (12.7), $D(t)$ for this model fluctuates after a short equilibration time and no improvement in accuracy is achieved by increasing t . Better statistics for D can be obtained by averaging D over as many particles as possible and hence by considering a lattice with L as large as possible. The accuracy of D also can be increased by averaging $\langle R(t)^2 \rangle$ over different initial starting times. Why is it necessary to limit the number of Monte Carlo steps so that $\langle R(t)^2 \rangle$ is less than $(L/2)^2$? Verify that deviations of $D(t)$ from its mean value are proportional to the inverse square root of the total number of particles that enter into the average in (12.7).
 - Why is D a monotonically decreasing function of the density ρ ? To gain some insight into this dependence, determine the dependence on ρ of the probability that if a particle jumps to a vacancy at time t , it returns to its previous position at time $t + 1$. Is there a qualitative relation between the density dependence of D and this probability?
 - Consider a one-dimensional lattice model for which particles move at random, but double occupancy of sites is excluded. This restriction implies that particles cannot pass by each other. Compute $\langle \Delta x^2 \rangle$ as a function of t . Do the particles diffuse, that is, is $\langle \Delta x^2 \rangle$ proportional to t ? If not, what is the t dependence of $\langle \Delta x^2 \rangle$?

Problem 12.6. Random walk on a continuum

One of the first continuum models of a random walk was proposed by Rayleigh in 1919. The model is known as the freely jointed chain in polymer physics. In this model the length a of each step is a random variable with probability density $p(a)$, and the direction of each step is uniformly random.

For simplicity, we first consider a walker in two dimensions with $p(a)$ chosen so that each step has unit length. At each step the walker takes a step of unit length at a random angle. Write a Monte Carlo program to compute $P_N(r) \Delta r$, the probability that the displacement of the walker is in the range r to $r + \Delta r$ after N steps, where r is the distance from the origin. Verify that for sufficiently large N , the probability density $P_N(r)$ can be approximated by a Gaussian. Is a Gaussian a good approximation for small N ? Is it necessary to do a Monte Carlo simulation to confirm that $\langle R_N^2 \rangle \sim N$, or can you give a simple argument for this dependence based on the form of $P_N(r)$?

Problem 12.7. Random walks with steps of variable length

- Consider a random walk in one dimension with jumps of all lengths allowed. The probability density that the length of a single step is a is denoted by $p(a)$. If the form of $p(a)$ is given by $p(a) = e^{-a}$, what is the form of $P_N(x)$? Suggestions: Use the inverse transform method discussed in Section 11.5 to generate step lengths according to the probability density $p(a)$. Consider a walk of N steps and determine the net displacement x . Generate many such walks and determine $P_N(x)$. Plot $P_N(x)$ versus x and confirm that the form of $P_N(x)$ is consistent with a Gaussian distribution. Is this random walk equivalent to a diffusion process for sufficiently large N ?
- Assume that the probability density $p(a)$ is given by $p(a) = C/a^2$ for $a \geq 1$. Determine the normalization constant C using the condition $C \int_1^\infty a^{-2} da = 1$. Does the second moment of $p(a)$ exist? Do a Monte Carlo simulation as in part (a) and verify that the form of $P_N(x)$ is given by

$$P_N(x) \sim \frac{bN}{x^2 + b^2 N^2}, \quad (12.8)$$

What is the magnitude of the constant b ? Does the variance $\langle x^2 \rangle - \langle x \rangle^2$ of $P_N(x)$ exist? Is this random walk equivalent to a diffusion process?

Problem 12.8. The central limit theorem

Consider a continuous random variable x with probability density $f(x)$. That is, $f(x)\Delta x$ is the probability that x has a value between x and $x + \Delta x$. The m th moment of $f(x)$ is defined as

$$\langle x^m \rangle = \int x^m f(x) dx. \quad (12.9)$$

The mean value $\langle x \rangle$ is given by (12.9) with $m = 1$. The variance σ_x^2 of $f(x)$ is defined as

$$\sigma_x^2 = \langle x^2 \rangle - \langle x \rangle^2. \quad (12.10)$$

Consider the sum y_n corresponding to the average of n values of x :

$$y_n = \frac{1}{n}(x_1 + x_2 + \dots + x_n). \quad (12.11)$$

We adopt the notation $y = y_n$. Suppose that we make many measurements of y . We know that the values of y are not identical, but are distributed according to a probability density $P(y)$, where $P(y)\Delta y$ is the probability that the measured value of y is in the range y to $y + \Delta y$. The main quantities of interest are the mean $\langle y \rangle$, the variance $\sigma_y^2 = \langle y^2 \rangle - \langle y \rangle^2$, and $P(y)$ itself.

- a. Suppose that $f(x)$ is uniform in the interval $[-1, 1]$. Calculate $\langle x \rangle$ and σ_x analytically. Use a Monte Carlo method to make a sufficient number of measurements of y to determine $P(y)$, $\langle y \rangle$, and σ_y with reasonable accuracy. For example, choose $n = 1000$ and make 100 measurements of y . Verify that σ_y is approximately equal to σ_x/\sqrt{n} . Plot $P(y)$ versus y and discuss its qualitative form. Does the form of $P(y)$ change significantly if n is increased? Does the form of $P(y)$ change if the number of measurements of y is increased?
- b. To test the generality of the results of part (a), consider the exponential probability density

$$f(x) = \begin{cases} e^{-x}, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}. \quad (12.12)$$

Calculate $\langle x \rangle$ and σ_x analytically. Modify your Monte Carlo program and estimate $\langle y \rangle$, σ_y , and $P(y)$. Is σ_y related to σ_x as in part (a)? Plot $P(y)$ and discuss its qualitative form and its dependence on n and on the number of measurements of y .

- c. Let y be the Monte Carlo estimate of the integral (see Problem 11.3a)

$$4 \int_0^1 dx \sqrt{1 - x^2}. \quad (12.13)$$

In this case y is found by sampling the integrand $f(x) = 4\sqrt{1 - x^2}$ n times. Choose $n \geq 1000$ and make at least 100 measurements of y . Show that the values of y are distributed according to a Gaussian distribution. How is the variance of $P(y)$ related to the variance of $f(x)$?

- d. Consider the Lorentzian probability density

$$f(x) = \frac{1}{\pi} \frac{1}{x^2 + 1}. \quad (12.14)$$

Calculate the mean value $\langle x \rangle$. Does the second moment and hence the variance of $f(x)$ exist? Do a Monte Carlo calculation of $\langle y \rangle$, σ_y , and $P(y)$. Plot $P(y)$ as a function of y and discuss its qualitative form. What is the dependence of $P(y)$ on the number of trials?

Problem 12.8 illustrates the *central limit theorem* which states that the probability distribution of a variable y is a Gaussian centered at $\langle y \rangle$ with a standard deviation $1/\sqrt{n}$ times the standard deviation of $f(x)$. The requirements are that $f(x)$ has finite first and second moments, that the measurements of y are statistically independent, and that n is large. Use the central limit theorem to explain your results in Problem 12.8 and in Problem 12.7a. What is the relation of the central limit theorem to the calculations of the probability distribution in the random walk models that we already have considered?

Problem 12.9. Generation of the Gaussian distribution

Consider the sum

$$y = \sum_{i=1}^{12} r_i, \quad (12.15)$$

where r_i is a uniform random number in the unit interval. Make many “measurements” of y and show that the probability distribution of y approximates the Gaussian distribution with mean value 6 and variance 1. Discuss how to use this result to generate a Gaussian distribution with arbitrary mean and variance. This way of generating a Gaussian distribution is particularly useful when a “quick and dirty” approximation is appropriate.

Many of the problems we have considered have revealed the slow convergence of Monte Carlo simulations and the difficulty of obtaining quantitative results for asymptotic quantities. We conclude this section with a cautionary note and consider a “simple” problem for which straightforward Monte Carlo methods give misleading asymptotic results.

**Problem 12.10.* Random walk on lattices containing random traps

- a. We have considered the mean survival time of a one-dimensional random walker in the presence of a periodic distribution of traps (see Problem 12.4b). Now suppose that the trap sites are distributed at *random* on a one-dimensional lattice with density ρ . If a walker is placed at random at any nontrapping site, determine its mean survival time τ , the mean number of steps before a trap site is reached. Assume that the walker has an equal probability of moving to nearest neighbor sites at each step and use periodic boundary conditions.

This problem is more difficult than it might first appear, and there are a number of pitfalls. The major complication is that it is necessary to perform *three* averages: the distribution of traps, the origin of the walker, and the different walks for a given trap distribution and origin. Choose reasonable values for the number of trials associated with each average and do a Monte Carlo simulation to estimate the mean survival time τ . If τ exhibits a power law dependence on ρ , for example, $\tau \approx \tau_0 \rho^{-z}$, estimate the exponent z .

- b. A seemingly straightforward extension of part (a) is to estimate the probability of survival S_N of an N step random walk. Choose $\rho = 0.5$ and do a Monte Carlo simulation of S_N for N as large as possible. (Published results are for $N = 2 \times 10^3$ on lattices with $L = 50\,000$ sites and 50 000 trials.) Assume that the asymptotic form of S_N is given by

$$S_N \sim e^{-bN^\alpha}, \quad (12.16)$$

where b is a constant that depends on ρ . Are your results consistent with this form? Is it possible to make a meaningful estimate of the exponent α ?

- c. The object of part (b) is to convince you that it is not possible to use Monte Carlo methods directly to obtain the correct asymptotic behavior of S_N . The difficulty is that we are trying to estimate S_N in the asymptotic region where S_N is very small, and the small number of samples in this region prevent us from obtaining meaningful results. It has been proved using analytical methods, that the asymptotic N dependence of S_N has the form (12.16), but with $\alpha = 1/3$. Are your Monte Carlo results consistent with this value of α ?
- d. A method that reduces the number of required averages and hence reduces the fluctuations is to determine exactly, for a given distribution of trap sites, the probability that the walker is at site i after N steps. The method is illustrated in Figure 12.2. The first line represents a given configuration of traps distributed randomly on a one-dimensional lattice. One walker is placed at each regular site; trap sites are assigned the value 0. Because each walker moves with

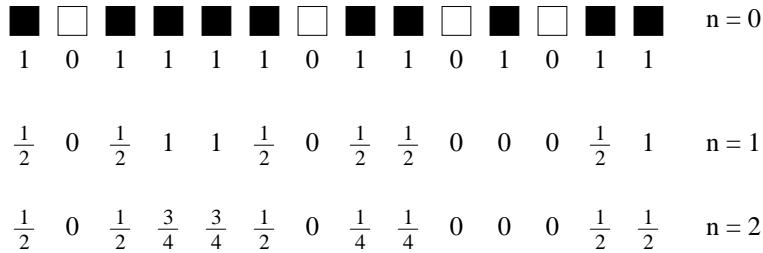


Figure 12.2: Example of the exact enumeration of walks on a given configuration of traps. The filled and empty squares denote regular and trap sites respectively. At step $N = 0$, a walker is placed at each regular site. The numbers at each site i represent the number of walkers W_i . Periodic boundary conditions are used. The initial number of walkers in this example is $m_0 = 10$. The mean survival probability at step $N = 1$ and $N = 2$ is found to be 0.6 and 0.475 respectively.

probability $\frac{1}{2}$ to each neighbor, the number of walkers $W_i(N+1)$ on site i at step $N+1$ is given by

$$W_i(N+1) = \frac{1}{2}[W_{i+1}(N) + W_{i-1}(N)]. \quad (12.17)$$

(Compare the relation (12.17) to (12.31) and to the relation that you found in Problem d.) The survival probability S_N after N steps for a given configuration of traps is given exactly by

$$S_N = \frac{1}{m_0} \sum_i W_i(N), \quad (12.18)$$

where m_0 is the initial number of walkers and the sum is over all sites in the lattice. Explain the relation (12.18), and write a program that computes S_N using (12.18) and (12.17). Then obtain the $\langle S_N \rangle$ by averaging over several configurations of traps. Choose $\rho = 0.5$ and determine S_N for $N = 32, 64, 128, 512$, and 1024 . Choose periodic boundary conditions and as large a lattice as possible. How well can you estimate the exponent α ? For comparison, Havlin et al. consider a lattice of $L = 50\,000$ and values of N up to 10^7 .

12.3 Applications to Polymers

Random walk models play an important role in polymer physics (cf. de Gennes). A polymer consists of N repeat units (monomers) with N very large ($N \sim 10^3 - 10^5$). For example, polyethylene can be represented as $\cdots -CH_2 -CH_2 -CH_2 - \cdots$. The detailed structure of the polymer is important for many practical applications. For example, if we wish to improve the fabrication of rubber, a good understanding of the local motions of the monomers in the rubber chain is essential. However, if we are interested in the *global* properties of the polymer, the details of the chain structure can be ignored.

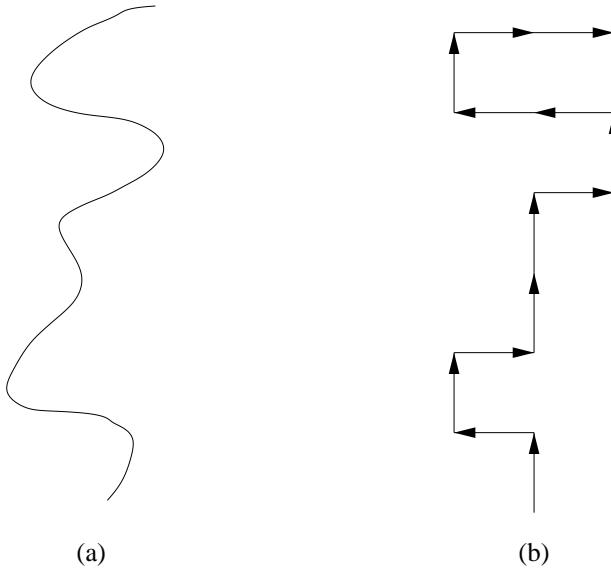


Figure 12.3: (a) Schematic illustration of a linear polymer in a good solvent. (b) Example of the corresponding self-avoiding walk on a square lattice.

Let us consider a familiar example of a polymer chain in a good solvent: a noodle in warm water. A short time after we place a noodle in warm water, the noodle becomes flexible, and it neither collapses into a little ball or becomes fully stretched. Instead, it adopts a random structure as shown schematically in Figure 12.3. If we do not add too many noodles, we can say that the noodles behave as a dilute solution of polymer chains in a good solvent. The dilute nature of the solution implies that we can ignore entanglement effects of the noodles and consider each noodle individually. The presence of a good solvent implies that the polymers can move freely and adopt many different configurations.

A fundamental geometrical property that characterizes a polymer in a good solvent is the mean square end-to-end distance $\langle R_N^2 \rangle$, where N is the number of monomers. It is known that for a dilute solution of polymer chains in a good solvent, the asymptotic dependence of $\langle R_N^2 \rangle$ is given by (12.4) with the exponent $\nu \approx 0.592$ in three dimensions. The result for ν in two dimensions is known to be exactly $\nu = 3/4$ for the model of polymers that we will discuss. The proportionality constant in (12.4) depends on the structure of the monomers and on the solvent. In contrast, the exponent ν is independent of these details.

We now discuss a random walk model that incorporates the global features of linear polymers in solution. We already have introduced a model of a polymer chain consisting of straight line segments of the same size joined together at random angles (see Problem 12.6). A further idealization is to place the polymer chain on a lattice (see Figure 12.3b). If we ignore the interactions of the monomers, this simple random walk model would yield $\nu = \frac{1}{2}$, independent of the dimension and symmetry of the lattice. Because this result for ν does not agree with experiment, we know that we are overlooking an important physical feature of polymers.

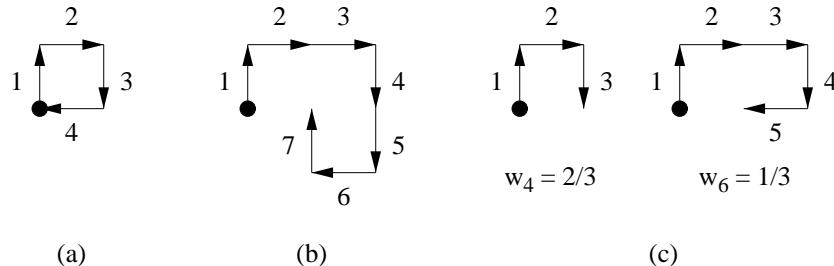


Figure 12.4: Examples of self-avoiding walks on the square lattice. The origin is denoted by a filled circle. (a) A $N = 3$ walk. The fourth step shown is forbidden. (b) A $N = 7$ walk that leads to a self-intersection at the next step; the weight of the $N = 8$ walk is zero. (c) Examples of the weights of walks in the enrichment method.

A more realistic model of linear polymers accounts for its most important physical feature, that is, two monomers cannot occupy the same spatial position. This constraint, known as the *excluded volume* condition, implies that the walk cannot be adequately described by a purely random walk. A well known lattice model for a flexible polymer chain that incorporates this constraint is known as the *self-avoiding walk* (SAW). Consider the set of all N step walks starting from the origin subject to the global constraint that no lattice site can be visited more than once in each walk; this constraint accounts for the excluded volume condition. Self-avoiding walks have many applications in the sciences, such as the physics of magnetic materials and the study of phase transitions, and they are of interest as purely mathematical objects. Many of the obvious questions about them have resisted rigorous analysis, and exact enumeration and Monte Carlo simulation have played an important role in our current understanding. We consider Monte Carlo simulations of the self-avoiding walk in two dimensions in Problems 12.11 and 12.12. Another algorithm for the self-avoiding walk is considered in Project 12.20.

Problem 12.11. The two-dimensional self-avoiding walk

- a. Consider the self-avoiding walk on the square lattice. Choose an arbitrary site as the origin and assume that the first step is “up.” The walks generated by the three other possible initial steps only differ by a rotation of the whole lattice and do not have to be considered explicitly. The second step can be in three possible directions because of the constraint that the walk cannot return to the origin. To obtain unbiased results, we generate a random number to choose one of the three directions. Successive steps are generated in the same way. Unfortunately, the walk will not usually continue indefinitely. The difficulty is that to obtain unbiased results, we must generate a random number (for example, 1, 2, or 3) as usual, even though one or more of the steps might lead to a self-intersection. If the next step does lead to a self-intersection, the walk must be terminated to keep the statistics unbiased. An example of a self-intersection for $N = 3$ is shown in Figure 12.4a. The next step leads to a self-intersection and violates the constraint for $N = 3$. In this case a new walk is started again at the origin.

Write a program that implements this algorithm and record the fraction $f(N)$ of successful attempts of constructing polymer chains with N total monomers. It is convenient to represent

the lattice as a two-dimensional array so that you can record the sites that already have been visited. What is the qualitative dependence of $f(N)$ on N ? What is the maximum value of N that you can reasonably consider? Determine the mean square end-to-end distance $\langle R_N^2 \rangle$ for these values of N .

- b. The disadvantage of the straightforward sampling method in part (a) is that it becomes very inefficient for long chains, that is, the fraction of successful attempts decreases exponentially fast. To overcome this attrition, several “enrichment” techniques have been developed. We first discuss a relatively simple procedure proposed by Rosenbluth and Rosenbluth in which each walk of N steps is associated with a weighting function $W(N)$. Because the first step to the north is always possible, we have $W(1) = 1$. In order that all allowed configurations of a given N are counted equally, the weights $W(N)$ for $N > 1$ are determined according to the following possibilities:
 - (a) All three possible steps violate the self-intersection constraint (see Figure 12.4b). The walk is terminated with a weight $W(N) = 0$, and a new walk is generated at the origin.
 - (b) All three steps are possible and $W(N) = W(N - 1)$.
 - (c) Only m steps are possible with $1 \leq m < 3$ (see Figure 12.4c). In this case $W(N) = (m/3)W(N - 1)$, and a random number is generated to choose one of the m possible steps.

The correct (unbiased) value of $\langle R_N^2 \rangle$ is obtained by weighting $R_{N,i}^2$, the value of R_N^2 found in the i th trial, by the value of $W_i(N)$, the weight found for the particular walk. Hence we write

$$\langle R_N^2 \rangle = \frac{\sum_i W_i(N) R_{N,i}^2}{\sum_i W_i(N)}, \quad (12.19)$$

where the sum is over all trials. Incorporate the Rosenbluth method into your Monte Carlo program, and calculate $\langle R_N^2 \rangle$ for $N = 4, 8, 16$, and 32 . Estimate the exponent ν from a log-log plot of $\langle R_N^2 \rangle$ versus N . Can you distinguish your estimate for ν from its random walk value $\nu = \frac{1}{2}$?

**Problem 12.12.* The reptation method

One of the more efficient enrichment algorithms is the “reptation” method (see Wall and Mandel). For simplicity, consider a model polymer chain in which all bond angles are $\pm 90^\circ$. As an example of this model, the five independent $N = 5$ polymer chains are shown in Figure 12.5. (Other chains differ only by a rotation or a reflection.) The reptation method can be stated as follows:

- i. Choose a chain at random and remove the tail link.
- ii. Attempt to add a link to the head of the chain. There is a maximum of two directions in which the new head link can be added.
- iii. If the attempt violates the self-intersection constraint, return to the original chain and interchange the head and tail. Include the chain in the statistical sample.

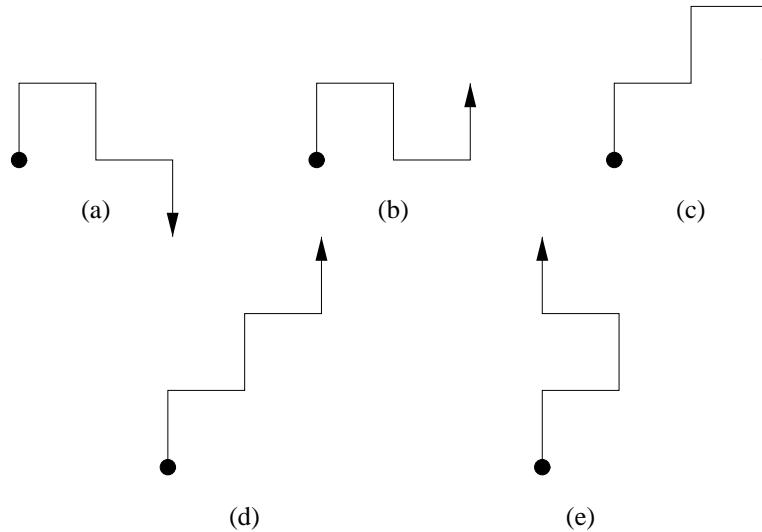


Figure 12.5: The five independent possible walks of $N = 5$ steps on a square lattice with $\pm 90^\circ$ bond angles. The tail and head of each walk are denoted by a circle and arrow respectively.

The above steps are repeated many times to obtain a statistical average of $\langle R_N^2 \rangle$.

As an example of the reptation method, suppose we choose chain a of Figure 12.5. A new link can be added in two directions (see Figure 12.6a), so that on the average we find, $a \rightarrow \frac{1}{2}c + \frac{1}{2}d$. In contrast, a link can be added to chain b in only one direction, and we obtain $b \rightarrow \frac{1}{2}e + \frac{1}{2}b$, where the tail and head of chain b have been interchanged (see Figure 12.6b). Confirm that $c \rightarrow \frac{1}{2}e + \frac{1}{2}a$, $d \rightarrow \frac{1}{2}c + \frac{1}{2}d$, and $e \rightarrow \frac{1}{2}a + \frac{1}{2}b$, and that all five chains are equally probable. That is, the transformations in the reptation method preserve the proper statistical weights of the chains without attrition. There is just one problem: unless we begin with a double ended “cul-de-sac” configuration such as shown in Figure 12.7, we will never obtain such a configuration using the above transformation. Hence, the reptation method introduces a small statistical bias, and the calculated mean end-to-end distance will be slightly larger than if all configurations were considered. However, the probability of such trapped configurations is very small, and the bias can be neglected for most purposes.

Adopt the $\pm 90^\circ$ bond angle restriction and calculate by hand the exact value of $\langle R_N^2 \rangle$ for $N = 5$. Then write a Monte Carlo program that implements the reptation method. Generate one walk of $N = 5$ and use the reptation method to generate a statistical sample of chains. As a check on your Monte Carlo program, compute $\langle R_N^2 \rangle$ for $N = 5$ and compare your result with the exact result. Then extend your Monte Carlo computations of $\langle R_N^2 \rangle$ to larger N . Modify the reptation model so that the bond angle also can be 180° . This modification leads to a maximum of three directions for a new bond. Compare the results of the two models.

Problem 12.13. The dynamics of polymers in a dilute solution

In principle, the dynamics of a polymer chain undergoing collisions with solvent molecules can

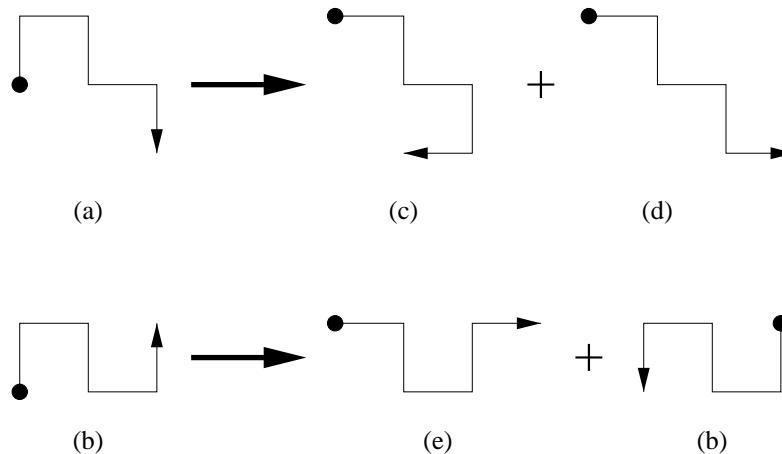


Figure 12.6: The possible transformations of chains *a* and *b*. One of the two possible transformations of chain *b* violates the self-intersection restriction and the head and tail are interchanged.

be simulated by using a molecular dynamics method. However, in practice only relatively small chains can be simulated in this way. An alternative approach is to use a Monte Carlo model that simplifies the effect of the random collisions of the solvent molecules with the atoms of the chain. Most of these models (cf. Verdier and Stockmayer) consider the chain to be composed of beads connected by bonds and restrict the motion of the beads to a lattice. For simplicity, we assume that the bond angles can be either $\pm 90^\circ$ or 180° . Begin with an allowed configuration of N beads ($N + 1$ bonds). A possible starting configuration can be generated by taking successive steps in the positive y direction and positive x directions. The dynamics of the model is summarized by the following algorithm.

1. Select at random a bead (occupied site) on the polymer chain. If the bead is not an end site, then the bead can move to a nearest neighbor site of another bead if this site is empty and if the new angle between adjacent bonds is either $\pm 90^\circ$ or 180° . For example, bead 4 in Figure 12.8 can move to position $4'$ while bead 3 cannot move if selected. That is, a selected bead can move to a diagonally opposite unoccupied site only if the two bonds to which it is attached are mutually perpendicular.
2. If the selected bead is an end site, move it to one of two (maximum) possible unoccupied sites so that the bond to which it is connected changes its orientation by $\pm 90^\circ$ (see Figure 12.8).
3. If the selected bead cannot move, retain the previous configuration.

The physical quantities of interest include the mean square end-to-end distance $\langle R_N^2 \rangle$ and the mean square displacement of the center of mass of the chain $\langle \Delta R_{\text{cm}}^2(N) \rangle$. The unit of time is the number of Monte Carlo steps per bead during which all beads have one chance on the average to move to a different site.

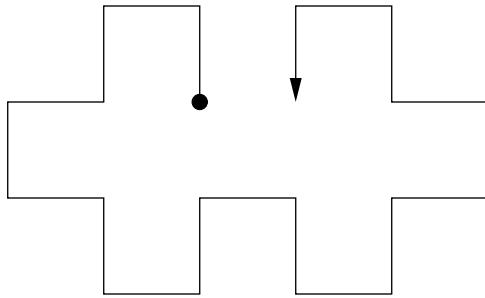


Figure 12.7: Example of a double cul-de-sac configuration for the self-avoiding walk that cannot be obtained by the reptation method.

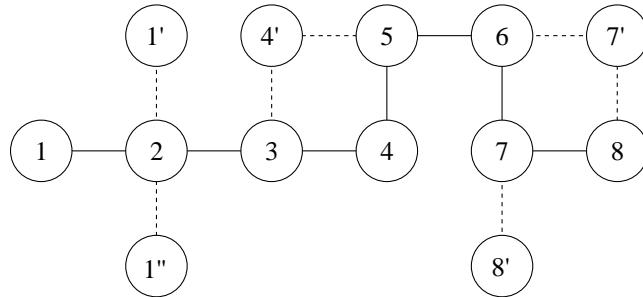


Figure 12.8: Examples of possible moves of the simple polymer dynamics model considered in Problem 12.13. For this configuration beads 2, 3, 5, and 6 cannot move, while beads 1, 4, 7, and 8 can move to the positions shown if they are selected. Only one bead can move at a time. This figure is adopted from the article by Verdier and Stockmayer.

- Consider a two-dimensional lattice and compute $\langle R_N^2 \rangle$ and $\langle \Delta R_{\text{cm}}^2(N) \rangle$ for various values of N . How do these quantities depend on N ? (The first published results for three dimensions were limited to 32 Monte Carlo steps per bead for $N = 8, 16$, and 32 and only 8 Monte Carlo steps per bead for $N = 64$.) Estimate the accuracy of your calculation by calculating the standard deviations of the means. Also compute the probability $P_N(N)\Delta R$ that the end-to-end distance is R . How does this probability compare to a Gaussian distribution?
- We know that two configurations are strongly correlated if they differ by only the position of one bead. Hence, it would be a waste of computer time to measure the end-to-end distance and the position of the center of mass after every single move. Ideally, we wish to compute these quantities for configurations that are approximately statistically independent. Because we do not know *a priori* the mean number of Monte Carlo steps per bead needed to obtain configurations that are statistically independent, we need to estimate this time in our preliminary calculations. A simple way to estimate this time is to plot the time average of R^2 as a function of the time t . If you start with a configuration that is not typical, then you will notice that the

time average of R^2 eventually approaches a value that no longer changes with t except for small fluctuations. Plot the difference between the time average of R^2 as a function of t and determine if this difference can be roughly approximated by an exponential of the form e^{-t/τ_r} ? The time τ_r is known as the *relaxation time*. If τ_r depends on N , try to quantify this relationship.

- c. The relaxation time τ_r is usually the same order of magnitude as the correlation time τ_c , where τ_c is the time needed to obtain statistically independent configurations. This time can be obtained by computing the equilibrium averaged autocorrelation function for a chain of fixed N :

$$C_N(t) = \frac{\langle R_N^2(t' + t)R_N^2(t') \rangle - \langle R_N^2 \rangle^2}{\langle R_N^4 \rangle - \langle R_N^2 \rangle^2}. \quad (12.20)$$

Note that $C_N(t)$ has the same form as the velocity correlation function (8.40) and the autocorrelation function that is introduced in (12.40). $C_N(t)$ is defined so that $C_N(t=0) = 1$ and $C_N(t) = 0$ if the configurations are not correlated. Because the configurations will become uncorrelated if the time t between the configurations is sufficiently long, we have that $C_N(t) \rightarrow 0$ for $t \gg 1$. In general, we expect that $C(t) \sim e^{-t/\tau_c}$, that is, $C(t)$ decays exponentially with a decay or correlation time τ_c . Estimate τ_c from a plot of $\ln C(t)$ versus t . Another way of estimating τ_c is from the integral $\int_0^\infty dt C(t)$. (Because we determine $C(t)$ at discrete values of t , this integral is actually a sum.) How do your two estimates of τ_c compare? A more detailed discussion of the estimation of correlation times can be found in Section ??.

Another type of random walk that is less constrained than the self-avoiding random walk is the “true” self-avoiding walk (TSAW). The TSAW describes the path of a random walker that avoids visiting a lattice site with a probability that is a function of the number of times the site has been visited already. This constraint leads to a reduced excluded volume interaction in comparison to the usual self-avoiding walk.

Problem 12.14. The true self-avoiding walk in one dimension

In one dimension the true self-avoiding walk corresponds to a walker who can jump to one of two nearest neighbors with a probability that depends on the number of times these neighbors already have been visited. Suppose that the walker is at site i at step t . The probability that at time $t+1$, the walker will jump to site $i+1$ is given by

$$p_{i+1} = \frac{e^{-gn_{i+1}}}{e^{-gn_{i+1}} + e^{-gn_{i-1}}}, \quad (12.21)$$

where $n_{i\pm 1}$ is the number of times that the walker has already visited site $i \pm 1$. The probability of a jump to site $i-1$ is $p_{i-1} = 1 - p_{i+1}$. The parameter g ($g > 0$) is a measure of the “desire” of the path to avoid itself. The first few steps of a typical true self-avoiding walk walk are shown in Figure 12.9. The main quantity of interest is the exponent ν . We know that $g = 0$ corresponds to the usual random walk with $\nu = \frac{1}{2}$ and that the limit $g \rightarrow \infty$ corresponds to the self-avoiding walk. What is the value of ν for a self-avoiding walk in one dimension? Is the value of ν for any finite value of g different than these two limiting cases?

Write a program to do a Monte Carlo simulation of the true self-avoiding walk in one dimension. Use an array to record the number of visits to every site. At each step calculate the probability p of a jump to the right. Generate a random number r and compare it to p . If $r \leq p$,

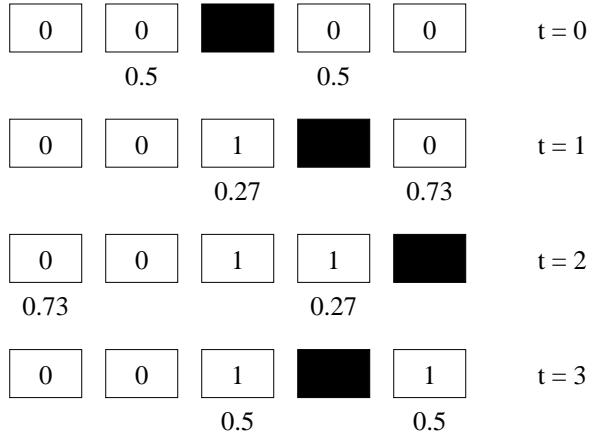


Figure 12.9: Example of the time evolution of the true self-avoiding walk with $g = 1$. The shaded site represents the location of the walker at time t . The number of visits to each site are given within each site and the probability of a step to a nearest neighbor site is given below it. Note the use of periodic boundary conditions.

move the walker to the right; otherwise move the walker to the left. Compute $\langle \Delta x_N^2 \rangle$, where x is the distance of the walker from the origin, as a function of the number of steps N . Make a log-log plot of $\langle \Delta x_N^2 \rangle$ versus N and estimate ν . Can you distinguish ν from its random walk and self-avoiding walk values? Reasonable choices of parameters are $g = 0.1$ and $N \sim 10^3$. Averages over 10^3 trials give qualitative results. For comparison, published results (see Bernasconi and Pietronero) are for $N = 10^4$ and for 10^3 trials; extended results for $g = 2$ are given for $N = 2 \times 10^5$ and 10^4 trials.

12.4 Diffusion Controlled Chemical Reactions

Imagine a system containing particles of a single species A . The particles diffuse, and when two particles collide, a “reaction” occurs such that either one particle is annihilated or the two combine to form an inert species which is no longer involved in the reaction. In the latter case we can represent the chemical reaction as



If we ignore the spatial fluctuations of the density of species A , we can describe the kinetics by a simple rate equation:

$$\frac{dA(t)}{dt} = -kA^2(t), \quad (12.23)$$

where A is the concentration of A particles at time t and k is the rate constant. (In the chemical kinetics literature it is traditional to use the term concentration rather than the number density.) For simplicity, we assume that all reactants are entered into the system at $t = 0$ and that no

reactants are added later (a closed system). It is easy to show that the solution of the first-order differential equation (12.23) is

$$A(t) = \frac{1}{kt + 1/A(0)}, \quad (12.24)$$

and $A(t) \sim t^{-1}$ in the limit of long times.

Another interesting case is the bimolecular reaction



If we neglect fluctuations in the concentration as before (this neglect yields what is known as a mean-field approximation), we can write the corresponding rate equation as

$$\frac{dA(t)}{dt} = \frac{dB(t)}{dt} = -kA(t)B(t). \quad (12.26)$$

We also have that

$$A(t) - B(t) = \text{constant}, \quad (12.27)$$

because each reaction leaves the difference between the concentration of A and B particles unchanged. For the special case of equal initial concentrations, the solution of (12.26) with (12.27) is the same as (12.24). What is the solution for the case $A(0) \neq B(0)$?

The above derivation of the time dependence of A for the kinetics of the one and two species annihilation process is straightforward, but is based on the neglect of spatial fluctuations. In the following two problems, we simulate the kinetics of these processes and test this assumption.

Problem 12.15. Diffusion controlled chemical reactions in one dimension

- Assume that N particles do a random walk on a one-dimensional lattice of length L with periodic boundary conditions. Every particle moves once in one unit of time. It is convenient to associate an array `site(j)` which records the label of the particle, if any, at site j . Because we are interested in the long time behavior of the system when the concentration $A = N/L$ of particles is small, it is efficient to also maintain an array of particle positions, `posx` such that `site(posx(i)) = i`. For example, if particle 5 is located at site 12, then `posx(5) = 12` and `site(12) = 5`. We also need an array, `newSite`, to maintain the new positions of the walkers as they are moved one at a time. If two walkers land on the same position k , then `newSite(k) = 0`, and the value of `posx` for these two walkers to 0. After all the walkers have moved, we let `site = ewSite` for all sites, and remove all the reacting particles in `posx` that have values equal to 0. This operation can be accomplished by replacing any reacting particle in `posx` by the last particle in the array. Begin with all sites occupied, $A(t = 0) = 1$.

Make a log-log plot of the quantity $1/A(t) - 1$ versus the time t . The times should be separated by exponential intervals so that your data is equally spaced on a logarithmic plot. For example, you might include data with times equal to 2^p , with $p = 1, 2, 3, \dots$. Does your log-log plot yield a straight line in the limit of long times? If so, calculate its slope. Is the mean-field approximation for $A(t_0)$ valid in one dimension? You can obtain crude results for small lattices of order $L = 100$ and times of order $t = 10^2$. To obtain results to within ten percent, you need lattices of order $L = 10^4$ and times of order $t = 2^{13}$.

- b. More insight into the origin of the time dependence of $A(t)$ can be gained from the behavior of the quantity $P(r, t)$, the probability distribution of the nearest neighbor distances at time t . The nearest neighbor distance of a particle is defined as the minimum distance between the particle and all other particles. ($P(r)$ is not the same as the radial distribution function $g(r)$ considered in Section 8.8.) The distribution of these distances changes dramatically as the reaction proceeds, and this change can give information about the reaction mechanism. Place particles at random on a one-dimensional lattice and verify that the most probable nearest neighbor distance is $r = 1$ (one lattice constant) for all concentrations. (This result is true in any dimension.) Then verify that the distribution of nearest neighbor distances on a $d = 1$ lattice is given by

$$P(r, t = 0) = 2A e^{-2A(r-1)}. \quad (12.28)$$

Is the form (12.28) properly normalized? Start with $A(t = 0) = 0.1$ and find $P(r, t)$ for $t = 10, 100$, and 1000 . Average over all particles. How does $P(r, t)$ change as the reaction proceeds? Does it retain the same form as the concentration decreases?

- c. Compute the quantity $D(t)$, the number of *distinct* sites visited by an individual walker. How does the time dependence of $D(t)$ compare to the computed time dependence of $1/A(t) - 1$?
- d. Write a program to simulate the $A + B = 0$ reaction. For simplicity, assume that multiple occupancy of the same site is not allowed, for example, an A particle cannot jump to a site already occupied by an A particle. The easiest procedure is to allow a walker to choose one of its nearest neighbor sites at random, but to not move the walker if the chosen site is already occupied by a particle of the same type. If the site is occupied by a walker of another type, then the pair of reacting particles is annihilated. Keep separate arrays for the A and B particles, with the value of the array denoting the label of the particle as before. An easy way to distinguish A and B walkers is to make the array element `site(k)` positive if the site is occupied by an A particle and negative if the site is occupied by a B particle. Start with equal concentrations of A and B particles and occupy the sites at random. Some of the interesting questions are similar to those that we posed in parts (a)–(c). Color code the particles and observe what happens to the relative positions of the particles.

*Problem 12.16. Reaction diffusion in two dimensions

- a. Do a similar simulation as in Problem 12.15 on a two-dimensional lattice for the reaction $A+A \rightarrow 0$. In this case it is convenient to have one array for each dimension, for example, `posx` and `posy`. Set $A(t = 0) = 1$, and choose $L = 50$. Show the configuration of walkers after each Monte Carlo step per walker. Describe the geometry of the clusters of particles as the diffusion process proceeds. Are the particles uniformly distributed throughout the lattice for all times? Calculate $A(t)$ and compare your results for $1/A(t) - 1/A(0)$ to the t -dependence of $D(t)$, the number of distinct lattice sites that are visited in time t . ($D(t) \sim t / \log t$ for two dimensions.) How well do the slopes compare? Do a similar simulation with $A(t = 0) = 0.01$. What slope do you obtain in this case? What can you conclude about the initial density dependence? Is the mean-field approximation valid in this case? If time permits, do a similar simulation in three dimensions.

- b. Begin with A and B type random walkers initially segregated on the left and right halves (in the x direction) of a square lattice. The process $A + B \rightarrow C$ exhibits a reaction front where the production of particles of type C is nonzero. Some of the quantities of interest are the time dependence of the mean position $x(t)$ and the width $w(t)$ of the reaction front. The rules of this process are the same as in part (a) except that a particle of type C is added to a site when a reaction occurs. A particular site can be occupied by one particle of type A or type B as well as any number of particles of type C . If $n(x, t)$ is the number of particles of type C at a distance x from the initial boundary of the reactants, then $x(t)$ and $w(t)$ can be written as

$$x(t) = \frac{\sum_x x n(x, t)}{\sum_x n(x, t)} \quad (12.29)$$

$$w^2(t) = \frac{\sum_x [x - x(t)]^2 n(x, t)}{\sum_x n(x, t)}. \quad (12.30)$$

Choose lattice sizes of order 100×100 , and average over at least 10 trials. The fluctuations in $x(t)$ and $w(t)$ can be reduced by averaging $n(x, t)$ over the order of 100 time units centered about t . More details can be found in Jiang and Ebner.

12.5 The Continuum Limit

In Chapter 7 we showed that the solution of the diffusion equation is equivalent to the long time behavior of a simple random walk on a lattice. In the following, we show directly that the continuum limit of the one-dimensional random walk model is a diffusion equation.

If there is an equal probability of taking a step to the right or left, the random walk can be written in terms of the simple “master” equation

$$P(i, N) = \frac{1}{2}[P(i+1, N-1) + P(i-1, N-1)], \quad (12.31)$$

where $P(i, N)$ is the probability that the walker is at site i after N steps. To obtain a differential equation for the probability density $P(x, t)$, we identify $t = N\tau$, $x = ia$, and $P(i, N) = aP(x, t)$, where τ is the time between steps and a is the lattice spacing. This association allows us to rewrite (12.31) in the equivalent form

$$P(x, t) = \frac{1}{2}[P(x+a, t-\tau) + P(x-a, t-\tau)]. \quad (12.32)$$

We rewrite (12.32) by subtracting $P(x, t-\tau)$ from both sides of (12.32) and dividing by τ :

$$\begin{aligned} \frac{1}{\tau}[P(x, t) - P(x, t-\tau)] &= \frac{a^2}{2\tau}[P(x+a, t-\tau) - 2P(x, t-\tau) \\ &\quad + P(x-a, t-\tau)]a^{-2}. \end{aligned} \quad (12.33)$$

If we expand $P(x, t-\tau)$ and $P(x \pm a, t-\tau)$ in a Taylor series and take the limit $a \rightarrow 0$ and $\tau \rightarrow 0$ with the ratio $D \equiv a^2/2\tau$ finite, we obtain the diffusion equation

$$\frac{\partial P(x, t)}{\partial t} = D \frac{\partial^2 P(x, t)}{\partial x^2}. \quad (12.34a)$$

The generalization of (12.34a) to three dimensions is

$$\frac{\partial P(x, y, z, t)}{\partial t} = D \nabla^2 P(x, y, z, t). \quad (12.34b)$$

where $\nabla^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2 + \partial^2/\partial z^2$ is the Laplacian operator. Equation (12.34) is known as the *diffusion* equation and is frequently used to describe the dynamics of fluid molecules.

The numerical solution of the prototypical *parabolic* partial differential equation (12.34) is a nontrivial problem in numerical analysis (cf. Press et al. or Koonin and Meredith.) An indirect method of solving (12.34) numerically is to use a Monte Carlo method, that is, replace (12.34) by a corresponding random walk on a lattice with discrete time steps. Because the asymptotic behavior of the partial differential equation and the random walk model are equivalent, this approach uses the Monte Carlo technique as a method of *numerical analysis*. In contrast, if our goal is to understand a random walk lattice model directly, the Monte Carlo technique is a *simulation* method. The difference between simulation and numerical analysis is sometimes in the eyes of the beholder.

Problem 12.17. Biased random walk

Show that the form of the differential equation satisfied by $P(x, t)$ corresponding to a random walk with a drift, that is, a walk for $p \neq q$, is

$$\frac{\partial P(x, t)}{\partial t} = D \nabla^2 P(x, y, z, t) - v \frac{\partial P(x, t)}{\partial x} \quad (12.35)$$

How is v related to p and q ?

12.6 Random Number Sequences

So far we have used the random number generator supplied with Java to generate the desired random numbers for our Monte Carlo applications. In principle, we might have generated these numbers from a random physical process, such as the decay of radioactive nuclei or the thermal noise from a semiconductor device. In practice, random number sequences are generated from a physical process only for specific purposes such as a lottery. Although we can store the outcome of a random physical process so that the random number sequence would be both truly random and reproducible, such a method would be inconvenient and inefficient in part because we often require very long sequences. In fact, there are companies that market CD-ROMs and DVDs that contain many millions of random numbers. In practice we use a digital computer, a deterministic machine, to generate sequences of random numbers. These sequences cannot be truly random and are sometimes referred to as *pseudorandom*. However, such a distinction is not important if the sequence satisfies all our criteria for randomness.

Most random number generators yield a sequence in which each number is used to find the succeeding one according to a well defined algorithm. The most important features of a desirable random number generator are that its sequence satisfies the known statistical tests for randomness, the probability distribution is uniform, the sequence has a long period, the method is efficient, the sequence is reproducible, and the algorithm is machine independent.

The most widely used random number generator is based on the *linear congruential* method. That is, given the *seed* x_0 , each number in the sequence is determined by the one-dimensional map

$$x_n = (ax_{n-1} + c) \bmod m, \quad (12.36)$$

where a , c , and m as well as x_n are integers. The notation $y = z \bmod m$ means that m is subtracted from z until $0 \leq y < m$. The map (12.36) is characterized by three parameters, the *multiplier* a , the *increment* c , and the *modulus* m . Because m is the largest integer generated by (12.36), the maximum possible *period* is m .

In general, the period depends on all three parameters. For example, if $a = 3$, $c = 4$, $m = 32$, and $x_0 = 1$, the sequence generated by (12.36) is 1, 7, 25, 15, 17, 23, 9, 31, 1, 7, 25, ... and the period is 8 rather than the maximum possible value of 32. If we are careful to choose a , c , and m such that the maximum period is obtained, then all possible integers between 0 and $m - 1$ will occur in the sequence. Because we usually wish to have random numbers r in the unit interval $0 \leq r < 1$ rather than random integers, random number generators usually return the ratio x_n/m which is always less than unity. One advantage of the linear congruential method is that it is very fast. As the above example illustrates, m , a , and c must be chosen carefully to achieve optimum results. Several rules have been developed (see Knuth) to obtain the longest period. Some of the properties of the linear congruential method are explored in Problem 12.18.

Another popular random number generator is the *generalized feedback shift register* method which uses bit manipulation. Every integer is represented as a series of 1's and 0's called bits. These bits can be shuffled by using the bitwise *exclusive or* operator \oplus defined by $a \oplus b = 1$ if the bits $a \neq b$; $a \oplus b = 0$ if $a = b$. The n th member of the sequence is given by

$$x_n = x_{n-p} \oplus x_{n-q}, \quad (12.37)$$

where $p > q$, and p , q and x_n are integers. The first p random numbers must be supplied by another random number generator. As an example of how the exclusive or operator works, suppose that $n = 6$, $p = 5$, $q = 3$, $x_3 = 11$, and $x_1 = 6$. Then $x_6 = x_1 \oplus x_3 = 0110 \oplus 1011 = 1101 = 2^3 + 2^2 + 2^0 = 8 + 4 + 1 = 13$. Not all values of p and q lead to good results. Some common pairs are $(p, q) = (31, 3)$, $(250, 103)$, and $(521, 168)$. In Fortran the *exclusive or* operation on the integers m and n is given by the intrinsic function `ieor(m,n)`; the equivalent operation in C and Java is given by `m ^ n`.

The above two examples of random number generators illustrate their general nature. That is, numbers in the sequence are used to find the succeeding ones according to a well defined algorithm, and the sequence is determined by the *seed*, the first number of the sequence (or the first p members of the sequence for the generalized feedback shift register method). In general, the maximum possible period is related to the size of the computer word, for example, 16, 32, or 64 bits, that is used. We also note that the choice of the constants and the proper initialization of the sequence is very important and that the algorithm must be implemented with care.

There is no necessary and sufficient test for the randomness of a finite sequence of numbers; the most that can be said about any finite sequence of numbers is that it is apparently random. Because no single statistical test is a reliable indicator, we need to consider several tests. Some of the best known tests are discussed in Problem 12.18. Many of these tests can be stated in terms of random walks.

Problem 12.18. Statistical tests of randomness

- Period.* The most obvious requirement for a random number generator is that its period be much greater than the number of random numbers needed in a specific calculation. One way to visualize the period of the random number generator is to use it to generate a plot of

the displacement x of a random walker as a function of the number of steps N . When the period of the random number is reached, the plot will begin to repeat itself. Generate such a plot using (12.36) for the two choices of parameters $a = 899, c = 0$, and $m = 32768$, and $a = 16807, c = 0, m = 2^{31} - 1$ with $x_0 = 12$. What are the periods of the corresponding random number generators? Obtain similar plots using different values for the parameters a, c , and m . Why is the seed value $x_0 = 0$ forbidden for this choice of c ? Do some combinations of these parameters give longer periods than others? What is the period of the random number generator supplied with the random number generator that you are using?

- b. *Uniformity.* A random number sequence should contain numbers distributed in the unit interval with equal probability. The simplest test of uniformity is to divide this interval into M equal size subintervals or bins and place each member of the sequence into one of the bins. For example, consider the first $N = 10^4$ numbers generated by (12.36) with $a = 106, c = 1283$, and $m = 6075$ (see Press et al.). Place each number into one of $M = 100$ bins. Is the number of entries in each bin approximately equal? What happens if you increase N ?
- c. *Chi-square test.* Is the distribution of numbers in the bins of part (b) consistent with the laws of statistics? The most common test of this consistency is the *chi-square* or χ^2 test. Let y_i be the observed number in bin i and E_i be the expected value. For the example in part (b) with $N = 10^4$ and $M = 100$, we have $E_i = 100$. The chi-square statistic is

$$\chi^2 = \sum_{i=1}^M \frac{(y_i - E_i)^2}{E_i}. \quad (12.38)$$

The magnitude of the number χ^2 is a measure of the agreement between the observed and expected distributions. In general, the individual terms in the sum (12.38) are expected to be order 1, and because there are M terms in the sum, we expect $\chi^2 \leq M$. As an example, we did five independent runs of a random number generator with $N = 10^4$ and $M = 100$, and found $\chi^2 \approx 92, 124, 85, 91$, and 99 . These values of χ^2 are consistent with this expectation. Although we usually want χ^2 to be as small as possible, we would be suspicious if $\chi^2 \approx 0$, because such a small value suggests that N is a multiple of the period of the generator and that each value in the sequence appears an equal number of times.

A more quantitative measure of our confidence that the discrepancy $(y_i - E_i)$ is distributed according to the Gaussian distribution is given by the chi-square probability function $P(x, \nu)$ defined as:

$$P(x, \nu) = \frac{1}{2^{\nu/2}\Gamma(\nu/2)} \int_0^x t^{(\nu-2)/2} e^{-t/2} dt. \quad (12.39)$$

The Gamma function $\Gamma(z)$ in (12.39) is given by $\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$; the familiar relation $\Gamma(z+1) = z!$ holds if z is a positive integer. The quantity ν in (12.39) is the number of degrees of freedom. In our case $\nu = M - 1$, because we have imposed the constraint that $\sum_{i=1}^M E_i = N$. The function $Q(x, \nu) = 1 - P(x, \nu)$ is the probability that the measured value of χ^2 is greater than x . For our example we can solve for x in the equation $Q(x, \nu) = q$ with $\nu = 99$ for various values of q or find the solution in a statistical table. (A quick search of the Web will yield several sites that will compute x for particular values of ν and probability q .) For $\nu = 99$, we find that $x \approx 139$ for $q = 0.005$, $x \approx 123$ for $q = 0.05$, $x \approx 111$ for $q = 0.2$, and $x \approx 98$ for $q = 0.5$.

Our above results for χ^2 show that $\chi^2 > 123$ for one run out of five (20%). Because 123 is the value of x at the 5% level, we expect to see $\chi^2 \geq 123$ in only one out of twenty runs. Hence, our confidence level is less than 95%. Instead, we can assume an approximately 80% confidence level in our random number generator because the value of x for this confidence level is 111. We might be able to increase our confidence level by doing more runs. Suppose that we make twenty runs and we still find only one measurement of χ^2 greater than 123. In this case our confidence level would rise to 95%. Determine χ^2 for twenty independent runs using the values of a , c , and m given in parts (12.18) and (b). Estimate your level of confidence in these random number generators.

- d. *Filling sites.* Although a random number sequence might contain numbers that are distributed in the unit interval with equal probability, consecutive numbers might not appear in a perfectly uniform way, but have a tendency to be clumped or correlated in some way. One test of this correlation is to fill a square lattice of L^2 sites at random. Consider an array $n(x, y)$ that is initially empty, where $1 \leq x_i, y_i \leq L$. A point is selected randomly by choosing its two coordinates x_i and y_i from two consecutive numbers in the sequence. If the site is empty, it is filled and $n(x_i, y_i) = 1$; otherwise it is not changed. This procedure is repeated tL^2 times, where t is the number of Monte Carlo steps per site. Because this process is analogous to the decay of radioactive nuclei, we expect that the fraction of empty lattice sites should decay as e^{-t} . Determine the fraction of unfilled sites using the random number generator that you have been using for $L = 10$, 15 , and 20 . Are your results consistent with the expected fraction? Repeat the same test using (12.36) with $a = 65549$, $c = 0$, and $m = 231$. The existence of triplet correlations can be determined by a similar test on a simple cubic lattice by choosing the three coordinates x_i , y_i , and z_i from three consecutive random numbers.
- e. *Hidden correlations.* Sometimes a picture is worth a thousand numbers. Another way of checking for correlations is to plot x_{i+k} versus x_i . If there are any obvious patterns in the plot, then there is something wrong with the generator. Use the generator (12.36) with $a = 16807$, $c = 0$, and $m = 2^{31} - 1$. Can you detect any structure in the plotted points for $k = 1$ to $k = 5$? Also test the random number generator that you have been using. Do you see any evidence of lattice structure, for example, equidistant parallel lines? Is the logistic map $x_{n+1} = 4x_n(1 - x_n)$ for $r = 1$ a suitable random number generator?
- f. *Short-term correlations.* Another measure of short term correlations is the autocorrelation function

$$C(k) = \frac{\langle x_{i+k}x_i \rangle - \langle x_i \rangle^2}{\langle x_i x_i \rangle - \langle x_i \rangle \langle x_i \rangle}, \quad (12.40)$$

where x_i is the i th term in the sequence. We have used the fact that $\langle x_{i+k} \rangle = \langle x_i \rangle$, that is, the choice of the origin of the sequence is irrelevant. The quantity $\langle x_{i+k}x_i \rangle$ is found for a particular choice of k by forming all the possible products of $x_{i+k}x_i$ and dividing by the number of products. If x_{i+k} and x_i are not correlated, then $\langle x_{i+k}x_i \rangle = \langle x_{i+k} \rangle \langle x_i \rangle$ and $C(k) = 0$. Is $C(k)$ identically zero for any finite sequence? Compute $C(k)$ for $a = 106$, $c = 1283$, and $m = 6075$.

- g. *Random walk.* Another test based on the properties of random walks has been proposed (see Vattulainen et al.). Assume that a walker begins at the origin of the x - y plane and generate n_w walks of N steps. Count the number of walks in each quadrant q_i of the x - y plane, and use the

χ^2 test (12.38) with $y_i \rightarrow q_i$, $M = 4$, and $E_i = n_w/4$. If $\chi^2 > 7.815$ (a 5% probability if the random number generator is perfect), we say that the run fails. The random number generator fails if two out of three independent runs fail. The probability of a perfect generator failing two out of three runs would be approximately $3 \times 0.95 \times (0.05)^2 \approx 0.007$. Test several random number generators.

**Problem 12.19.* An “improved” random number generator

One way to reduce sequential correlation and to lengthen the period is to mix or *shuffle* two different random number generators. The following procedure illustrates the approach for two random number generators that we denote as `ran1` and `ran2`.

1. Make a list of 256 random numbers using `ran1`. (The number 256 is arbitrary, but should be less than the period of `ran1`.)
2. Choose a random number x from this list by using `ran2` to generate a random index between 1 and 256. The integer x is the desired number.
3. Replace the number chosen in step 2 by a new random number generated by `ran1`.

Consider two random number generators with relatively short periods and strong sequential correlation and show that the above shuffling scheme improves the quality of the random numbers.

At least some of the statistical tests given in Problem 12.18 should be done whenever serious calculations are contemplated. However, even if a random number generator passes all these tests, there still can be problems in rare cases. Typically, these problems arise when a small number of events have a large weight. In these cases a very small bias in the random number generator might lead to a systematic error in the final results, and two generators, which appear equally good as determined by various statistical tests, might give statistically different results when used in a specific application. For this reason, it is important that the random number generator that is used be reported along with the actual results. Confidence in the results also can be increased by repeating the calculation with another random number generator.

Because all random number generators are based on a deterministic algorithm, it always is possible to construct a test generator for which a particular algorithm will fail. The success of a random number generator in passing various statistical tests is necessary and improves our overall confidence in its statistical properties, but it is not a sufficient condition for their use in all applications. In Project ?? we discuss an application of Monte Carlo methods to the Ising model for which some commonly used random number generators give incorrect results.

12.7 Projects

Almost all of the problems in this chapter can be done using more efficient programs, greater number of trials, and larger systems. More applications of random walks and random number sequences are discussed in subsequent chapters. Many more ideas for projects can be gained from the references.

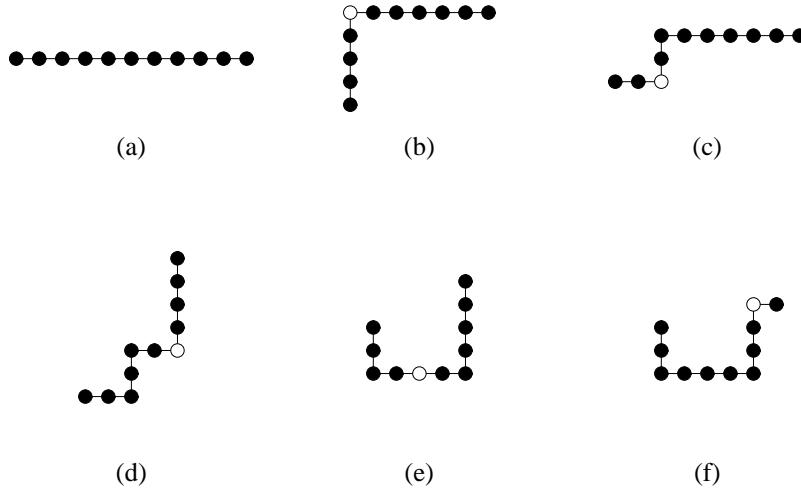


Figure 12.10: Examples of the first several changes generated by the pivot algorithm for a self-avoiding walk of $N = 10$ steps (11 sites). The open circle denotes the pivot point. This figure is adopted from the article by MacDonald et al.

Project 12.20. Application of the pivot algorithm

The algorithms that we have discussed for generating self-avoiding random walks are all based on making *local* deformations of the walk (polymer chain) for a given value of N , the number of bonds. As discussed in Problem 12.13, the time τ_c between statistically independent configurations is nonzero. The problem is that τ_c increases with N as some power, for example, $\tau_c \sim N^3$. This power law dependence of τ_c on N is called *critical slowing down* and implies that it becomes increasingly more time consuming to generate long walks. We now discuss an example of a *global* algorithm that reduces the dependence of τ_c on N . Another example of a global algorithm that reduces critical slowing down is discussed in Project ??.

- Consider the walk shown in Figure 12.10a. Select a site at random and one of the four possible directions. The shorter portion of the walk is rotated (pivoted) to this new direction by treating the walk as a rigid structure. The new walk is accepted only if the new walk is self-avoiding; otherwise the old walk is retained. (The shorter portion of the walk is chosen to save computer time.) Some typical moves are shown in Figure 12.10. Note that if an end point is chosen, the previous walk is retained. Write a program to implement this algorithm and compute the dependence of the mean square end-to-end distance $\langle R_N^2 \rangle$ on N . Consider values of N in the range $10 \leq N \leq 80$. A discussion of the results and the implementation of the algorithm can be found in MacDonald et al. and Madras and Sokal, respectively.
- Compute the correlation time τ_c for different values of N using the approach discussed in Problem 12.13c.

Project 12.21. A simple reaction diffusion model

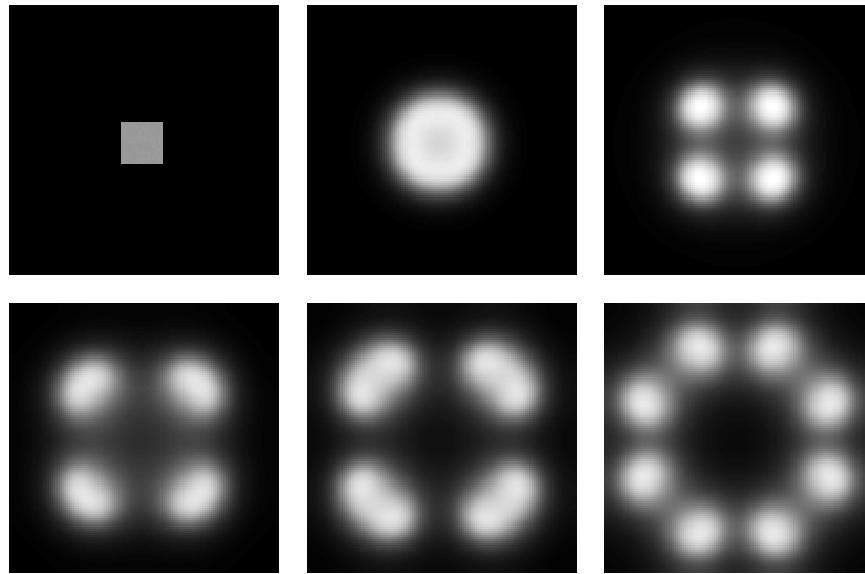


Figure 12.11: Evolution of the pattern starting from the initial conditions suggested in Project c.

In Problem 12.16 we saw that simple patterns can develop as a result of random behavior. The phenomenon of pattern formation is of much interest in a variety of contexts ranging from the large scale structure of the universe to the roll patterns seen in convection (for example, smoke rings). In the following, we explore the patterns that can develop in a simple reaction diffusion model based on the reactions, $A + 2B \rightarrow 3B$, and $B \rightarrow C$, where C is inert. Such a reaction is called *autocatalytic*. In Problem 12.16 we considered chemical reactions in a closed system where the reactions can proceed to equilibrium. In contrast, open systems allow a continuous supply of fresh reactants and a removal of products. These two processes allow steady states to be realized and oscillatory conditions to be maintained indefinitely. The model of interest also assumes that A is added at a constant rate and that both A and B are removed by the feed process. Pearson (see references) models these processes by two coupled reaction diffusion equations:

$$\frac{\partial A}{\partial t} = D_A \nabla^2 A - AB^2 + f(1 - A) \quad (12.41a)$$

$$\frac{\partial B}{\partial t} = D_B \nabla^2 B + AB^2 - (f + k)B. \quad (12.41b)$$

The AB^2 term represents the reaction $A + 2B \rightarrow 3B$. This term is negative in (12.41a) because the reactant A decreases, and is positive in (12.41b) because the reactant B increases. The term $+f$ represents the constant addition of A , and the terms $-fA$ and $-fB$ represent the removal process; the term $-kB$ represents the reaction $B \rightarrow C$. All the quantities in (12.41) are dimensionless. We assume that the diffusion coefficients are $D_A = 2 \times 10^{-5}$ and $D_B = 10^{-5}$, and the behavior of the system is determined by the values of the rate constant k and the feed rate f .

- a. We first consider the behavior of the reaction kinetics that results when the diffusion terms

in (12.41) are neglected. It is clear from (12.41) that there is a trivial steady state solution $A = 1, B = 0$. Are there other solutions, and if so, are they stable? The steady state solutions can easily be found by solving (12.41) with $\partial A / \partial t = \partial B / \partial t = 0$. To determine the stability, we can add a perturbation and determine whether the perturbation grows or not. However, without the diffusion terms, it is more straightforward to solve (12.41) numerically using a simple Euler algorithm. Choose a time step equal to unity, and let $A = 0.1$ and $B = 0.5$ at $t = 0$. Determine the steady state values for $0 < f \leq 0.3$ and $0 < k \leq 0.07$ in increments of $\Delta f = 0.02$ and $\Delta k = 0.005$. Record the steady state values of A and B . Then repeat this exercise for the initial values $A = 0.5$ and $B = 0.1$. You should find that for some values of f and k , only one steady state solution can be obtained for the two initial conditions, and for other initial values of A and B there are two steady state solutions. Try other initial conditions. If you obtain a new solution, change the initial A or B slightly to see if your new solution is stable. On an f versus k plot indicate where there are two solutions and where there are one. In this way you can determine the approximate phase diagram for this process.

- b. There is a small region in f - k space where one of the steady state solutions becomes unstable and periodic solutions occur (the mechanism is known as a Hopf bifurcation). Try $f = 0.009$, $k = 0.03$, and set $A = 0.1$ and $B = 0.5$ at $t = 0$. Plot the values of A and B versus the time t . Are they periodic? Try other values of f and k and estimate where the periodic solutions occur.
- c. Numerical solutions of the full equation with diffusion (12.41) can be found by making a finite difference approximation to the spatial derivatives as in (3.19) and using a simple Euler algorithm for the time integration. Adopt periodic boundary conditions. Although it is straightforward to write a program to do the numerical integration, an exploration of the dynamics of this system requires a supercomputer. However, we can find some preliminary results with a small system and a coarse grid. Consider a 0.5×0.5 system with a spatial mesh of 128×128 grid points on a square lattice. Choose $f = 0.18$, $k = 0.057$, and $\Delta t = 0.1$. Let the entire system be in the initial trivial state ($A = 1, B = 0$) except for a 20×20 grid located at the center of the system where the sites are $A = 1/2, B = 1/4$ with a $\pm 1\%$ random noise. The effect of the noise is to break the square symmetry. Let the system evolve for approximately 80,000 time steps and look at the patterns that develop. Color code the grid according to the concentration of A , with red representing $A = 1$ and blue representing $A \approx 0.2$ and with several intermediate colors. Very interesting patterns have been found by Pearson.

References and Suggestions for Further Reading

- Daniel J. Amit, G. Parisi, and L. Peletti, “Asymptotic behavior of the “true” self-avoiding walk,” *Phys. Rev. B* **27**, 1635–1645 (1983).
- Panos Argyrakis, “Simulation of diffusion-controlled chemical reactions,” *Computers in Physics* **6**, 525 (1992).
- J. Bernasconi and L. Pietronero, “True self-avoiding walk in one dimension,” *Phys. Rev. B* **29**, 5196 (1984). The authors present results for the exponent ν accurate to 1%.

Roan Dawkins and Daniel ben-Avraham, “Computer simulations of diffusion-limited reactions,” *Comput. Sci. Eng.* **3**(1), 72–76 (2001).

Neal Madras and Alan D. Sokal, “The pivot algorithm: a highly efficient Monte Carlo method for the self-avoiding walk,” *J. Stat. Phys.* **50**, 109–186 (1988). The pivot algorithm was invented by Moti Lai, “Monte Carlo’ computer simulation of chain molecules I,” *Mol. Phys.* **17**, 57–64 (1969) See also Neal Madras and Gordon Slade, *The Self-Avoiding Walk*, Birkhauser (1993) and “Monte Carlo methods for the self-avoiding walk,” in *Monte Carlo and Molecular Dynamics Simulations in Polymer Science*, Kurt Binder, editor, Oxford University Press (1995).

S. Chandrasekhar, “Stochastic problems in physics and astronomy,” *Rev. Mod. Phys.* **15**, 1 (1943). This article is reprinted in M. Wax, *Selected Papers on Noise and Stochastic Processes*, Dover (1954).

Mohamed Daoud, “Polymers,” Chapter 6 in Armin Bunde and Shlomo Havlin, editors, *Fractals in Science*, Springer-Verlag (1994).

Robert Ehrlich, *Physics and Computers*, Houghton Mifflin (1973). See Chapter 4 for a discussion of the linear congruential method.

R. Everaers, I. S. Graham, and M. J. Zuckermann, “End-to-end distance and asymptotic behavior of self-avoiding walks in two and three dimensions,” *J. Phys. A* **28**, 1271 (1995).

Pierre-Giles de Gennes, *Scaling Concepts in Polymer Physics*, Cornell University Press (1979). An important but difficult text.

Shlomo Havlin and Daniel Ben-Avraham, “Diffusion in disordered media,” *Adv. Phys.* **36**, 695 (1987). Section 7 of this review article discusses trapping and diffusion-limited reactions. See also Daniel Ben-Avraham and Shlomo Havlin, *Diffusion and Reactions in Fractals and Disordered Systems*, Cambridge University Press (2001).

Shlomo Havlin, George H. Weiss, James E. Kiefer, and Menachem Dishon, “Exact enumeration of random walks with traps,” *J. Phys. A: Math. Gen.* **17**, L347, (1984). The authors discuss a method based on exact enumeration for calculating the survival probability of random walkers on a lattice with randomly distributed traps.

Brian Hayes, “How to Avoid Yourself,” *Amer. Scientist* **86**(#4), 314 (1998).

Z. Jiang and C. Ebner, “Simulation study of reaction fronts,” *Phys. Rev. A* **42**, 7483 (1990).

Steven E. Koonin and Dawn C. Meredith, *Computational Physics*, Addison-Wesley (1990).

Donald E. Knuth, *Seminumerical Algorithms*, second edition, Vol. 2 of *The Art of Computer Programming*, Addison-Wesley (1981). The standard reference on random number generators.

Bruce MacDonald, Naeem Jan, D. L. Hunter, and M. O. Steinitz, “Polymer conformations through ‘wiggling’,” *J. Phys. A* **18**, 2627 (1985). A discussion of the pivot algorithm discussed in Project 12.20.

Elliott W. Montroll and Michael F. Shlesinger, "On the wonderful world of random walks," in *Nonequilibrium Phenomena II: From Stochastics to Hydrodynamics*, J. L. Lebowitz and E. W. Montroll, editors, North-Holland Press (1984). The first part of this delightful review article chronicles the history of the random walk.

John E. Pearson, "Complex patterns in a simple fluid," *Science* **261**, 189 (1993). See also P. Gray and S. K. Scott, "Sustained oscillations and other exotic patterns of behavior in isothermal reactions," *J. Phys. Chem.* **89**, 22 (1985).

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes*, second edition, Cambridge University Press (1992).

Sidney Redner and Francois Leyvraz, "Kinetics and spatial organization of competitive reactions," Chapter 7 in Armin Bunde and Shlomo Havlin, editors, *Fractals in Science*, Springer-Verlag (1994).

F. Reif, *Fundamentals of Statistical and Thermal Physics*, McGraw-Hill (1965). This well known text on statistical physics has a good discussion on random walks (Chapter 1) and diffusion (Chapter 12).

Marshall N. Rosenbluth and Arianna W. Rosenbluth, "Monte Carlo calculation of the average extension of molecular chains," *J. Chem. Phys.* **23**, 356 (1955). One of the first Monte Carlo calculations for the self-avoiding walk.

John R. Taylor, *An Introduction to Error Analysis*, University Science Books, Oxford University Press (1982). See Chapter 12 for a discussion of the χ^2 test.

I. Vattulainen, T. Ala-Nissila, and K. Kankaala, "Physical tests for random numbers in simulations," *Phys. Rev. Lett.* **73**, 2513 (1994).

Peter H. Verdier and W. H. Stockmayer, "Monte Carlo calculations on the dynamics of polymers in dilute solution," *J. Chem. Phys.* **36**, 227 (1962).

Frederick T. Wall and Frederic Mandel, "Macromolecular dimensions obtained by an efficient Monte Carlo method without sample attrition," *J. Chem. Phys.* **63**, 4592 (1975). An exposition of the reptation method.

George H. Weiss, "A primer of random walkology," Chapter 5 in Armin Bunde and Shlomo Havlin, editors, *Fractals in Science*, Springer-Verlag (1994).

George H. Weiss and Shlomo Havlin, "Trapping of random walks on the line," *J. Stat. Phys.* **37**, 17 (1984). The authors discuss an analytical approach to the asymptotic behavior of one-dimensional random walkers with randomly placed traps.

George H. Weiss and Robert J. Rubin, "Random walks: theory and selected applications," *Adv. Chem. Phys.* **52**, 363 (1983). In spite of its research orientation, much of this review article can be understood by the well motivated student.

Charles A. Whitney, *Random Processes in Physical Systems*, John Wiley and Sons (1990). A good introduction to random processes including random walks.

Charles A. Whitney, “Generating and testing pseudorandom numbers,” *Byte*, pp. 128 (October, 1984). An accessible and informative article.

Chapter 13

Percolation

©2001 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
24 April 2001

We introduce several concepts associated with critical phenomena in the context of percolation.

13.1 Introduction

Our discussion of percolation and geometrical phase transitions requires little background in physics, e.g., no classical or quantum mechanics and little statistical physics. All that is required is some understanding of geometry and probability. Much of the appeal of percolation models is their game-like aspects and their intuitive simplicity. Moreover, these models serve as an excellent introduction to discrete computer models and the importance of graphical analysis. On the other hand, if you have a background in physics, this chapter will be more meaningful and can serve as an introduction to phase transitions and to important ideas such as scaling relations, critical exponents, and the renormalization group.

Although the term “percolation” might be familiar to you in the context of the brewing of coffee, our use of the term has little to do with it. Instead, we consider another example from the kitchen and imagine a large metal sheet on which we randomly place drops of cookie dough. Assume that each drop of cookie dough can spread while the cookies are baking in an oven. If two cookies touch, they coalesce to form one cookie. If we are not careful, we might find a very large cookie that spans from one edge of the sheet to the opposite edge (see Fig. 13.1). If such a spanning cookie exists, we say that there has been a *percolation transition*. If such a cookie does not exist, the cookie sheet is below the percolation threshold. (There is no percolation transition for ground coffee, because the water dissolves some of the ground coffee beans and flows, regardless of the density of the ground coffee.)

Let us make the cookie example more abstract to make the concept of percolation more clear. We represent the cookie sheet by a lattice where each site can be in one of two states, occupied (by a cookie) or empty. Each site is occupied independently of its neighbors with probability p . This model of percolation is called *site* percolation. The occupied sites either are isolated or form

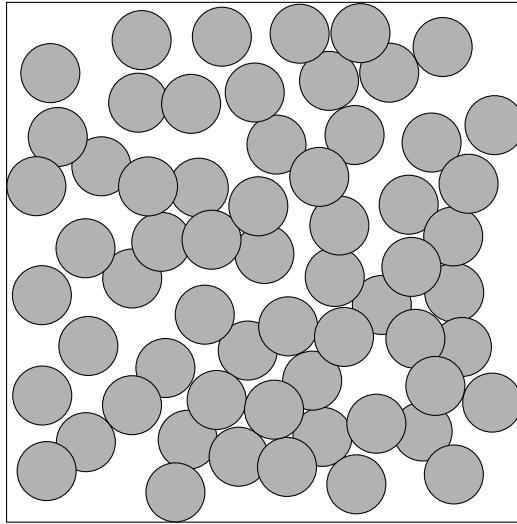


Figure 13.1: Cookies (circles) placed at random on a large sheet. Note that there is a path of overlapping circles that connects the bottom and top edges of the cookie sheet. If such a path exists, we say that the cookies “percolate” the lattice or that there is a “spanning path.” See Problem 13.4d for a discussion of the algorithm used to generate this configuration.

groups of nearest neighbors. We define a *cluster* as a group of occupied nearest neighbor lattice sites (see Fig. 13.2).

An easy way to study percolation uses the random number generator on a calculator. The procedure is to generate a random number r in the unit interval $0 < r \leq 1$ for each site in the lattice. A site is occupied if its random number satisfies the condition $r \leq p$. If p is small, we expect that only small isolated clusters will be present (see Fig. 13.3a). If p is near unity, we expect that most of the lattice will be occupied, and the occupied sites will form a large cluster that extends from one end of the lattice to the other (see Fig. 13.3c). Such a cluster is said to be a *spanning cluster*. Because there is no spanning cluster for small p and there is a spanning cluster for p near unity, there must be an intermediate value of p at which a spanning cluster first exists (see Fig. 13.3b). We shall see that in the limit of an infinite lattice, there exists a well defined threshold probability p_c such that:

For $p < p_c$, no spanning cluster exists and all clusters are finite.

For $p \geq p_c$, one spanning cluster exists.

We emphasize that the defining characteristic of percolation is *connectedness*. Because the connectedness exhibits a qualitative change at a well defined value of a continuous parameter, we shall see that the transition from a state with no spanning cluster to a state with one spanning cluster is a type of *phase transition*.

Our real interest is not in large cookies or in abstract models, but in the applications of percolation. An example of the application of percolation is to the electrical conductivity of

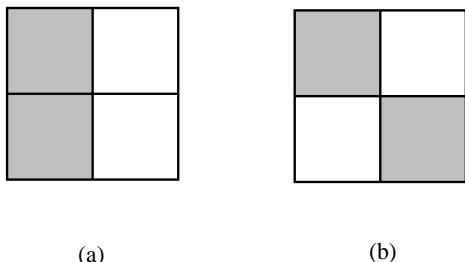


Figure 13.2: Example of a site percolation cluster on a square lattice of linear dimension $L = 2$. The two nearest neighbor occupied sites (shaded) in (a) are part of the same cluster; the two occupied sites in (b) are not nearest neighbor sites and do not belong to the same cluster.

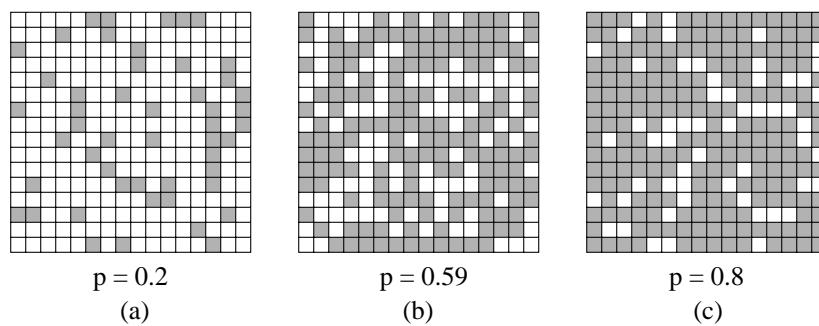


Figure 13.3: Examples of site percolation clusters on a square lattice of linear dimension $L = 16$ for $p = 0.2, 0.59$, and 0.8 . On the average, the fraction of occupied sites (shaded squares) is equal to p . Note that in this example, there exists a cluster that “spans” the lattice horizontally and vertically for $p = 0.59$.

composite systems made of a mixture of metallic and insulating materials. An easy way to make such a system is to place a mixture of small plastic and metallic spheres of equal size into a container (see Fitzpatrick et al.). Care must be taken to pack the spheres at random. If the metallic domains constitute a small fraction of the volume of the system, electricity cannot be conducted and the composite system is an insulator. However, if the metallic domains comprise a sufficiently large fraction of the container, electricity can flow from one domain to another and the composite system is a conductor. The description of the conduction of electricity through composite materials can be made more precise by introducing the parameter ϕ , the volume fraction of the container that consists of metallic spheres. The transition between the two types of behavior (nonconducting and conducting) occurs abruptly as ϕ (the analog of p) is increased and is associated with the nonexistence or existence of a *connected path* of metallic spheres. More realistic composite systems are discussed in Zallen’s book.

Percolation phenomena also can be observed with a piece of chicken wire or wire mesh. Watson and Leath measured the electrical conductivity of a large piece of uniform steel-wire screen mesh

as a function of the fraction of the nodes that are present. The coordinates of the nodes to be removed were given by a random number generator. The measured electrical conductivity is a rapidly decreasing function of the fraction of nodes p still present and vanishes below a critical threshold. A related conductivity measurement on a sheet of conducting paper with random “holes” has been performed (see Mehr et al.).

The applications of percolation phenomena go beyond metal-insulator transitions and the conductivity of chicken wire, and include the spread of disease in a population, the behavior of magnets diluted by nonmagnetic impurities, the flow of oil through porous rock, and the characterization of gels. We concentrate on understanding several simple models of percolation that have an intuitive appeal of their own. Many of the applications of percolation phenomena are discussed in the references.

13.2 The Percolation Threshold

Because it is not convenient to generate many percolation configurations using a calculator, we develop a simple program to do so. Consider a square lattice of linear dimension L and unit lattice spacing, and associate a random number between zero and one with each site in the lattice. A site is occupied if its random number is less than p . **Program site**, listed below, generates site percolation configurations and shows the occupied sites as filled circles of diameter unity. The program uses the FLOOD statement in True BASIC so that if the user clicks on an occupied site, all the sites that are connected to it are changed to the same color and the clusters can be identified visually. A click of the mouse outside the lattice increases p by the desired amount. The array **rsite** stores the random number associated with each lattice site.

The percolation threshold p_c is defined as the probability p at which a spanning cluster first appears in an infinite lattice. However, for the finite lattices of linear dimension L that we can simulate on a computer, there is a nonzero probability of a spanning cluster connecting one side of the lattice to the opposite side for any value of p . For small p , this probability is order p^L (see Fig. 13.4). This probability goes to zero as L becomes large, and hence for small p and sufficiently large L , only finite clusters exist. For a finite lattice, the definition of spanning is arbitrary. For example, we can define a connected path as one that (i) spans the lattice either horizontally or vertically; (ii) spans the lattice in a fixed direction, e.g., vertically; or (iii) spans the lattice both horizontally and vertically. In addition, the criteria for defining $p_c(L)$ for a finite lattice are somewhat arbitrary. One possibility is to define $p_c(L)$ as the average value of p at which a spanning cluster first appears. Another possibility is to define $p_c(L)$ as the value of p for which half of the configurations generated at random span the lattice. These criteria should lead to the same extrapolated value for p_c in the limit $L \rightarrow \infty$. In Problem 13.1 we will find an estimated value for $p_c(L)$ that is accurate to about 10%. A more sophisticated analysis discussed in Project 13.1413.35 allows us to extrapolate our results for $p_c(L)$ to $L \rightarrow \infty$.

Problem 13.1. Site percolation on the square lattice

- Use **Program site** to generate random site configurations on a square lattice. Estimate $p_c(L)$ by finding the average value of p at which a spanning cluster is first attained. Choose $L = 4$ and begin at a value of p for which a spanning cluster is unlikely to be present. Then increase p in increments of **delta-p**= 0.01 until you find a spanning cluster. Record the value of p

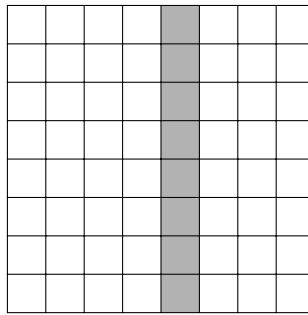


Figure 13.4: An example of a spanning cluster of probability p^L on a $L = 8$ lattice. How many other ways are there of realizing a spanning cluster of L sites?

at which spanning first occurs for each spanning criteria. Repeat this process for a total of ten configurations and find the average value of $p_c(L)$. (Remember that each configuration corresponds to a different set of random numbers.) Are your results for $p_c(L)$ using the three spanning criteria consistent with your expectations?

- b. Repeat part (a) for $L = 16$ and 32 . Is $p_c(L)$ better defined for larger L , that is, are the values of $p_c(L)$ spread over a smaller range of values? How quickly can you visually determine the existence of a spanning cluster? Describe your visual “algorithm” for determining if a spanning cluster exists.

The value of p_c depends on the symmetry of the lattice and on its dimension. In addition to the square lattice, the most common two-dimensional lattice is the triangular lattice. As discussed in Chapter 8, the essential difference between the square and triangular lattices is in the number of nearest neighbors.

**Problem 13.2.* Site percolation on the triangular lattice

Modify `Program site` to simulate random site percolation on a triangular lattice. Assume that a connected path connects the top and bottom sides of the lattice (see Fig. 13.5). Do you expect p_c for the triangular lattice to be smaller or larger than the value of p_c for the square lattice? Estimate $p_c(L)$ for $L = 4, 16$, and 32 . Are your results for p_c consistent with your expectations?

In *bond* percolation each lattice site is occupied, and only a fraction of the sites have connections or bonds between them and their nearest neighbor sites (see Fig. 13.6). Each bond either is occupied with probability p or not occupied with probability $1 - p$. A cluster is a group of sites connected by occupied bonds. The wire mesh described in Section 13.1 is an example of bond percolation if we imagine cutting the bonds between the nodes rather than removing the nodes themselves. An application of bond percolation to the description of gelation is discussed in Problem 13.3.

**Problem 13.3.* Bond percolation on a square lattice Suppose that all the lattice sites of a square lattice are occupied by monomers, each with functionality four, i.e., each monomer can react to form a maximum of four bonds. This model is equivalent to bond percolation on a square lattice. Also assume that the presence or absence of a bond between a given pair of monomers is random

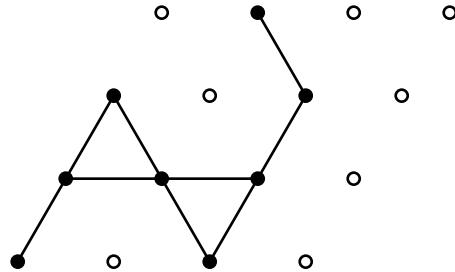


Figure 13.5: Example of a spanning cluster on a $L = 4$ triangular lattice. The bonds between the occupied sites are drawn to clarify the symmetry of the lattice.



Figure 13.6: Two examples of bond clusters. The occupied bonds are shown as bold lines.

and is characterized by a probability p . For small p , the system consists of only finite polymers (groups of monomers) and the system is in the *sol* phase. For some threshold value p_c , there will be a single polymer that is infinite in spatial extent. We say that for $p \geq p_c$, the system is in the *gel* phase. How does a bowl of jello, an example of a gel phase, differ from a bowl of broth? Write a program to simulate bond percolation on a square lattice and determine the bond percolation threshold. Are your results consistent with the exact result, $p_c = 1/2$?

We also can consider *continuum* percolation models. For example, we can place disks at random into a two-dimensional box. Two disks are in the same cluster if they touch or overlap. A typical continuum (off-lattice) percolation configuration is depicted in Fig. 13.7. One quantity of interest is the quantity ϕ , the fraction of the area (volume in three dimensions) in the system that is covered by disks. In the limit of an infinite size box, it can be shown that

$$\phi = 1 - e^{-\rho\pi r^2}, \quad (13.1)$$

where ρ is the number of disks per unit area, and r is the radius of a disk (see Xia and Thorpe). Equation (13.1) is significantly inaccurate for small boxes because disks located near the edge of the box might have a significant fraction of their area located outside of the box. Program site can be modified to simulate continuum percolation. Instead of placing the disks on regular lattice sites, place them at random within a square box of area L^2 . The relevant parameter is the density ρ , the number of disks per unit area, instead of the probability p . Because the disks overlap, it is convenient to replace the BOX SHOW statement in Program site with

```
BOX SHOW occup$ at x(i)-0.5,y(i)-0.5 using "or"
```

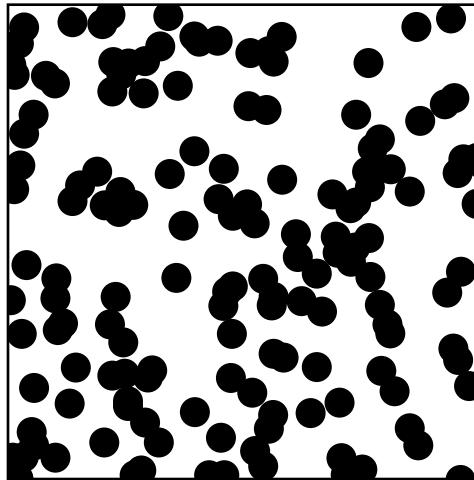


Figure 13.7: A model of continuum (off-lattice) percolation realized by placing disks of unit diameter at random into a square box of linear dimension L . If we concentrate on the voids between the disks rather than the disks, then this model of continuum percolation is known as the Swiss cheese model.

where the arrays $x(i)$ and $y(i)$ are used to store the disk positions of disk i . It also is a good idea to set the background color to red (not black or white).

Problem 13.4. Continuum percolation

- For site percolation, we can define ϕ as the area fraction covered by the disks that are placed on the sites as in [Program site](#). Convince yourself that $\phi_c = (\pi/4)p_c$ (for disks of unit diameter and unit lattice spacing). It is easy to do a Monte Carlo calculation of the area covered by the disks to confirm this result. (Choose points at random in the box and calculate the fraction of points within any disk.)
- Modify [Program site](#) to simulate continuum percolation as discussed in the text. Estimate the value of the percolation threshold ρ_c . Given this value of ρ_c , use a Monte Carlo method to estimate the corresponding area fraction ϕ_c , and compare the value of ϕ_c for site and continuum percolation. Explain why you might expect ϕ_c to be bigger for continuum percolation than for site percolation. Compare your direct Monte Carlo estimate of ϕ_c with the indirect value of ϕ_c obtained from (13.1) using the value of ρ_c . Explain any discrepancy.
- Consider the simple model of the cookie problem discussed in Section 13.1. Write a program that places disks at random into a square box and chooses their diameter randomly between 0 and 1. Estimate the value of ρ_c at which a spanning cluster first appears. How is the value of

- ρ_c changed from your estimate found in part (b)? Is your value for ϕ_c more or less than what was found in part (b)?
- d. A more realistic model of the cookie problem is to place disks with unit diameter at random into a square box with the constraint that the disks do not overlap. Continue to add disks until the probability of placing an additional disk becomes less than 1%, i.e., when one hundred successive attempts at adding a disk are not successful. Then increase the diameters of all the disks at a constant rate (in analogy to the baking of the cookies) until a spanning cluster is attained. How does ϕ_c for this model compare with ϕ_c found in part (c)?
 - e. A continuum model that is applicable to random porous media is known as the *Swiss cheese* model. In this model the relevant quantity (the cheese) is the space between the disks. For the Swiss cheese model in two dimensions, the cheese area fraction at the percolation threshold, $\tilde{\phi}_c$, is given by $\tilde{\phi}_c = 1 - \phi_c$, where ϕ_c is the disk area fraction at the threshold of the disks. Do you think such a relation holds in three dimensions (see Project 13.15)? Imagine that the disks are conductors and that the cheese is an insulator and let $\sigma(\phi)$ denote the conductivity of this system. Alternatively, we can imagine that the cheese is a conductor and the disks are insulators and define a conductivity $\sigma(\tilde{\phi})$. Do you think that $\sigma(\phi) = \sigma(\tilde{\phi})$ when $\phi = \tilde{\phi}$? This question is investigated in Project 13.15.

Our discussion of percolation has emphasized the existence of the percolation threshold p_c and the appearance of a spanning path or cluster for $p \geq p_c$. Another quantity that characterizes percolation is $P_\infty(p)$, the probability that an occupied site belongs to the spanning cluster. P_∞ is defined as

$$P_\infty = \frac{\text{number of sites in the spanning cluster}}{\text{total number of occupied sites}}. \quad (13.2)$$

As an example, $P_\infty(p = 0.59) = 140/154$ for the single configuration shown in Fig. 13.3b. A realistic calculation of P_∞ involves an average over many configurations for a given value of p . For an infinite lattice, $P_\infty(p) = 0$ for $p < p_c$ and $P_\infty(p) = 1$ for $p = 1$. Between p_c and 1, $P_\infty(p)$ increases monotonically.

More information can be obtained from the *mean cluster size distribution* $n_s(p)$ defined by

$$n_s(p) = \frac{\text{average number of clusters of size } s}{\text{total number of lattice sites}}. \quad (13.3)$$

For $p \geq p_c$, the spanning cluster is excluded from n_s . (For historical reasons, the *size* of a cluster refers to the *number* of sites in the cluster rather than to its spatial extent.) As an example, we see from Fig. 13.3a that $n_s(1) = 20$, $n_s(2) = 4$, $n_s(3) = 5$, and $n_s(7) = 1$ for $p = 0.2$ and is zero otherwise. Because $N \sum_s s n_s$ is the total number of occupied sites (N is the total number of lattice sites), and $N s n_s$ is the number of occupied sites in clusters of size s , the quantity

$$w_s = \frac{s n_s}{\sum_s s n_s} \quad (13.4)$$

is the probability that an occupied site chosen at random is part of an s -site cluster. Hence, the *mean cluster size* S is given by

$$S = \sum_s s w_s = \frac{\sum_s s^2 n_s}{\sum_s s n_s}. \quad (13.5)$$

The sum in (13.5) is over the finite clusters only. As an example, the weights corresponding to the clusters in Fig. 13.3a are $w_s(1) = 20/50$, $w_s(2) = 8/50$, $w_s(3) = 15/50$, and $w_s(7) = 7/50$, and hence $S = 130/50$.

Problem 13.5. Qualitative behavior of $n_s(p)$, $S(p)$, and $P_\infty(p)$

- a. Visually determine the cluster size distribution $n_s(p)$ for a square lattice with $L = 16$ and $p = 0.4$, $p = p_c$, and $p = 0.8$. Take $p_c = 0.5927$. One way to help you identify the clusters is to modify **Program site** so that you can use the FLOOD statement to show different clusters in different colors. Consider at least five configurations for each value of p and average $n_s(p)$ over the configurations. Because the lattice is finite, more consistent results can be obtained by discarding those configurations that have a spanning cluster for $p < p_c$ and those that do not have a spanning cluster for $p \geq p_c$. For each value of p , plot n_s as a function of s and describe the observed s -dependence. Does n_s decrease more rapidly with s for $p = p_c$ or for $p \neq p_c$?
- b. Use the same configurations considered in part (a) to compute the mean cluster size S as a function of p . Remember that for $p > p_c$, the spanning cluster is excluded.
- c. Similarly, compute $P_\infty(p)$ for $L = 16$, and for various values of $p \geq p_c$. Plot $P(p)$ as a function of p and discuss its qualitative behavior.
- d. Verify that $\sum_s s n_s(p) = p$ for $p < p_c$ and explain this relation. How is this relation modified for $p \geq p_c$?

13.3 Cluster Labeling

Your visual algorithm for determining the existence of a connected path probably is very sophisticated. Although using the FLOOD command in True BASIC helps us to automate the process for a single configuration, we need to average over many configurations to obtain quantitative results. Hence, we need to develop an algorithm that finds the clusters. In the following, we will find that this task is not easy. The difficulty is that the assignment of a site to a cluster is a *global* rather than a *local* property of the site.

We consider the multiple cluster labeling method of Hoshen and Kopelman. The algorithm can best be described by an example. Consider the configuration shown in Fig. 13.8. We define an array **site** to store the occupancy of the sites; an occupied site initially is assigned the value -1 and an unoccupied site is assigned the value 0 . We assign cluster labels to sites beginning at the lower left corner and continue from left to right. Because **site**(1, 1) is occupied, we assign to it cluster label 1. The next site is empty, and hence is not labeled. The next occupied site in the first row is **site**(3, 1). Because its left neighbor is unoccupied, we assign to it the next available cluster label, label 2. The assignment of cluster labels to the remainder of the row is straightforward, and we proceed to **site**(1, 2) of the second row. Because this site is occupied and its nearest neighbor in the preceding row is labeled 1, we assign label 1 to **site**(1, 2). We continue from left to right along the second row checking the occupancy of each site. If a site is occupied, we check the occupancy of its nearest neighbors in the previous row and column. If neither neighbor is occupied, we assign the next available cluster label. If only one nearest neighbor site is occupied, the site is assigned the label of its occupied neighbor. For example, **site**(2, 2) is assigned label 1 since its occupied neighbor, **site**(1, 2) has label 1.

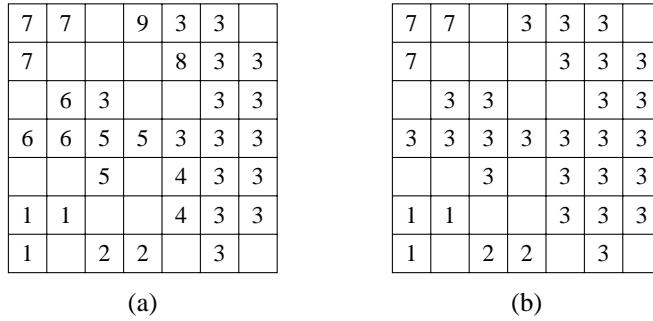


Figure 13.8: A percolation configuration on a square lattice with $L = 7$. Site coordinates are measured from the origin at the lower left corner (1,1). Part (a) shows the improper cluster labels initially assigned by `Program cluster`, a modified implementation of the Hoshen-Kopelman algorithm. Part (b) shows the proper cluster labels.

The difficulty arises when we come to an occupied site at which two clusters coalesce and cluster labels need to be reassigned. This case first occurs at `site(6,2)` — its two neighbors in the previous row and column have labels 3 and 4, respectively. We define the *proper* cluster label assignment at `site(6,2)` as the smaller of labels 3 and 4. Hence `site(6,2)` is assigned cluster label 3 and label 4 should be reassigned to label 3. It is inefficient to continually relabel the clusters, because there likely will be further reassessments. Hence, we delay the reassignment of cluster labels until the entire lattice is surveyed, and instead, keep track of the connections of the labels through a *label tree*. We introduce an array `np` that distinguishes proper and improper labels and provides their connections. Let us return to the configuration shown in Fig. 13.8 to explain the use of this array. Before we came to `site(6,2)`, labels 1 through 4 were proper labels and we set

$$np(1) = 1, \quad np(2) = 2, \quad np(3) = 3, \quad np(4) = 4. \quad (13.6)$$

At `site(6,2)` where labels 3 and 4 are linked, we set `np(4) = 3`. This reassignment of `np(4)` tells us that label 4 is improper, and the numerical value of `np(4)` tells us that label 4 is linked to label 3. Note that the argument `i` of `np(i)` always is greater than or equal to the value of `np(i)`.

This procedure is still not complete. What should we do when we come to a site with two previously labeled neighbors one or both of which are improper? For example, consider `site(5,4)` which has two occupied neighbors with labels 5 and 4. We might be tempted to assign `site(5,4)` the label 4 and set `np(5) = 4`. However, instead of assigning to a site the minimum label of its two neighbors, we should assign to it the minimum of the *proper* labels of the two neighboring sites. In addition, if the two neighboring sites have different proper labels, then we should set `np` of the maximum proper label equal to the minimum proper label. In this example, we have `np(5) = 3`.

The above version of the Hoshen-Kopelman cluster algorithm is implemented in the following subroutines. The arrays `site` and `np` are declared in a main program. After `SUB assign` is called, `site` contains the proper labels for each occupied site. The array `site` is given an extra empty column on the left ($x = 0$) and an extra empty row on the bottom ($y = 0$) so that the first column and row do not have to be treated differently. The following declaration in the main program for the arrays would be appropriate.

```
DIM site(0:64,0:64),np(0:1000)
```

A more efficient and different version of the Hoshen-Kopelman algorithm written in Fortran has been given by Stauffer and Aharony (see references). Although the Hoshen-Kopelman algorithm is the most efficient cluster identification method for two-dimensional systems, it is not obvious that this approach is the most efficient in higher dimensions. Can you think of another method for identifying the clusters?

Problem 13.6. Test of the cluster labeling algorithm

Incorporate `SUB assign` and its associated subroutines into `Program site`. You will need to add array `site` (in addition to array `rsite`) to the original subroutines of `Program site`. Use the following subroutine to show the improper and proper cluster labels and to help you check that the cluster labeling is being done correctly. Choose a particular configuration and explain how the Hoshen-Kopelman algorithm is implemented.

After the clusters are labeled, we can obtain a number of geometrical quantities of interest. Vertical spanning can be checked by seeing if there is a nonzero label in the bottom row equal to a nonzero label in the top row. The following subroutine checks for the existence of a vertical spanning cluster. Note that if `vspan <> 0`, there is a vertically spanning cluster with label `vspan`. Horizontal spanning can be determined by comparing the labels of the left and right columns.

The cluster distribution n_s can be found by computing the number of sites `m(i)` in cluster i , counting the number of clusters of size s , and normalizing the results by dividing by L^2 , the number of sites. The probability of an occupied site belonging to the spanning cluster P_∞ , can be determined by finding the label of the spanning cluster, `vspan`, and taking the ratio of `m(vspan)` to the total number of occupied sites. The mean cluster size S of the nonspanning clusters can be computed from the relation

$$S = \frac{\sum_i m^2(i)}{\sum_i m(i)}. \quad (13.7)$$

Convince yourself that (13.7) is equivalent to the definition (13.5) of S . Note that the spanning cluster is not included in the sums in (13.7).

In Problem 13.7 we apply the Hoshen-Kopelman cluster algorithm to a more systematic study of site percolation. In Section 13.4 we use a finite size scaling analysis to estimate the critical exponents.

**Problem 13.7.* Applications of the cluster labeling algorithm

- a. Compute $F(p) dp$, the probability of *first* spanning an $L \times L$ square lattice for p in the range p to $p + dp$. Write a subroutine to find when the spanning first occurs as p is increased. Do a minimum of 100 configurations for each value of L and plot $F(p)$ as a function of p . Consider $L = 4, 16$, and 32 . How does the shape of $F(p)$ change with increasing L ? At what value of p is $F(p) dp \approx 0.5$ for the various spanning criteria and for each value of L ? Call this value $p_c(L)$. How strongly does $p_c(L)$ depend on L for a given spanning criterion? How strongly does $p_c(L)$ depend on the spanning criterion for fixed L ?
- b. Compute P_∞ for $p = p_c, p = 0.65, p = 0.75$, and $p = 0.9$ for $L = 4, 16$, and 32 . Do a minimum of 100 configurations for each value of p . Use either the estimated value of $p_c(L)$ determined

- in part (a) or the known value $p_c = 0.5927$ (to four decimal places). What is the qualitative p -dependence of P_∞ ? Is $P_\infty(p = p_c)$ an increasing or decreasing function of L ? (Discard those configurations that do not have a spanning cluster.)
- Write a subroutine to compute $n_s(p)$. Consider $p = p_c$ and $p = p_c \pm 0.1$ for $L = 4, 16$, and 32 and average over at least ten configurations. Why is n_s a decreasing function of s ? Does n_s decrease more quickly for $p = p_c$ or for $p \neq p_c$?
 - Write a subroutine to compute the mean cluster size S for $p = p_c$ and $p = p_c \pm 0.1$ for $L = 4, 16$, and 32 . Average over at least ten configurations. What is the qualitative p -dependence of $S(p)$? How does $S(p = p_c)$ depend on L ? For $p < p_c$, discard the configurations that contain a spanning cluster and for $p > p_c$ discard the configurations that do not have a spanning cluster.

It is convenient to associate a characteristic linear dimension or *connectedness length* $\xi(p)$ with the clusters. One way to do so is to define the radius of gyration R_s of a single cluster of s particles as

$$R_s^2 = \frac{1}{s} \sum_{i=1}^s (\mathbf{r}_i - \bar{\mathbf{r}})^2, \quad (13.8)$$

where

$$\bar{\mathbf{r}} = \frac{1}{s} \sum_{i=1}^s \mathbf{r}_i, \quad (13.9)$$

and \mathbf{r}_i is the position of the i th site in the same cluster. The quantity $\bar{\mathbf{r}}$ is the familiar definition of the center of mass of the cluster. From (13.8), we see that R_s is the root mean square radius of the cluster measured from its center of mass. The connectedness length ξ can be defined as an average over the radii of gyration of all the finite clusters. The choice of the appropriate average is neither unique nor obvious. To find an expression for ξ , consider a site on a cluster of s sites. The site is connected to $s - 1$ other sites and the average square distance to these sites is R_s^2 (see Problem 13.8a). The probability that a site belongs to a cluster of site s is $w_s = sn_s$. These considerations suggest that one definition of ξ is

$$\xi^2 = \frac{\sum_s (s-1) w_s R_s^2}{\sum_s (s-1) w_s}. \quad (13.10)$$

To simplify the expression for ξ , we write s instead of $s - 1$ and let $w_s = sn_s$:

$$\xi^2 = \frac{\sum_s s^2 n_s R_s^2}{\sum_s s^2 n_s} \quad (13.11)$$

As before, the sum in (13.11) is over the nonspanning clusters only. The definition (13.11) and a simpler definition of $\xi(p)$ are explored in Problem 13.8.

*Problem 13.8. The connectedness length

- An alternative way of defining the radius of gyration of a cluster of s sites is as follows:

$$R_s^2 = \frac{1}{2s^2} \sum_{i,j} (\mathbf{r}_i - \mathbf{r}_j)^2. \quad (13.12)$$

- The sum (13.12) is over all pairs of particles in the cluster. What is the physical interpretation of (13.12)? Show that the form (13.12) is equivalent to (13.8). Which expression for R_s^2 is easier to compute?
- b. Write a subroutine to compute R_s for a nonspanning cluster of size s . Choose $L = 64$ and $p = 0.57$ and compute ξ using the definition (13.11). Average over at least five configurations. Does the largest nonspanning cluster make the dominant contribution to the sum?
 - c. Compute the p -dependence of either $\xi(p)$ using (13.11) or associate ξ with the radius of gyration of the largest nonspanning cluster. Choose $L = 64$ and consider at least 50 configurations for each value of p . Consider values of p in steps of 0.01 in the interval $[p_c - 0.05, p_c - 0.01]$ and $[p_c + 0.01, p_c + 0.05]$ with $p_c = 0.5927$. For $p < p_c$ discard those configurations that contain a spanning cluster, and for $p > p_c$ discard those configurations that do not have a spanning cluster. Plot $\xi(p)$ and discuss its qualitative dependence on p . Is $\xi(p)$ a monotonically increasing or decreasing function of p for $p < p_c$ and $p > p_c$?

13.4 Critical Exponents and Finite Size Scaling

We are familiar with different phases of matter from our everyday experience. The most familiar example is water which can exist as a vapor, liquid, or solid. It is well known that water changes from one phase to another at a well defined temperature and pressure, e.g., the transition from ice to liquid water occurs at 0°C at atmospheric pressure. Such a change of phase is an example of a *thermodynamic phase transition*. Most substances also exhibit a *critical point*; that is, beyond a particular temperature and pressure, it is no longer possible to distinguish between the liquid and gaseous phases and the phase boundary terminates.

Another example of a critical point occurs in magnetic systems at the Curie temperature T_c and zero magnetic field. We know that at low temperatures some substances exhibit ferromagnetism, a spontaneous magnetization in the absence of an external magnetic field. If we raise the temperature of a ferromagnet, the spontaneous magnetization decreases and vanishes continuously at a critical temperature T_c . For $T > T_c$, the system is a paramagnet. In Chapter ?? we use Monte Carlo methods to investigate the behavior of a magnetic system near the magnetic critical point.

In the following, we will find that the properties of the *geometrical* phase transition in the percolation problem are qualitatively similar to the properties of thermodynamic phase transitions. Hence, a discussion of the percolation phase transition also can serve as an introduction to thermodynamic phase transitions. We will see that in the vicinity of a phase transition, the qualitative behavior of the system is governed by the appearance of long-range correlations.

We have seen that the essential physics near the percolation threshold is associated with the existence of large but finite clusters. For example, for $p \neq p_c$, we found in Problem 13.7c that n_s decays rapidly with s . However for $p = p_c$, the s -dependence of n_s is qualitatively different, and n_s decreases much more slowly. This different behavior of n_s at $p = p_c$ is due to the presence of clusters of all length scales, e.g., the “infinite” spanning cluster and the finite clusters of all sizes. We also found (see Problem 13.8) that $\xi(p)$ is finite, and an increasing function of p for $p < p_c$ and a decreasing function of p for $p > p_c$ (see Fig. 13.9). Moreover, we know that $\xi(p = p_c)$ is approximately equal to L and hence diverges as $L \rightarrow \infty$. This qualitative behavior of $\xi(p)$ is consistent with our physical picture of the clusters, that is, as p approaches p_c , the probability that

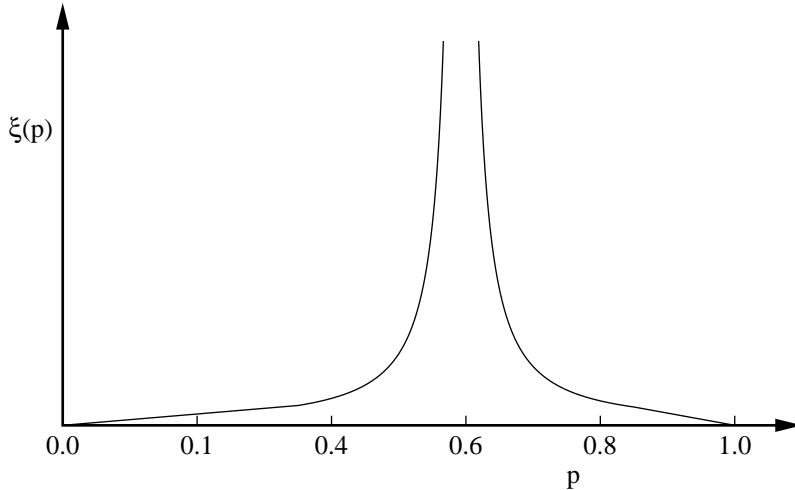


Figure 13.9: Qualitative p -dependence of the connectedness length $\xi(p)$. The divergent behavior of $\xi(p)$ in the critical region is characterized by the exponent ν (see (13.13)).

two occupied sites are in the same cluster increases. These qualitative considerations lead us to conjecture that in the limit $L \rightarrow \infty$, $\xi(p)$ grows rapidly in the *critical region*, $|p - p_c| \ll 1$. We can describe the divergence of $\xi(p)$ more quantitatively by introducing the *critical exponent* ν , defined by the relation

$$\xi(p) \sim |p - p_c|^{-\nu}. \quad (13.13)$$

Of course, there is no *a priori* reason why the divergence of $\xi(p)$ can be characterized by a simple power law. Note that ν is assumed to be the same above and below p_c .

How do the other quantities that we have considered behave in the critical region in the limit $L \rightarrow \infty$? According to the definition (13.2) of P_∞ , $P_\infty = 0$ for $p < p_c$ and is an increasing function of p for $p > p_c$. We conjecture that in the critical region, the increase of P_∞ with increasing p is characterized by an exponent β defined by the relation

$$P_\infty(p) \sim (p - p_c)^\beta. \quad (13.14)$$

Note that P_∞ is assumed to approach zero continuously from above p_c . We say that the percolation transition is a *continuous* phase transition. In the language of critical phenomena, P_∞ is an example of an *order parameter* of the system. An order parameter is a quantity that measures the “order” of a system, and is nonzero in the ordered phase and zero in the disordered phase. In the percolation context, we consider the phase with a spanning cluster to be ordered and the phase without a spanning cluster to be disordered.

The mean number of sites in the finite clusters, $S(p)$, also diverges in the critical region. Its critical behavior is written as

$$S(p) \sim |p - p_c|^{-\gamma}, \quad (13.15)$$

which defines the critical exponent γ . The common critical exponents for percolation are summarized in Table 13.1. For comparison, the analogous critical exponents of a magnetic critical point also are shown.

Quantity	Functional form	Exponent	$d = 2$	$d = 3$
Percolation				
order parameter	$P_\infty \sim (p - p_c)^\beta$	β	5/36	0.4
mean size of finite clusters	$S(p) \sim p - p_c ^{-\gamma}$	γ	43/18	1.8
connectedness length	$\xi(p) \sim p - p_c ^{-\nu}$	ν	4/3	0.9
cluster numbers	$n_s \sim s^{-\tau}$	$p = p_c$	τ	187/91
Ising model				
order parameter	$M(T) \sim (T_c - T)^\beta$	β	1/8	0.32
susceptibility	$\chi(T) \sim T - T_c ^{-\gamma}$	γ	7/4	1.24
correlation length	$\xi(T) \sim T - T_c ^{-\nu}$	ν	1	0.63

Table 13.1: Several of the critical exponents for the percolation and magnetism phase transitions in $d = 2$ and $d = 3$ dimensions. Ratios of integers correspond to known exact results. The critical exponents for the Ising model are discussed in Chapter ??.

Because we can simulate only finite lattices, a direct fit of the measured quantities ξ , P_∞ , and $S(p)$ to their assumed critical behavior for an infinite lattice would not yield good estimates for the corresponding exponents ν , β , and γ (see Problem 13.9b). The problem is that if p is close to p_c , the extent of the largest cluster becomes comparable to L , and the nature of the cluster distribution is affected by the finite size of the system. In contrast, for p far from p_c , $\xi(p)$ is small in comparison to L and the measured values of ξ , and hence the values of other physical quantities, are not appreciably affected by the finite size of the lattice. Hence for $p \ll p_c$ and $p \gg p_c$, the properties of the system are indistinguishable from the corresponding properties of a truly macroscopic system ($L \rightarrow \infty$). However, if p is close to p_c , $\xi(p)$ is comparable to L and the behavior of the system differs from that of an infinite system. In particular, a finite lattice cannot exhibit a true phase transition characterized by divergent physical quantities. Instead, ξ and S reach a finite maximum at $p = p_c(L)$.

The effects of the finite size of the system can be made more quantitative by the following argument. Consider for example, the critical behavior (13.14) of P_∞ . As long as ξ is much less than L , the power law behavior given by (13.14) is expected to hold. However, if ξ is comparable to L , ξ cannot change appreciably and (13.14) is no longer applicable. This qualitative change in the behavior of P_∞ and other physical quantities occurs for

$$\xi(p) \sim L \sim |p - p_c|^{-\nu}. \quad (13.16)$$

We invert (13.16) and write

$$|p - p_c| \sim L^{-1/\nu}. \quad (13.17)$$

The difference $|p - p_c|$ in (13.17) is the “distance” from the critical point at which “saturation” or finite size effects occur. Hence if ξ and L are approximately the same size, we can replace (13.14)

by the relation

$$P_\infty(p = p_c) \sim L^{-\beta/\nu} \quad (L \rightarrow \infty) \quad (13.18)$$

for the value of P_∞ at $p = p_c$ for a finite lattice. The relation (13.18) between P_∞ and L at $p = p_c$ is consistent with the fact that a phase transition, i.e., a singularity, is defined only for infinite systems.

One implication of (13.18) is that we can use it to determine the critical exponents. This method of analysis is known as *finite size scaling*. Suppose that we generate percolation configurations at $p = p_c$ for different values of L and analyze P_∞ as a function of L . If our values of L are sufficiently large, we can use the asymptotic relation (13.18) to estimate the ratio β/ν . A similar analysis can be used for $S(p)$ and other quantities of interest. We use this method in Problem 13.9.

Problem 13.9. Finite size scaling analysis of critical exponents

- a. Compute P_∞ at $p = p_c$ for at least 100 configurations. Consider $L = 10, 20, 40$, and 60 . Include in your average only those configurations that have a spanning cluster. Best results are obtained using the value of p_c for the infinite square lattice, $p_c \approx 0.5927$. Plot $\ln P_\infty$ versus $\ln L$, and estimate the ratio β/ν .
- b. Use finite size scaling arguments to determine the dependence of the mean cluster size S on L at $p = p_c$. Average S over the same configurations as considered in part (b). Remember that S is the mean number of sites in the nonspanning clusters.
- c. Analyze your data for the p -dependence of $S(p)$ obtained in Problem 13.5b for $L = 16$ and estimate the value of γ according to the assumed behavior given in (13.15). How does your estimate for γ compare with the answer that you obtained in part (b)?
- d. Find the mass (number of particles) M in the spanning cluster at $p = p_c$ as a function of L . Use the same configurations as in part (b). Determine an exponent from a plot of $\ln M$ versus $\ln L$. This exponent is called the fractal dimension of the cluster and is discussed in Chapter ??.

We found in Section 13.2 that the numerical value of the percolation threshold p_c depends on the symmetry and dimension of the lattice, e.g., $p_c \approx 0.5927$ for the square lattice and $p_c = 1/2$ for the triangular lattice. A remarkable feature of the power law dependencies summarized in Table 13.1 is that the values of the critical exponents do not depend on the symmetry of the lattice and are independent of the existence of the lattice itself, e.g., they are identical for the continuum percolation model discussed in Problem 13.4. Moreover, it is not necessary to distinguish between the exponents for site and bond percolation. In the vocabulary of critical phenomena, we say that site, bond, and continuum percolation all belong to the same *universality class* and that their critical exponents are identical.

Another important idea in critical phenomena is the existence of relations between the critical exponents. An example of such a *scaling law* is

$$2\beta + \gamma = \nu d, \quad (13.19)$$

where d is the spatial dimension of the lattice. The scaling law (13.19) indicates that the universality class depends on the spatial dimension. A more detailed discussion of finite size scaling and the scaling laws can be found in Chapter ?? and in the references.

13.5 The Renormalization Group

In Section 13.4, we studied the properties of various quantities on different length scales to determine the values of the critical exponents. The idea of examining physical quantities near the critical point on different length scales can be extended beyond finite size scaling and is the basis of the *renormalization group* method, probably the most important new method developed in theoretical physics during the past twenty-five years. Kenneth Wilson was honored in 1981 with the Nobel prize in physics for his contributions to the development of the renormalization group method. Although the method was first applied to thermodynamic phase transitions, it is simpler to introduce the method in the context of the percolation transition. We will find that the renormalization group method yields the critical exponents directly, and in combination with Monte Carlo methods, it is frequently more powerful than Monte Carlo methods alone.

To introduce the method, consider a photograph of a percolation configuration generated at $p = p_0 < p_c$. If we view the photograph (or screen) from further and further distances, what would we see? Convince yourself that when you are far away from the photograph, you cannot distinguish occupied sites that are adjacent to each other and you cannot observe single site clusters. In addition, branches emanating from larger clusters and narrow bridges connecting large “blobs” are lost in your distant view of the photograph. Hence for $p_0 < p_c$, the distant photograph looks like a percolation configuration generated at a value of $p = p_1$ less than p_0 . In addition, the connectedness length $\xi(p_1)$ of the remaining clusters is smaller than $\xi(p_0)$. If we move even further away from the photograph, the new clusters look even smaller with a value of $p = p_2$ less than p_1 . Eventually we will not be able to distinguish any clusters and the photograph will appear as if it were at the trivial *fixed point* $p = 0$.

What would we observe as we go away from the photograph for $p_0 > p_c$? We can use the same reasoning to deduce that we would see only small regions of unoccupied sites. As we move further away from the photograph, these spaces become less discernible and the configuration looks as though a larger percentage of the lattice were occupied. Hence, the photograph will look like a configuration generated at a value of $p = p_1$ greater than p_0 with $\xi(p_1) < \xi(p_0)$. As we move further and further away from the photograph, it will eventually appear to be at the other trivial fixed point $p = 1$.

What would we observe at $p_0 = p_c$? We know that at the percolation threshold, all length scales are present and it does not matter which length scale we use to observe the system. Hence, the photograph appears the same (although smaller overall) regardless of the distance at which we observe it. In this sense, p_c is a special, *nontrivial* fixed point.

We now consider a way of using a computer to change the configurations in a way that is similar to moving away from the photograph. Consider a square lattice that is partitioned into *cells* or *blocks* that cover the lattice (see Fig. 13.10). If we view the lattice from the perspective in which the sites in a cell merge to become a new supersite or renormalized site, then the new lattice has the same symmetry as the original lattice. However, the replacement of cells by the new sites has changed the length scale — all distances are now smaller by a factor of b , where b is the linear dimension of the cell. Hence, the effect of a “renormalization” is to replace each group of sites with a single renormalized site and to rescale the connectedness length for the renormalized lattice by a factor of b .

How can we decide whether the renormalized site is occupied or not? Because we want to

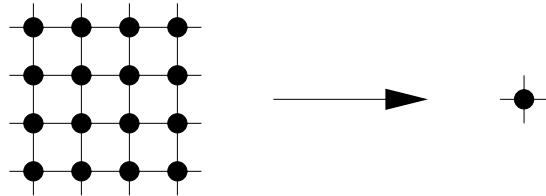


Figure 13.10: An example of a $b = 4$ cell used on the square lattice. The cell contains b^2 sites which are rescaled to a single supersite after a renormalization group transformation.

preserve the main features of the original lattice and hence its connectedness (and its symmetry), we assume that a renormalized site is occupied if the original group of sites spans the cell. We adopt the vertical spanning criterion for convenience. The effect of performing a renormalization transformation on typical percolation configurations for p above and below p_c is illustrated in Fig. 13.11 and Fig. 13.12 respectively. In both cases, the effect of the successive transformations is to move the system away from p_c . We see that for $p = 0.7$, the effect of the transformations is to drive the system toward $p = 1$. For $p = 0.5$, the trend is to drive the system toward $p = 0$. As we discuss in the following, we can associate p_c with an unstable fixed point of the renormalization transformation. Of course, because we began with a finite lattice, we cannot continue the renormalization transformation indefinitely.

Program rg implements a visual interpretation of the renormalization group. The program divides the screen into four windows with the original lattice in the first window and three renormalized lattices in windows 2 through 4. In **Program site** we represented an occupied site at lattice point x, y as a filled circle of unit diameter centered about the point (x, y) . In contrast, **Program rg** represents an occupied site at x, y as a filled box whose lower left corner is at $x - 1, y - 1$.

Problem 13.10. Visual renormalization group

Use **Program rg** with $L = 32$ to estimate the value of the percolation threshold. For example, confirm that for small p , e.g., $p = 0.4$, the renormalized lattice almost always renormalizes to a nonspanning cluster. What happens for $p = 0.8$? How can you use the properties of the renormalized lattices to estimate p_c ?

Although a visual implementation of the renormalization group allows us to estimate p_c , it does not allow us to estimate the critical exponents. In the following, we present a renormalization group method that allows us to obtain p_c and the critical exponent ν associated with the connectedness length. This analysis follows closely the method presented by Reynolds et al. (see references).

The implementation of a renormalization group method consists of two parts: (i) an average over the underlying variables together with a specification of the variables that determine the state of the renormalized configuration, and (ii) a parameterization of the renormalized configuration in terms of the original parameters and possibly others. We adopt the same average as before, i.e., we replace the b^d sites within a cell of linear dimension b by a single site that represents whether or not the original lattice sites span the cell. The second step is to determine which parameters specify the new configuration after the averaging. We make the simple approximation that each cell is independent of all the other cells and is characterized only by the probability p' that the cell

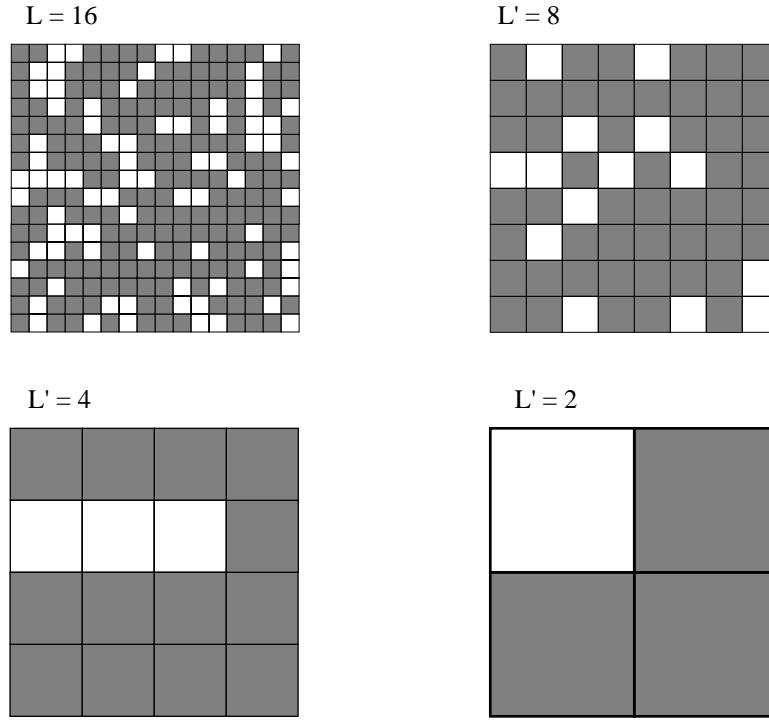


Figure 13.11: A percolation configuration generated at $p = 0.7$. The original configuration has been renormalized three times by transforming cells of four sites into one new supersite. What would be the effect of an additional transformation?

is occupied. The renormalization transformation between p and p' reflects the fact that the basic physics of percolation is connectedness, because we define a cell to be occupied only if it contains a set of sites that span the cell. If the sites are occupied with probability p , then the cells are occupied with probability p' , where p' is given by a *renormalization transformation* or a *recursion relation* of the form

$$p' = R(p). \quad (13.20)$$

The quantity $R(p)$ is the total probability that the sites form a spanning path.

An example will make the formal relation (13.20) more clear. In Fig. 13.13, we show the seven vertically spanning site configurations for a $b = 2$ cell. The probability p' that the renormalized site is occupied is given by the sum of the probabilities of all spanning configurations:

$$p' = R(p) = p^4 + 4p^3(1 - p) + 2p^2(1 - p)^2. \quad (13.21)$$

In general, the probability p' of the occupied renormalized sites is different than the occupation probability p of the original sites. For example, suppose that we begin with $p = p_0 = 0.5$. After a single renormalization transformation, the value of p' obtained from (13.21) is $p_1 = p' = R(p_0) =$

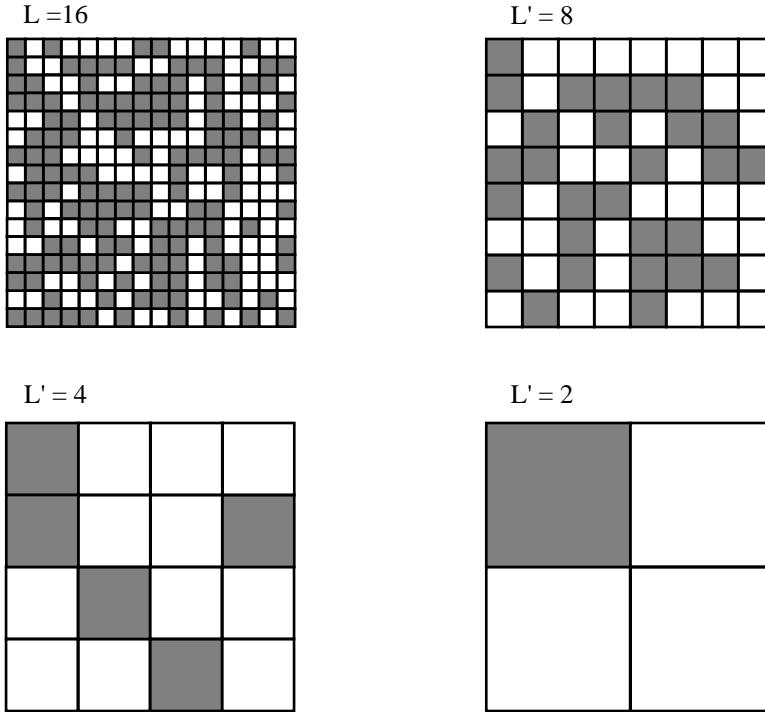


Figure 13.12: A percolation configuration generated at $p = 0.5$. The original configuration has been renormalized three times by transforming blocks of four sites into one new site. What would be the effect of an additional transformation?

$0.5) = 0.44$. If we perform a second renormalization transformation, we have $p_2 = R(p_1) = 0.35$. It is easy to see that further transformations drive the system to the fixed point $p = 0$. Similarly, if we begin with $p = p_0 = 0.7$, we find that successive transformations drive the system to the fixed point $p = 1$. This behavior is qualitatively similar to what we observed in the visual renormalization group.

To find the nontrivial fixed point associated with the critical threshold p_c , we need to find the special value of p such that

$$p^* = R(p^*). \quad (13.22)$$

For the recursion relation (13.21), we find that the solution of the fourth degree equation for p^* yields the two trivial fixed points, $p^* = 0$ and $p^* = 1$, and the nontrivial fixed point $p^* = 0.61804$ which we associate with p_c . This calculated value of p^* for $b = 2$ should be compared with the estimate $p_c = 0.5927$.

To calculate the critical exponent ν , we recall that all lengths are reduced on the renormalized lattice by a factor of b in comparison to the lengths in the original system. Hence the connectedness

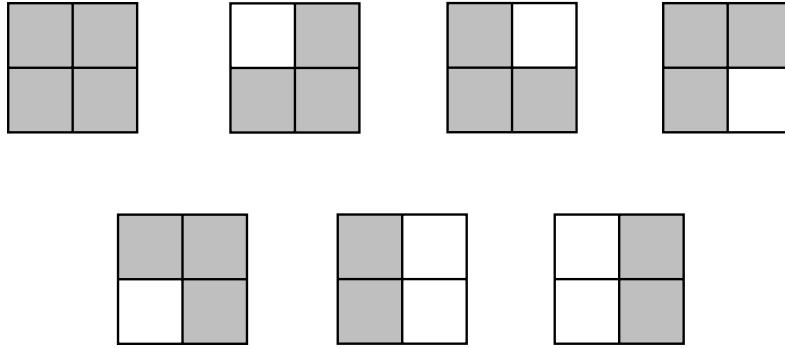


Figure 13.13: The seven (vertically) spanning configurations on a $b = 2$ cell.

length transforms as

$$\xi' = \xi/b. \quad (13.23)$$

Because $\xi(p) = \text{const}|p - p_c|^{-\nu}$ for $p \sim p_c$, we have

$$|p' - p^*|^{-\nu} = b^{-1}|p - p^*|^{-\nu}, \quad (13.24)$$

where we have identified p_c with p^* . To find the relation between p' and p near p_c , we expand the renormalization transformation (13.20) in a Taylor series about p^* and obtain to first order in $(p - p^*)$:

$$p' - p^* = R(p) - R(p^*) \approx \lambda(p - p^*), \quad (13.25)$$

where

$$\lambda = \frac{dR(p = p^*)}{dp}. \quad (13.26)$$

We need to do a little algebra to obtain an explicit expression for ν . We first raise both sides of (13.25) to the ν th power and write

$$|p' - p^*|^\nu = \lambda^\nu(p - p^*)^\nu. \quad (13.27)$$

We then compare (13.27) and (13.24) and obtain

$$b = \lambda^\nu. \quad (13.28)$$

Finally, we take the logarithm of both sides of (13.28) and obtain the desired relation for the critical exponent ν :

$$\nu = \frac{\log b}{\log \lambda}. \quad (13.29)$$

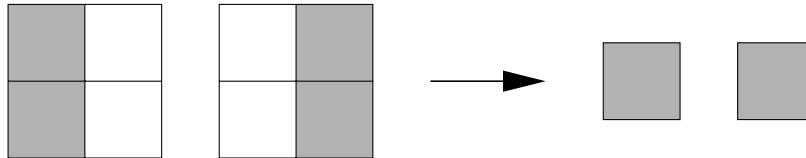


Figure 13.14: Example of the interface problem between cells. The two cells are not connected at the original site level but are connected at the cell level.

As an example, let us calculate λ for a square lattice with $b = 2$. We write (13.21) in the form $R(p) = -p^4 + 2p^2$. The derivative of $R(p)$ with respect to p yields $\lambda = 4p(1-p^2) = 1.5279$ at $p = p^* = 0.61804$. We then use the relation (13.29) to obtain

$$\nu = \frac{\log 2}{\log 1.5279} = 1.635\dots \quad (13.30)$$

A comparison of (13.30) with the exact result $\nu = 4/3$ (see Table 13.1) in two dimensions shows remarkable agreement for such a simple calculation. (What would we be able to conclude if we were to measure $\xi(p)$ directly on a 2×2 lattice?) However, the accuracy of our calculation of ν is not known. What is the nature of our approximations? Our major assumption has been that the occupancy of each cell is independent of all other cells. This assumption is correct for the original sites, but after one renormalization, we lose some of the original connecting paths and gain connecting paths that are not present in the original lattice. An example of this interface problem is shown in Fig. 13.14. Because this surface effect becomes less probable with increasing cell size, one way to improve the renormalization group calculation is to consider larger cells. We consider a $b = 3$ calculation in Problem 13.11d. In Project 13.14 we combine the renormalization group method with a Monte Carlo approach to treat still larger cells.

Problem 13.11. Renormalization group method for small cells

- Enumerate the spanning configurations for a $b = 2$ cell assuming that a cell is occupied if a spanning path exists in either the vertical or the horizontal directions. Obtain the recursion relation and solve for the fixed point p^* . Although you could use a root finding algorithm to solve for p^* , it is easy to use trial and error to find the value of p such that $R(p) - p$ is zero. Or you can plot the function $R(p) - p$ versus p and find the value of p at which $R(p) - p$ crosses the horizontal axis. How do p^* and ν compare to their values using the vertical spanning criterion?
- Repeat the simple renormalization group calculation in (a) using the criterion that a cell is occupied only if a spanning path exists in both directions.
- The association of p_c with p^* is not the only possible one. Two alternatives involve the derivative $R'(p) = dR/dp$. For example, we could let $p_c = \bar{p} = \int_0^1 p R'(p) dp$. Alternatively, we could choose $p_c = p_{\max}$, where p_{\max} is the value of p at which $R'(p)$ has its maximum value. Compute p_c using these two alternative definitions and the various spanning criteria. In the limit of large cells, all three definitions should lead to the same values of p_c .
- Enumerate the possible spanning configurations of a $b = 3$ cell, assuming that a cell is occupied if a cluster spans the cell vertically. Determine the probability of each configuration, and verify

that the renormalization transformation $R(p) = p^9 + 9p^8q + 36p^7q^2 + 67p^6q^3 + 59p^5q^4 + 22p^4q^5 + 3p^3q^6$. It is possible to do the exact enumeration by hand. Solve the recursion relation (13.22) for p^* . Use this value of p^* to find the slope λ and the exponent ν . Then assume a cell is occupied if a cluster spans the cell both vertically and horizontally and obtain $R(p)$. Determine $p^*(b=3)$ and $\nu(b=3)$ for the two spanning criteria. Are your results for p^* and ν closer to their known values than for $b=2$ for the same spanning criteria?

Problem 13.12. Renormalization group method for triangular lattice

- There are some difficulties with the above renormalization group method in the infinite cell limit, if a cell is said to span when there is a path in one fixed direction (see Ziff). This problem is absent for the triangular lattice. For this symmetry a cell can be formed by grouping three sites that form a triangle into one renormalized site. The only reasonable spanning criterion is that the cell spans if any two sites are occupied. Verify that $R(p) = p^3 + 3p^2(1-p)$ and find $p_c = p^*$. How does p^* compare to the exact result $p_c = 1/2$?
- Calculate the critical exponent ν and compare its value with the exact result. Explain why b is given by $b^2 = 3$. Give a qualitative argument why the renormalization group argument might work better for small cells on a triangular lattice than on a square lattice.

It is possible to improve our renormalization group results for p_c and ν by enumerating the spanning clusters for larger b . However, because the 2^{b^2} possible configurations for a $b \times b$ cell increase rapidly with b , exact enumeration is not practical for $b > 7$, and we must use Monte Carlo methods if we wish to proceed further. Two Monte Carlo approaches are discussed in Project 13.14. The combination of methods, Monte Carlo and renormalization group (MCRG), provides a powerful tool for obtaining information on phase transitions and other properties of materials.

13.6 Projects

We have seen that the percolation problem illustrates many of the important ideas in critical phenomena. In later chapters we apply similar ideas and approaches to thermal systems. The following projects require larger systems and more computer resources than the problems in this chapter, but they are not much more difficult conceptually. More ideas for projects can be obtained from the references.

Project 13.13. Cell-to-cell renormalization group method

In Section 13.5 we discussed the cell-to-site renormalization group transformation for a system of cells of linear dimension b . An alternative transformation is to go from cells of linear dimension b_1 to cells of linear dimension b_2 . For such a “cell-to-cell” transformation, the rescaling length b_1/b_2 can be made close to unity. Many errors in a cell-to-cell renormalization group transformation cancel, resulting in a transformation that is more accurate in the limit in which the change in length scale is infinitesimal. We can use the fact that the connectedness lengths of the two systems are related by $\xi(p_2) = (b_1/b_2)^{-1}\xi(p_1)$ to derive the relation

$$\nu = \frac{\ln b_1/b_2}{\ln \lambda_1/\lambda_2}, \quad (13.31)$$

where $\lambda_i = dR(p^*, b_i)/dp$ is evaluated at the solution to the fixed point equation, $R(b_2, p^*) = R(b_1, p^*)$. Note that (13.31) reduces to (13.29) for $b_2 = 1$. Use the results you found in Problem 13.11d for one of the spanning criteria to estimate ν from a $b_1 = 3$ to $b_2 = 2$ transformation. Then consider larger values of b_2 and b_1 .

Project 13.14. Monte Carlo renormalization group

- One way to estimate $R(p)$, the total probability of all the spanning clusters, can be understood by writing $R(p)$ in the form

$$R(p) = \sum_{n=1}^N \binom{N}{n} p^n q^{(N-n)} S(n), \quad (13.32)$$

where $N = b^2$. The binomial coefficient $\binom{N}{n} = N!/((N-n)!n!)$ represents the number of possible configurations of n occupied sites and $N-n$ empty sites. The quantity $S(n)$ is the probability that a random configuration of n occupied sites spans the cell. A comparison of (13.21) and (13.32) shows that for $b = 2$ and the vertical spanning criterion, $S(1) = 0$, $S(2) = 2/6$, $S(3) = 1$, and $S(4) = 1$. What are the values of $S(n)$ for $b = 3$?

We can estimate the probability $S(n)$ by straightforward Monte Carlo methods. One way to sample $S(n)$ is to add a particle at random to an unoccupied site and check if a spanning path exists. If a spanning path does not exist, add another particle at random to a previously unoccupied site. If a spanning path exists after s particles are added, then let $S(n) = S(n) + 1$ for $n \geq s$ and generate a new configuration. After a reasonable number of configurations, the results for $S(n)$ can be normalized. Of course, this procedure can be made more efficient by checking for a spanning cluster only after the total number of particles added is near $s \sim p^*N$ and by checking for spanning after adding several particles.

Write a Monte Carlo program to sample $S(n)$. (Hint: store the location of the unoccupied sites in a separate array.) To check your program, first sample $S(n)$ for $b = 2$ and $b = 3$ and compare your results to the exact results for $S(n)$. Consider larger values of b and determine $S(n)$ for $b = 5, 8, 16$, and 32 . For $b \geq 16$, the total probability $R(p)$ can be found by using (13.32) and the Gaussian approximation for the probability of a configuration of n occupied sites:

$$P_N(n) = \binom{N}{n} p^n q^{(N-n)} \approx (2\pi Npq)^{-\frac{1}{2}} e^{-(n-pN)^2/2Npq}. \quad (13.33)$$

Because $P_N(n)$ is sharply peaked for large b , it is necessary to sample $S(n)$ only near $n = p^*N$. This method has been investigated by Hu (see references).

- In part (a) the number of particles rather than the occupation probability p was varied. Another Monte Carlo procedure is to vary p and sample $F(p) dp$, the probability of *first* spanning a $b \times b$ cell in the range p to $p+dp$. Because the renormalization group transformation defines p' as the *total* probability of spanning at p , p' can be interpreted as the *cumulative distribution function* and is related to $F(p)$ by

$$p' = R(p) = \int_0^p F(p) dp. \quad (13.34)$$

The sampling of $F(p)$ for finite width bins Δp implies that the integral in (13.34) reduces to a sum. Because $\lambda = dR(p = p^*)/dp$, we have $\lambda = F(p^*)$. The simplest way to estimate λ is by setting $\lambda = F(p_{\max})$, where p_{\max} is the value of p for which $F(p)$ is a maximum. Determine $p_c(b)$ and $\nu(b)$ for $b = 5, 8, 16$, and 32 . How do your results compare with those found in part (refproj:13/mcrgproba)? Which method yields smaller error estimates for p_c and ν ?

- c. It is possible to extrapolate the results for the successive estimates $p_c(b)$ and $\nu(b)$ to the limit $b \rightarrow \infty$. Finite size scaling arguments (cf. Stauffer and Aharony) suggest that

$$\nu(b) \approx \nu - c_1/\ln b \quad (13.35a)$$

and

$$p^*(b) \approx p_c - a_1 b^{-1/\nu} \quad (13.35b)$$

for b sufficiently large. The relation (13.35a) suggests that the sequence $\nu(b)$ should be plotted as a function of $1/\ln b$ and the extrapolated result should be a straight line with an intercept of ν . The quantities a_1 and c_1 in (13.35) are fitting parameters. The relation (13.35b) suggests that we should plot $p_c(b)$ versus $b^{-1/\nu}$ using the value of ν found from (13.35a). How sensitively does your result for p_c depend on the assumed value of ν ? It is necessary to consider cells on the order of $b = 500$, and to do a more sophisticated analysis of $\nu(b)$ and $p^*(b)$, to obtain extrapolated values that agree to four places with the exact value $\nu = 4/3$ and the estimate $p_c = 0.5927$.

Project 13.15. Percolation in three dimensions

- a. The value of p_c for site percolation on the simple cubic lattice is approximately 0.311. Write a program using the Hoshen-Kopelman cluster labeling method to verify this value. Compute ϕ_c , the volume fraction occupied at p_c , if a sphere with a diameter equal to the lattice spacing is placed on each occupied site.
- b. Consider continuum percolation in three dimensions where spheres of unit diameter are placed at random in a cubical box of linear dimension L . Two spheres that overlap are in the same cluster. As each sphere is added to the box, determine if the sphere overlaps with any other sphere in the box. If it does not, then the sphere constitutes a new cluster. If it does, then the sphere adopts the cluster label of the sphere with which it overlaps. If it overlaps with spheres of different cluster labels, then it is necessary to either relabel the clusters or to use the Hoshen-Kopelman algorithm to determine the proper label and generate a label tree. The volume fraction occupied by the spheres is given by

$$\phi = 1 - e^{-\rho 4\pi r^3/3}, \quad (13.36)$$

where ρ is the number density of the spheres, and r is their radius. Write a program to simulate continuum percolation in three dimensions and find the percolation threshold ρ_c . Use the Monte Carlo procedure discussed in Problem 13.4 to estimate ϕ_c and compare its value with the value obtained using (13.36). How does ϕ_c for continuum percolation compare with the value of ϕ_c found for site percolation in part (a)? Which do you expect to be larger and why?

- c. In the Swiss cheese model in three dimensions, we are concerned with the percolation of the space between the spheres. This model is appropriate for porous rock with the spheres representing

solid material and the space between the spheres representing the pores. Superimpose a regular grid with lattice spacing equal to $0.1r$ on the system, where r is the radius of the spheres. If a point on the grid is not within any sphere, it is “occupied.” The use of the grid allows us to determine the connectivity between different regions of the pore space. Use your cluster labeling routine from part (a) to label the clusters, and determine ϕ_c , the volume fraction occupied by the pores at threshold. You might be surprised to find that ϕ_c is relatively small. If time permits, use a finer grid and repeat the calculation to improve the accuracy of your results.

- d. Use finite size scaling to estimate the critical percolation exponents for the three models presented in parts (a)–(c). Are they the same within the accuracy of your calculation?

Project 13.16. Conductivity in a random resistor network

- a. An important critical exponent for percolation is the conductivity exponent t defined by

$$\sigma \sim (p - p_c)^t, \quad (13.37)$$

where σ is the conductance (or inverse resistance) per unit length in two dimensions. Consider bond percolation on a square lattice where each occupied bond between two neighboring sites is a resistor of unit resistance. Unoccupied bonds have infinite resistance. Because the total current into any node must equal zero by Kirchhoff’s law, the voltage at any site (node) is equal to the average of the voltages of all nearest neighbor sites connected by resistors (occupied bonds). Since this relation for the voltage is the same as the algorithm for solving Laplace’s equation on a lattice, the voltage at each site can be computed using a relaxation method discussed in Chapter ???. To compute the conductivity for a given $L \times L$ resistor network, we fix the voltage $V = 0$ at sites for which $x = 0$ and fix $V = 1$ at sites for which $x = L + 1$. In the y direction we use periodic boundary conditions. We then compute the voltage at all sites using the relaxation method. The current through each resistor connected to a site at $x = 0$ is simply $I = \Delta V/R = (V - 0)/1 = V$. The conductivity is the sum of the currents through all the resistors connected to $x = 0$ divided by L . In a similar way, the conductivity can be computed from the resistors attached to the $x = L + 1$ boundary. Write a program to implement the relaxation method for the conductivity of a random resistor network on a square lattice. An indirect, but easier way of computing the conductivity, is considered in Problem ??.

- b. The bond percolation threshold on a square lattice is $p_c = 0.5$. Use your program to compute the conductivity for a $L = 30$ square lattice. Average over at least ten spanning configurations for $p = 0.51, 0.52$, and 0.53 . Note that you can eliminate all bonds that are not part of the spanning cluster and all occupied bonds connected to only one other occupied bond. Why? If possible, consider more values of p . Estimate the critical exponent t defined in (13.37).
- c. Fix p at $p = p_c = 1/2$ and use finite size scaling to estimate the conductivity exponent t .
- d. Use larger lattices and the multigrid method (see Project ??) to improve your results. If you have sufficient computing resources, compute t for a simple cubic lattice for which $p_c \approx 0.247$. (In two dimensions t is the same for lattice and continuum percolation. However, in three dimensions t can be different.)

References and Suggestions for Further Reading

- Joan Adler, “Series expansions,” *Computers in Physics* **8**, 287 (1994). The critical exponents and the value of p_c also can be determined by doing exact enumeration.
- I. Balberg, “Recent developments in continuum percolation,” *Phil. Mag.* **56**, 991 (1987). An earlier paper on continuum percolation is by Edward T. Gawlinski and H. Eugene Stanley “Continuum percolation in two dimensions: Monte Carlo tests of scaling and universality for non-interacting discs,” *J. Phys. A: Math. Gen.* **14**, L291 (1981). These workers divide the system into cells and use the Poisson distribution to place the appropriate number of disks in each cell.
- Armin Bunde and Shlomo Havlin, editors, *Fractals and Disordered Systems*, Springer-Verlag (1991). Chapter 2 by the editors is on percolation.
- C. Domb, E. Stoll, and T. Schneider, “Percolation clusters,” *Contemp. Phys.* **21**, 577 (1980). This review paper discusses the nature of the percolation transition using illustrations from a film of a Monte Carlo simulation of a percolation process.
- J. W. Essam, “Percolation theory,” *Reports on Progress in Physics* **53**, 833 (1980). A mathematically oriented review paper.
- Jens Feder, *Fractals*, Plenum Press (1988). See Chapter 7 on percolation. We discuss the fractal properties of the spanning cluster at the percolation threshold in Chapter ??.
- J. P. Fitzpatrick, R. B. Malt, and F. Spaepen, “Percolation theory of the conductivity of random close-packed mixtures of hard spheres,” *Phys. Lett. A* **47**, 207 (1974). The authors describe a demonstration experiment done in a first year physics course at Harvard.
- J. Hoshen and R. Kopelman, “Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm,” *Phys. Rev. B* **14**, 3438 (1976). The original paper on an efficient cluster labeling algorithm.
- Chin-Kun Hu, Chi-Ning Chen, and F. Y. Wu, “Histogram Monte Carlo position-space renormalization group: applications to site percolation,” preprint. The authors use a histogram Monte Carlo method which is similar to the method discussed in Project 13.14. A similar Monte Carlo method also was used by M. Ahsan Khan, Harvey Gould, and J. Chalupa, “Monte Carlo renormalization group study of bootstrap percolation,” *J. Phys. C* **18**, L223 (1985).
- Ramit Mehr, Tal Grossman, N. Kristianpoller, and Yuval Gefen, “Simple percolation experiment in two dimensions,” *Am. J. Phys.* **54**, 271 (1986). A simple experiment for an undergraduate physics laboratory is proposed.
- D. C. Rapaport, “Percolation on large lattices,” *Phil. Mag.* **56**, 1027 (1987). See also D. C. Rapaport, “Cluster size distribution at criticality,” *J. Stat. Phys.* **66**, 679 (1992). The author generates many independent sublattices and then combines them to study percolation on a $640\ 000^2$ lattice.

- Peter J. Reynolds, H. Eugene Stanley, and W. Klein, “Large-cell Monte Carlo renormalization group for percolation,” *Phys. Rev.* **B21**, 1223 (1980). An especially clearly written research paper. Our discussion on the renormalization group in Section 13.5 is based upon this paper.
- Muhammad Sahimi, *Applications of Percolation Theory*, Taylor & Francis (1994). The author is a chemical engineer, and the emphasis is on modeling various phenomena in disordered media.
- Muhammad Sahimi and Hossein Rassamdana, “On position-space renormalization group approach to percolation,” *J. Stat. Phys.* **78**, 1157 (1995).
- Dietrich Stauffer and Amnon Aharony, *Introduction to Percolation Theory*, second edition, Taylor & Francis (1994). A delightful book by two of the leading workers in the field. An efficient Fortran implementation of the Hoshen-Kopelman algorithm is given in Appendix A.3.
- D. Stauffer, “Percolation clusters as teaching aid for Monte Carlo simulation and critical exponents,” *Am. J. Phys.* **45**, 1001 (1977); D. Stauffer, “Scaling theory of percolation clusters,” *Physics Reports* **54**, 1 (1979).
- B. P. Watson and P. L. Leath, “Conductivity in the two-dimensional-site percolation problem,” *Phys. Rev.* **B9**, 4893 (1974). A research paper on the conductivity of chicken wire.
- Kenneth G. Wilson, “Problems in physics with many scales of length,” *Sci. Am.* **241**, 158 (1979). An accessible article on the renormalization group method and its applications in particle and condensed matter physics. See also K. G. Wilson, “The renormalization group and critical phenomena,” *Rev. Mod. Phys.* **55**, 583 (1983). The latter article is the text of Wilson’s lecture on the occasion of the presentation of the 1982 Nobel Prize in Physics. In this lecture he claims that he “... found it very helpful to demand that a correctly formulated field theory be soluble by computer, the same way an ordinary differential equation can be solved on a computer ...”
- W. Xia and M. F. Thorpe, “Percolation properties of random ellipses,” *Phys. Rev. A* **38**, 2650 (1988). The authors consider continuum percolation of elliptical shapes, and show that $\phi = e^{-A\rho}$, where A is the area of the object, and ρ is the number density.
- Richard Zallen, *The Physics of Amorphous Solids*, Wiley-Interscience (1983). Chapter 4 discusses many of the applications of percolation concepts to realistic systems.
- Robert M. Ziff, “Spanning probability in 2D percolation,” *Phys. Rev. Lett.* **69**, 2670 (1992). The author finds $p_c = 0.592\,7460 \pm 0.000\,0005$ for a square lattice.

Chapter 14

Fractals

We introduce the concept of fractal dimension and discuss several processes that generate fractal objects.

14.1 The Fractal Dimension

One of the more interesting geometrical properties of objects is their shape. As an example, we show in Figure 14.1 a spanning cluster generated at the percolation threshold. Although the visual description of such a cluster is subjective, such a cluster can be described as ramified, airy, tenuous, and stringy, and would not be described as compact or space-filling.

In recent years a new *fractal* geometry has been developed by Mandelbrot and others to describe the characteristics of ramified objects. One quantitative measure of the structure of these objects is their *fractal dimension* D . To define D , we first review some simple ideas of dimension in ordinary Euclidean geometry. Consider a circular or spherical object of mass M and radius R . If the radius of the object is increased from R to $2R$, the mass of the object is increased by a factor of 2^2 if the object is circular, or by 2^3 if the object is spherical. We can express this relation between mass and length as

$$M(R) \sim R^D, \quad (\text{mass dimension}) \quad (14.1)$$

where D is the dimension of the object. Equation (14.1) implies that if the linear dimensions of an object are increased by a factor of b while preserving its shape, then the mass of the object is increased by b^D . This mass-length scaling relation is closely related to our intuitive understanding of dimension. Note that if the dimension of the object, D , and the dimension of the Euclidean space in which the object is embedded, d , are identical, then the mass density $\rho = M/R^d$ scales as

$$\rho(R) \propto M(R)/R^d \sim R^0. \quad (14.2)$$

An example of a two-dimensional object is shown in Figure 14.2. An object whose mass-length relation satisfies (14.1) with $D = d$ is said to be *compact*.

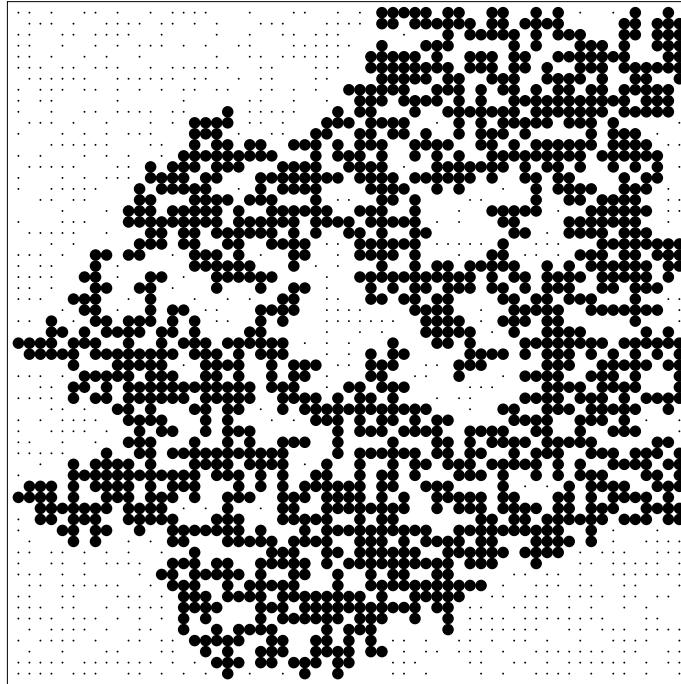


Figure 14.1: Example of a percolation cluster generated at $p = 0.5927$ on a $L = 61$ square lattice. Occupied sites that are not part of the spanning cluster are shown as points; unoccupied sites are not shown.

Equation (14.1) can be used to define the fractal dimension. We denote objects as fractals if they satisfy (14.1) with a value of D different from the spatial dimension d . Note that if an object satisfies (14.1) with $D < d$, its density is not the same for all R , but scales as

$$\rho(R) \propto M/R^d \sim R^{D-d}. \quad (14.3)$$

Because $D < d$, a fractal object becomes less dense at larger length scales. The scale dependence of the density is a quantitative measure of the ramified or stringy nature of fractal objects. That is, one characteristic of fractal objects is that they have holes of all sizes.

The percolation cluster shown in Figure 14.1 is an example of a *random* or statistical fractal because the mass-length relation (14.1) is satisfied only on the average, that is, only if the quantity $M(R)$ is averaged over many different origins in a given cluster and over many clusters.

In physical systems, the relation (14.1) does not extend over all length scales, but is bounded by both upper and lower cut-off lengths. For example, a lower cut-off length is provided by a microscopic distance such as a lattice spacing or the mean distance between the constituents of the object. In computer simulations a maximum length usually is provided by the finite system size. The presence of these cut-offs complicates the determinations of the fractal dimension.

Another important characteristic of fractal objects is that they look the same over a range

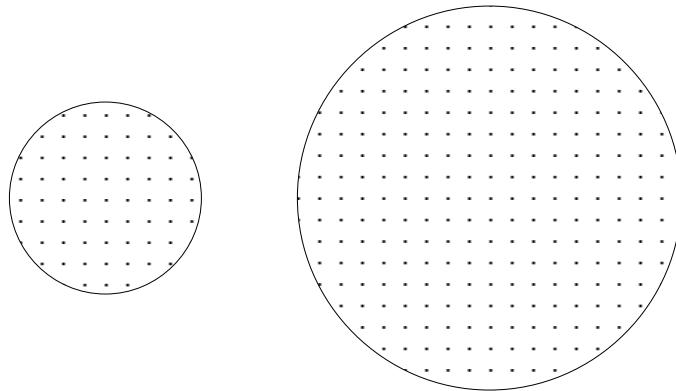


Figure 14.2: The number of dots per unit area in each circle is uniform. How does the total number of dots (mass) vary with the radius of the circle?

of length scales. This property of self-similarity or scale invariance means that if we take part of a fractal object and magnify it by the same magnification factor in all directions, the magnified picture is indistinguishable from the original.

In Problem 14.1 we compute the fractal dimension of percolation clusters using straightforward Monte Carlo methods. A renormalization group method for estimating the fractal dimension is considered in Problem 14.2. Remember that data extending over several decades is required to obtain convincing evidence for a power law relationship between M and R and to determine accurate estimates for the fractal dimension. Hence, conclusions based on the limited simulations posed in the problems need to be interpreted with caution.

Problem 14.1. The fractal dimension of percolation clusters

- Generate a site percolation configuration on a square lattice with $L = 61$ at $p = p_c \approx 0.5927$. Why might it be necessary to generate a number of configurations before a spanning cluster is obtained? Obtain a feel for the ramified nature of the spanning cluster by printing a configuration and marking the positions of the sites in the spanning cluster as in Figure 14.1. Does the spanning cluster have many dangling ends?
- Choose a point on the spanning cluster and count the number of points in the spanning cluster $M(b)$ within a square of area b^2 centered about that point. Then double b and count the number of points within the larger box. Repeat this procedure until you can estimate the b -dependence of the number of points. Can you repeat this procedure indefinitely? Use the b -dependence of $M(b)$ to estimate D according to the definition, $M(b) \sim b^D$ (see (14.1)). Choose another point in the cluster and repeat this procedure. Are your results similar? A better estimate for D can be found by averaging $M(b)$ over several origins in each spanning cluster and averaging over many spanning clusters.
- If you have not done Problem 13.9d, compute D by determining the mean size (mass) M of the spanning cluster at $p = p_c$ as a function of the linear dimension L of the lattice. Consider $L = 11, 21, 41$, and 61 and estimate D from a log-log plot of M versus L .

**Problem 14.2.* Renormalization group calculation of the fractal dimension

Compute $\langle M^2 \rangle$, the mean square number of occupied sites in the spanning cluster at $p = p_c$, and the quantity $\langle M'^2 \rangle$, the mean square number of occupied sites in the spanning cluster on the renormalized lattice of linear dimension $L' = L/b$. Because $\langle M^2 \rangle \sim R^{2D}$ and $\langle M'^2 \rangle \sim (R/b)^{2D}$, we can obtain D from the relation $b^{2D} = \langle M^2 \rangle / \langle M'^2 \rangle$. Choose the length rescaling factor to be $b = 2$ and adopt the same blocking procedure as used in Section 13.5. An average over ten spanning clusters for $L = 16$ and $p = 0.5927$ is sufficient for qualitative results.

In Problems 14.1 and 14.2 we were interested only in the properties of the spanning clusters. For this reason, our algorithm for generating percolation configurations is inefficient because it generates many clusters. There is a more efficient way of generating single percolation clusters due independently to Hammersley, Leath, and Alexandrowicz. This algorithm, commonly known as the Leath algorithm, is equivalent to the following steps (see Figure 14.3):

1. Occupy a single seed on the lattice. The nearest neighbors (four on the square lattice) of the seed represent the *perimeter* sites.
2. For each perimeter site, generate a random number r in the unit interval. If $r \leq p$, the site is occupied and added to the cluster; otherwise the site is not occupied. In order that sites be unoccupied with probability $1 - p$, these sites are not tested again.
3. For each site that is occupied, determine if there are any new perimeter sites, that is, untested neighbors. Add the new perimeter sites to the perimeter list.
4. Continue steps 2 and 3 until there are no untested perimeter sites to test for occupancy.

Program `perc_cluster` implements this algorithm and computes the number of occupied sites within a radius r of the seed particle. The seed site is placed at the center of a square lattice. Two one-dimensional arrays, `perx` and `pery`, store the x and y positions of the perimeter sites. The status of a site is stored in the array `site` with `site(x,y) = 1` an occupied site, `site(x,y) = 2` a perimeter site, and `site(x,y) = -1` a site that has already been tested and not occupied, and `site(x,y) = 0` an untested and unvisited site. To avoid checking for the boundaries of the lattice, we add an extra row and column at the boundary and set these sites equal to -1 .

```

PROGRAM perc_cluster
! cluster generated by Hammersley, Leath, and Alexandrowicz algorithm
DIM xs(10000),ys(10000),status$(-1:2)
LIBRARY "csgraphics"
CALL initial(p,L,status$())
CALL initialize_arrays(xs(),ys())
CALL grow(p,L,N,xs(),ys(),status$())
CALL mass_dist(L,N,xs(),ys())
END

SUB initial(p,L,status$())
  RANDOMIZE
  INPUT prompt "value of L (odd) = ": L

```

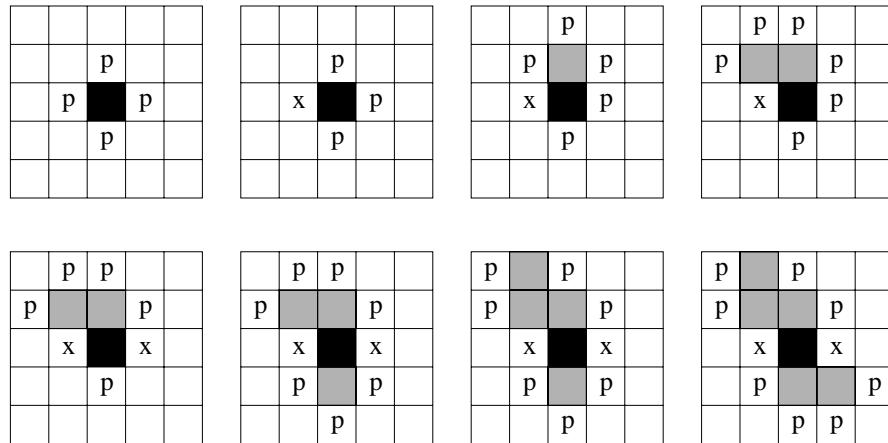


Figure 14.3: An example of the growth of a percolation cluster. Sites are occupied with probability $p = 0.5927$. Occupied sites are represented by a shaded square, perimeter sites are labeled by ‘p,’ and tested unoccupied sites are labeled by ‘x.’ Because the seed site is occupied but not tested, we have represented it differently than the other occupied sites. The perimeter sites are chosen at random.

```

INPUT prompt "site occupation probability = ": p
CALL compute_aspect_ratio(L,xwin,ywin)
SET WINDOW 0,xwin,0,ywin
SET COLOR "red"
BOX CIRCLE 0,1,0,1
FLOOD 0.5,0.5
BOX KEEP 0,1,0,1 in status$(1)      ! occupied site
CLEAR
SET COLOR "blue"
BOX CIRCLE 0,1,0,1
FLOOD 0.5,0.5
BOX KEEP 0,1,0,1 in status$(2)      ! perimeter site
CLEAR
SET COLOR "black"
BOX CIRCLE 0,1,0,1
FLOOD 0.5,0.5
BOX KEEP 0,1,0,1 in status$(-1)     ! tested, unoccupied site
CLEAR
BOX LINES 0.5,L+0.5,0.5,L+0.5
FOR y = 1 to L
  FOR x = 1 to L
    PLOT POINTS: x,y
NEXT x

```

```

NEXT y
BOX SHOW status$(1) at 1,L+2
PLOT TEXT, AT 1,L+2: " occupied site"
BOX SHOW status$(2) at 30,L+2
PLOT TEXT, AT 30,L+2: " perimeter site"
BOX SHOW status$(-1) at 60,L+2
PLOT TEXT, AT 60,L+2: " tested site"
END SUB

SUB initialize_arrays(xs(),ys())
  MAT xs = 0
  MAT ys = 0
END SUB

SUB grow(p,L,N,xs(),ys(),status$())
  ! generate single percolation cluster
  DIM perx(25000),pery(25000),site(0:131,0:131)
  DIM nx(4),ny(4)          ! set up direction vectors for lattice
  DATA 1,0,-1,0,0,1,0,-1
  ! set up boundary sites
  FOR i = 1 to L
    LET site(0,i) = -1
    LET site(L+1,i) = -1
    LET site(i,L+1) = -1
    LET site(i,0) = -1
  NEXT i
  ! seed at center of lattice
  LET xseed = int(L/2) + 1
  LET yseed = xseed
  LET site(xseed,yseed) = 1      ! seed site
  LET xs(1) = xseed
  LET ys(1) = yseed
  LET N = 1                  ! number of sites in the cluster
  BOX SHOW status$(1) at xseed-0.5,yseed-0.5
  FOR i = 1 to 4
    ! nx,ny direction vectors for new perimeter sites
    READ nx(i),ny(i)
    ! perx,pery, positions of perimeter sites of seed
    LET perx(i) = xseed + nx(i)
    LET pery(i) = yseed + ny(i)
    ! perimeter sites labeled by 2
    LET site(perx(i),pery(i)) = 2 ! site placed on perimeter list
    BOX SHOW status$(2) at perx(i)-0.5,pery(i)-0.5
  NEXT i
  LET nper = 4                ! initial number of perimeter sites
  DO

```

```

! randomly choose perimeter site
LET iper = int(rnd*nper) + 1
LET x = perx(iper)      ! coordinate of a perimeter site
LET y = pery(iper)
! relabel remaining perimeter sites so that
! last perimeter site in array replaces newly chosen site
LET perx(iper) = perx(nper)
LET pery(iper) = pery(nper)
LET nper = nper - 1
IF rnd < p then          ! site occupied
    LET site(x,y) = 1
    LET N = N + 1
    LET xs(N) = x      ! save position of occupied site
    LET ys(N) = y
    BOX SHOW status$(1) at x-0.5,y-0.5
FOR nn = 1 to 4      ! find new perimeter sites
    LET xnew = x + nx(nn)
    LET ynew = y + ny(nn)
    IF site(xnew,ynew) = 0 then
        LET nper = nper + 1
        LET perx(nper) = xnew
        LET pery(nper) = ynew
        ! place site on perimeter list
        LET site(xnew,ynew) = 2
        BOX SHOW status$(2) at xnew-0.5,ynew-0.5
    END IF
NEXT nn
ELSE                  ! rnd >= p so site is not occupied
    LET site(x,y) = -1
    BOX SHOW status$(-1) at x-0.5,y-0.5
END IF
LOOP until nper < 1      ! all perimeter sites tested
BOX LINES 0.5,L+0.5,0.5,L+0.5      ! redraw box
END SUB

SUB mass_dist(L,N,xs(),ys())
    DIM mass(10000)
    PRINT "press any key or click mouse to see data"
    DO
        GET MOUSE xm,ym,s
    LOOP until key input or s <> 0
    FOR i = 1 to N
        LET xcm = xcm + xs(i)      ! compute center of mass
        LET ycm = ycm + ys(i)
    NEXT i
    LET xcm = xcm/N

```

```

LET ycm = ycm/N
FOR i = 1 to N
    LET dx = xs(i) - xcm
    LET dy = ys(i) - ycm
    LET r = int(sqr(dx*dx + dy*dy))
    ! distance from center of mass
    ! mass(r) = number of sites at distance r from center of mass
    IF r > 1 then LET mass(r) = mass(r) + 1
NEXT i
LET rprint = 2
CLEAR
PRINT " r "," m "," ln(r) "," ln(m) "
FOR r = 2 to L/2
    LET masstotal = masstotal + mass(r)
    IF r = rprint then
        PRINT r, masstotal, log(r), log(masstotal)
        LET rprint = 2*rprint ! use logarithmic scale for r
    END IF
NEXT r
END SUB

```

We use the growth algorithm in Problem 14.3 to generate a spanning cluster at the percolation threshold. The fractal dimension is determined by counting the number of sites M in the cluster within a distance r of the center of mass of the cluster. The center of mass is defined by

$$\mathbf{r}_{\text{cm}} = \frac{1}{N} \sum_i \mathbf{r}_i, \quad (14.4)$$

where N is the total number of particles in the cluster. A typical plot of $\ln M$ versus $\ln r$ is shown in Figure 14.4. Because the cluster cannot grow past the edge of the lattice, we do not include data for $r \approx L$.

Problem 14.3. Single cluster growth and the fractal dimension

- Explain how the Leath algorithm generates single clusters in a way that is equivalent to the multiple clusters that are generated by visiting all sites. More precisely, the Leath algorithm generates percolation clusters with a distribution of cluster sizes, sn_s . The additional factor of s is due to the fact that each site of the cluster has an equal chance of being the seed of the cluster, and hence the same cluster can be generated in s ways. See Project 14.18 for a discussion of the scaling form of n_s .
- Use **Program perc_cluster** to grow percolation clusters using the Leath algorithm. Consider a spanning cluster to be one that connects the top and bottom rows of the lattice. Can you grow a spanning cluster for $p = 0.4$ or does the growth usually stop after a few sites are occupied? Choose $L \geq 31$.
- Choose $p = 0.5927$ and $L \geq 31$ and generate several pictures of spanning clusters. Do all your trials generate a spanning cluster? Explain. Determine the number of occupied sites $M(r)$

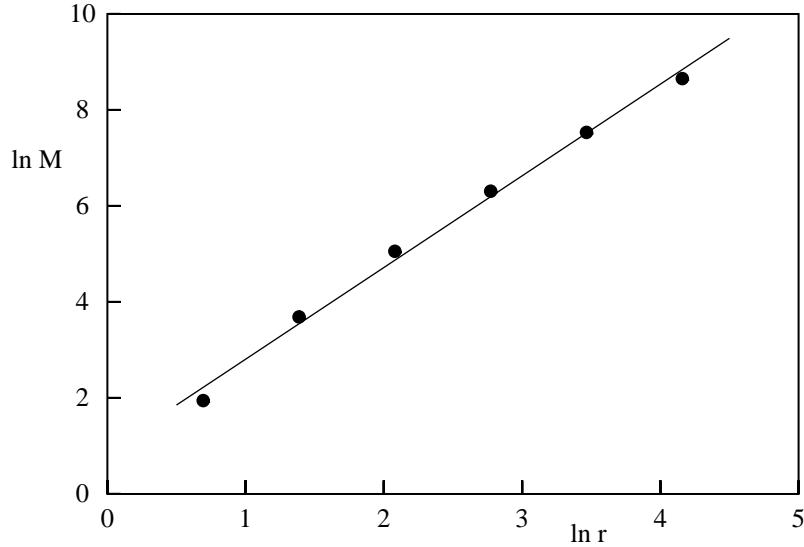


Figure 14.4: Plot of $\ln M$ versus $\ln r$ for a single spanning percolation cluster generated at $p = 0.5927$ on a $L = 129$ square lattice. The straight line is a linear least squares fit to the data. The slope of this line is 1.91 and is an estimate of the fractal dimension D . The exact value of D for a percolation cluster is $D = 91/48 \approx 1.896$.

within a distance r of the center of mass of the cluster. Determine M for several values of r and average $M(r)$ over at least ten spanning clusters. Estimate D from the log-log plot of M versus r (see Figure 14.4). If time permits, generate percolation clusters on larger lattices.

- d. Grow as large a spanning cluster as you can and look at it on different length scales. One way to do so is to divide the screen into four windows, each of which magnifies a part of the cluster shown in the previous window. Does the part of the cluster shown in each window look approximately self-similar?
- e. Generate clusters at $p = 0.65$, a value of p greater than p_c , for $L = 61$. Make a log-log plot of $M(r)$ versus r . Is the slope approximately equal to the value of D found in part (b)? Does the slope increase or decrease for larger r ? Repeat for $p = 0.80$. Is a spanning cluster generated at $p > p_c$ a fractal?
- f. The fractal dimension of percolation clusters is not an independent exponent, but satisfies the scaling law

$$D = d - \beta/\nu, \quad (14.5)$$

where β and ν are defined in Table 13.1. The relation (14.5) can be understood by a finite-size scaling argument which we now summarize. The number of sites in the spanning cluster on a lattice of linear dimension L is given by

$$M(L) \sim P_\infty(L)L^d, \quad (14.6)$$

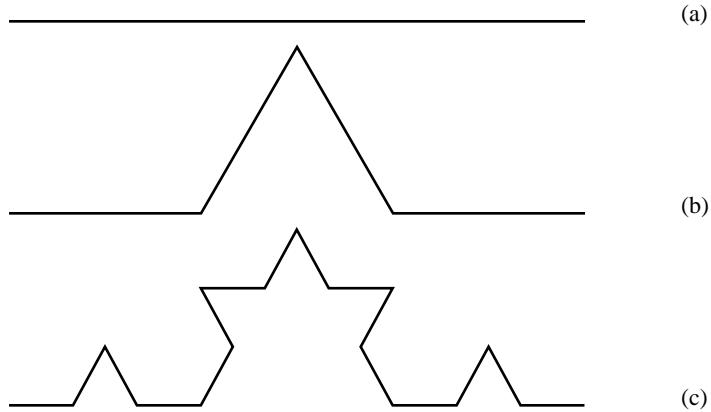


Figure 14.5: The first three stages (a)–(c) of the generation of a self-similar Koch curve. At each stage the displacement of the middle third of each segment is in the direction that increases the area under the curve. The curves were generated using `Program koch`. The Koch curve is an example of a continuous curve for which there is no tangent defined at any of its points. The Koch curve is self-similar on each length scale.

where P_∞ is the probability that an occupied site belongs to the spanning cluster and L^d is the total number of sites in the lattice. In the limit of an infinite lattice and p near p_c , we know that $P_\infty(p) \sim (p - p_c)^\beta$ and $\xi(p) \sim (p - p_c)^{-\nu}$ independent of L . Hence for $L \sim \xi$, we have that $P_\infty(L) \sim L^{-\beta/\nu}$ (see (13.14)), and we can write

$$M(L) \sim L^{-\beta/\nu} L^d \sim L^D. \quad (14.7)$$

The relation (14.5) follows. Use the exact values of β and ν from Table 13.1 to find the exact value of D for $d = 2$. Is your estimate for D consistent with this value?

- g. Estimate the fractal dimension for percolation clusters on a simple cubic lattice. Take $p_c = 0.3117$.

14.2 Regular Fractals

As we have seen, one characteristic of random fractal objects is that they look the same on a range of length scales. To gain a better understanding of the meaning of self-similarity, consider the following example of a *regular* fractal, a mathematical object that is self-similar on *all* length scales. Begin with a line one unit long (see Figure 14.5a). Remove the middle third of the line and replace it by two lines of length $1/3$ each so that the curve has a triangular bump in it and the total length of the curve is $4/3$ (see Figure 14.5b). In the next stage, each of the segments of length $1/3$ is divided into lines of length $1/9$ and the procedure is repeated as shown in Figure 14.5c. What is the length of the curve shown in Figure 14.5c?

The three stages shown in Figure 14.5 can be extended an infinite number of times. The resulting curve is infinitely long, containing an infinite number of infinitesimally small segments. Such a curve is known as the triadic Koch curve. A True BASIC program that uses a recursive procedure (see Section 6.3) to draw this curve is given in the following. Note that **SUB draw** calls itself. Use **Program koch** to generate the curves shown in Figure 14.5.

```

PROGRAM koch
! improved output 8/9/95 by computing aspect ratio
! generate triadic Koch curve using recursion
LIBRARY "csgraphics"
CALL initial(x1,y1,x2,y2,n)
! draw Koch curve for different number of iterations
DO
LET k = 0
CALL draw(x1,y1,x2,y2,n)
DO                                ! pause until any key is hit
    GET KEY k
LOOP UNTIL k <> 0
LET n = n + 1                      ! number of stages of generation
CLEAR
LOOP UNTIL k = ord("s")
END

SUB initial(x1,y1,x2,y2,n)
LET n = 0                           ! number of iterations
LET x1 = 1                           ! coordinates at left end of line
LET y1 = 5                           ! arbitrary units
LET x2 = 10                          ! coordinates at right end of line
LET y2 = 5
CALL compute_aspect_ratio(8,xwin,ywin)
SET WINDOW 0,xwin,0,ywin
ASK MAX COLOR mc
IF mc > 2 then SET COLOR "blue"
END SUB

SUB draw(x1,y1,x2,y2,n)
IF n > 0 then
    LET dx = (x2 - x1)/3
    LET dy = (y2 - y1)/3
    LET x1n = x1 + dx
    LET y1n = y1 + dy
    LET x2n = x1 + 2*dx
    LET y2n = y1 + 2*dy
    ! rotate line segment (dx,dy) by 60 degs and add to (x1n,y1n)
    LET xmid = 0.5*dx - 0.866*dy + x1n
    LET ymid = 0.5*dy + 0.866*dx + y1n

```

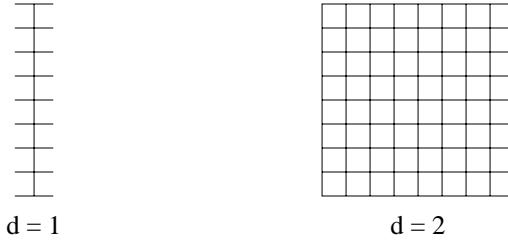


Figure 14.6: Examples of one-dimensional and two-dimensional objects.

```

CALL draw(x1,y1,x1n,y1n,n-1)
CALL draw(x1n,y1n,xmid,ymid,n-1)
CALL draw(xmid,ymid,x2n,y2n,n-1)
CALL draw(x2n,y2n,x2,y2,n-1)
ELSE
    PLOT LINES: x1,y1;x2,y2
END IF
END SUB

```

How can we determine the fractal dimension of the Koch and similar mathematical objects? In Section 14.5 we will see that there are several generalizations of the Euclidean dimension that lead naturally to a definition of the fractal dimension. Here we consider a definition based on counting boxes. Consider a one-dimensional curve of unit length that has been divided into N equal segments of length ℓ so that $N = 1/\ell$ (see Figure 14.6). As ℓ is decreased, N increases linearly—the expected result for a one-dimensional curve. Similarly if we divide a two-dimensional square of unit area into N equal subsquares of length ℓ , we have $N = 1/\ell^2$, the expected result for a two-dimensional object (see Figure 14.6). In general, we can write that $N = 1/\ell^D$, where D is the fractal dimension of the object. If we take the logarithm of both sides of this relation, we can express the fractal dimension as

$$D = \frac{\log N}{\log(1/\ell)}. \quad (\text{box dimension}) \quad (14.8)$$

Now let us apply these ideas to the Koch curve. We found that each time the length ℓ of our measuring unit is reduced by a factor of 3, the number of segments is increased by a factor of 4. Hence, we have $N = 4$ and $\ell = 1/3$, and the fractal dimension of the triadic Koch curve is given by

$$D = \frac{\log 4}{\log 3} \approx 1.2619. \quad (\text{triadic Koch curve}) \quad (14.9)$$

From (14.9) we see that the Koch curve has a dimension between that of a line and a plane. Is this statement consistent with your visual interpretation of the degree to which the triadic Koch curve fills space?

Problem 14.4. The recursive generation of regular fractals

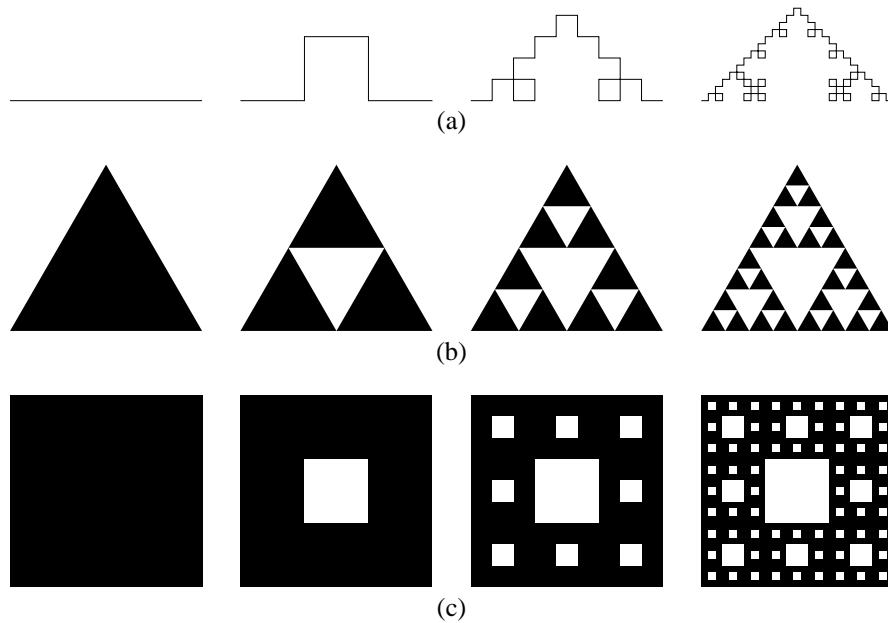


Figure 14.7: (a) The first few iterations of the quadric Koch curve; (b) The first few iterations of the Sierpiński gasket; (c) The first few iterations of the Sierpiński carpet.

- The concept of recursive programming as illustrated in [Program koch](#) is probably one of the most difficult programming concepts you will encounter. Explain the nature of recursion and the way it is implemented in [Program koch](#).
- Regular fractals can be generated from a pattern that is used in a self-replicating manner. Write a program to generate the quadric Koch curve shown in Figure 14.7a. What is its fractal dimension?
- What is the fractal dimension of the Sierpiński gasket shown in Figure 14.7b? Write a program that generates the next several iterations.
- What is the fractal dimension of the Sierpiński carpet shown in Figure 14.7c? How does the fractal dimension of the Sierpiński carpet compare to the fractal dimension of a percolation cluster? Are the two fractals visually similar?

14.3 Fractal Growth Processes

Many systems occurring in nature exhibit fractal geometry. Fractals have been used to describe the irregular shapes of such varied objects as coastlines, clouds, coral reefs, and the human lung. Why are fractal structures so common? How do fractal structures form? In this section we discuss

several simple “growth” models that generate structures which show a remarkable similarity to forms observed in nature. The first two models are already familiar to us and exemplify the flexibility and general utility of kinetic growth models.

Epidemic model. In the context of the spread of disease, we usually want to know the conditions for an epidemic. A simple lattice model of the spread of a disease can be formulated as follows. Suppose that an occupied site corresponds to an infected person. Initially there is a single infected site and the four nearest neighbor perimeter sites (on the square lattice) are susceptible. At the next time step, we visit the four susceptible sites and occupy (infect) each site with probability p . If a susceptible site is not occupied, we say that the site is immune and we do not test it again. We then find the new susceptible sites and continue until either the disease is controlled or reaches the boundary of the lattice. Convince yourself that this growth model of a disease generates a cluster of infected sites that is identical to a percolation cluster at probability p . The only difference is that we have introduced a discrete time step into the model. Some of the properties of this model are explored in Problem 14.5.

Problem 14.5. A simple epidemic model

- a. Explain why the simple epidemic model discussed in the text generates the same clusters as the Leath algorithm if the probability that a susceptible site becomes infected is p . What is the minimum value of p necessary for an epidemic to occur? Recall that in one time step, all susceptible sites are visited *simultaneously* and infected with probability p . Determine how N , the number of infected sites, depends on the time t (the number of time steps) for various values of p . A straightforward way to proceed is to modify Program `perc_cluster` so that all perimeter sites are visited and occupied with probability p before new perimeter sites are found. In Chapter ?? we will learn that this model is an example of a cellular automaton.
- b. The susceptible (or growth) sites S are the only sites from which the disease can spread. Verify that for p near p_c^+ , S increases as $f(p)N^{\delta_s}$ with $f(p) \propto (p - p_c)^y$. Estimate the numerical values of the exponents δ_s and y . How does S depend on the time? Does δ_s have a different value at p_c for clusters that grow indefinitely? Choose $L \geq 61$ and average over at least 10 realizations.
- c. A similar growth exponent can be defined for $p < p_c$. In this case the number of susceptible sites does not increase without bound, and S usually first increases and then goes to zero. Determine the maximum number S_{\max} of susceptible sites and show that $S_{\max} \propto (p_c - p)^{-x}$ near p_c . Estimate the numerical value of the exponent x .

Eden model. An even simpler example of a growth model was proposed by Eden in 1958 to simulate the growth of cell colonies. Although we will find that the resultant mass distribution is not a fractal, the description of the Eden growth algorithm illustrates the general nature of the fractal growth models we discuss.

The algorithm can be summarized as follows. Place a seed site at the origin, for example, the center of the lattice. The unoccupied nearest neighbors of the occupied sites are denoted as *growth* or perimeter sites. In the simplest version of the model, a growth site is chosen at random and occupied. The newly occupied site is removed from the list of growth sites and the new growth sites are added to the list. This growth process is repeated many times until a large cluster of occupied sites are formed (see Figure 14.8). The basic difference between this model and the previous one is that all tested sites are occupied. In other words, no sites are ever “immune.” Some of the properties of Eden clusters are investigated in Problem 14.6.

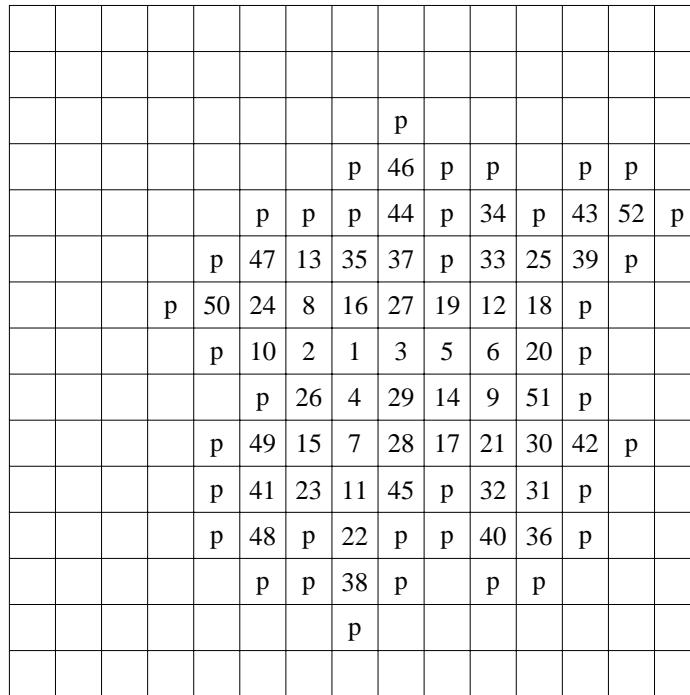


Figure 14.8: An example of a cluster grown on the square lattice according to the Eden model. The numbers on the sites denote the order in which these sites were occupied and the growth sites are denoted by the letter p.

Problem 14.6. Eden model

- Modify `Program perc_cluster` so that clusters are generated on a square lattice according to the Eden model. A straightforward modification is to occupy perimeter sites with probability $p = 1$ until the cluster reaches the edge of the lattice. What would happen if we were to occupy perimeter sites indefinitely? Follow the procedure of Problem 14.3 and determine the number of occupied sites $M(r)$ within a distance r of the seed site. Assume that $M(r) \sim r^D$ for sufficiently large r , and estimate D from the slope of a log-log plot of M versus r . A typical log-log plot is shown in Figure 14.9 for $L = 61$. Can you conclude from your data that Eden clusters are compact?
- Modify your program so that only the perimeter or growth sites are shown. Where are the majority of the perimeter sites relative to the center of the cluster? Grow as big a cluster as your time and patience permits.

Invasion percolation. A dynamical process known as *invasion percolation* can be used to model the shape of the oil-water interface which occurs when water is forced into a porous medium containing oil. The idea is to use the water to recover as much oil as possible. In this process

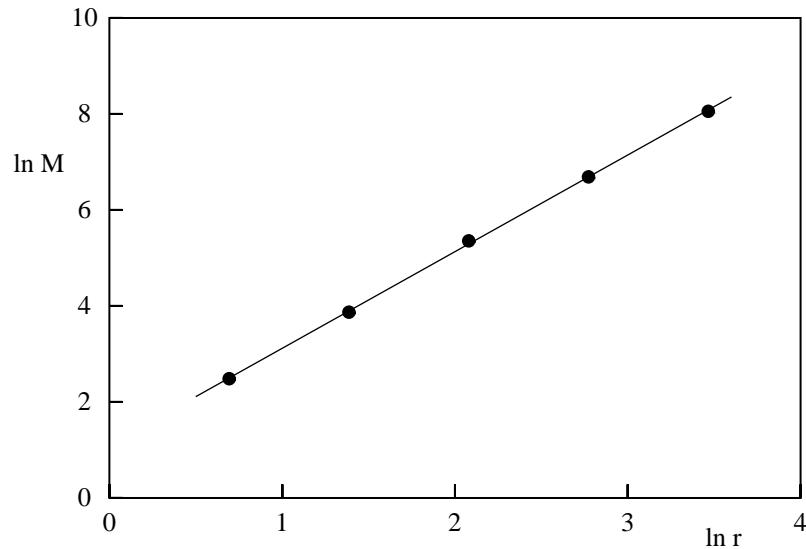


Figure 14.9: Plot of $\ln M$ versus $\ln r$ for a single Eden cluster generated on a $L = 61$ square lattice. A least squares fit to the data from $r = 2$ to $r = 32$ yields a slope of approximately 2.01.

a water cluster grows into the oil through the path of least resistance. Consider a lattice of size $L \times 2L$, with the water (the invader) initially occupying the left edge (see Figure 14.10). The resistance to the invader is given by uniformly distributed random numbers between 0 and 1 which are assigned to each site in the lattice and held fixed throughout the invasion. Sites that are nearest neighbors of the invader sites are the perimeter sites. At each time step, the perimeter site with the lowest random number is occupied by the invader and the oil (the defender) is displaced. The invading cluster grows until a path forms which connects the left and right edges of the lattice. Note that after this path forms, there is no need for the water to occupy any additional sites. To minimize boundary effects, periodic boundary conditions are used for the top and bottom edges and all quantities are measured only over the central $L \times L$ region of the lattice.

Program `invasion` implements the invasion percolation algorithm. The two-dimensional array element `site(i,j)` initially stores a random number for the site at (i,j) . If the site at (i,j) is occupied, then `site(i,j)` is increased by 1. If the site at (i,j) is a perimeter site, then `site(i,j)` is increased by 2, and is inserted into its proper ordered position in the perimeter lists `perx` and `pery`. The perimeter lists are ordered with the site with the largest random number at the beginning of the list.

Two searching routines are provided in Program `invasion` for determining the position of a new perimeter site in the perimeter lists. In a *linear search* we go through the list in order until the random number associated with the new perimeter site is between two random numbers in the list. In a *binary search* we divide the list in two, and determine the half in which the new random number belongs. Then we divide this half into half again and so on until the correct position is found. A comparison of the linear and binary search methods is investigated in Problem 14.7d.

The binary search is the default method used in Program invasion.

The main quantities of interest are the fraction of sites occupied by the invader, and the probability $P(r) dr$ that a site with a random number between r and $r + dr$ is occupied. The properties of the invasion percolation model are explored in Problem 14.7.

```

PROGRAM invasion
! generate invasion percolation cluster
DIM site(0 to 200,0 to 100),perx(5000),pery(5000)
LIBRARY "csgraphics"
CALL initial(Lx,Ly,water$,#1)
CALL assign(site(),perx(),pery(),nper,Lx,Ly,water$,#1)
CALL invade(site(),perx(),pery(),nper,Lx,Ly,water$,#1)
CALL average(site(),Lx,Ly,#1)
END

SUB initial(Lx,Ly,water$,#1)
  RANDOMIZE
  INPUT prompt "length in y direction = ": Ly
  LET Lx = 2*Ly
  OPEN #1: screen 0.01,0.65,0.2,0.99
  CALL compute_aspect_ratio(Lx,xwin,ywin)
  SET WINDOW 0,xwin,0,ywin
  SET COLOR "blue"
  BOX AREA 0,1,0,1
  FLOOD 0.5,0.5
  BOX KEEP 0,1,0,1 in water$
  CLEAR
  SET COLOR "black"
  BOX LINES 0.5,Lx+0.5,0.5,Ly+0.5
  FOR y = 1 to Ly
    FOR x = 1 to Lx
      PLOT POINTS: x,y
    NEXT x
  NEXT y
END SUB

SUB assign(site(),perx(),pery(),nper,Lx,Ly,water$,#1)
  FOR y = 1 to Ly
    LET site(1,y) = 1           ! occupy first column
    BOX SHOW water$ at 0.5,y-0.5
  NEXT y
  ! assign random numbers to remaining sites
  FOR y = 1 to Ly
    FOR x = 2 to Lx
      LET site(x,y) = rnd
    NEXT x
  
```

```

NEXT y
! sites in second column are initial perimeter sites
! site(x,y) greater than 2 if perimeter site
LET x = 2
LET nper = 0
FOR y = 1 to Ly
    LET site(x,y) = 2 + site(x,y)
    LET nper = nper + 1      ! number of perimeter sites
    ! sort perimeter sites
    ! order perimeter list
    CALL insert(site(),perx(),pery(),nper,x,y)
NEXT y
END SUB

SUB insert(site(),perx(),pery(),nper,x,y)
! call linear or binary sort
CALL binary_search(site(),perx(),pery(),nper,x,y,ninsert)
! move sites with smaller random numbers to next higher array index
FOR ilist = nper to ninsert + 1 step - 1
    LET perx(ilist) = perx(ilist-1)
    LET pery(ilist) = pery(ilist-1)
NEXT ilist
LET perx(ninsert) = x          ! new site inserted in list
LET pery(ninsert) = y
END SUB

SUB binary_search(site(),perx(),pery(),nper,x,y,ninsert)
! divide list in half and determine half in which random # belongs
! continue this division until position of number is determined
LET nfirst = 1                ! beginning of list
LET nlast = nper - 1          ! end of list
IF nlast < 1 then LET nlast = 1
LET nmid = int((nfirst + nlast)/2)      ! middle of list
! determine which half of list new number is located
DO
    IF nlast - nfirst <= 1 then      ! exact position equal to nlast
        LET ninsert = nlast
        EXIT SUB
    END IF
    LET xmid = perx(nmid)
    LET ymid = pery(nmid)
    IF site(x,y) > site(xmid,ymid) then
        LET nlast = nmid      ! search upper half
    ELSE
        LET nfirst = nmid      ! search lower half
    END IF

```

```

        LET nmid = int((nfirst + nlast)/2)
    LOOP
END SUB

SUB linear_search(site(),perx(),pery(),nper,x,y,ninsert)
    IF nper = 1 then
        LET ninsert = 1
    ELSE
        FOR iper = 1 to nper - 1
            LET xperim = perx(iper)
            LET yperim = pery(iper)
            IF site(x,y) > site(xperim,yperim) then
                LET ninsert = iper ! insert new site
                EXIT SUB
            END IF
        NEXT iper
    END IF
    LET ninsert = nper
END SUB

SUB invade(site(),perx(),pery(),nper,Lx,Ly,water$,#1)
    ! nx and ny are components of vectors pointing to nearest neighbors
    DIM nx(4),ny(4)
    DATA 1,0,-1,0,0,1,0,-1
    FOR i = 1 to 4
        READ nx(i),ny(i)
    NEXT i
    DO
        LET x = perx(nper)
        LET y = pery(nper)
        LET nper = nper - 1
        ! mark site occupied and no longer perimeter site
        LET site(x,y) = site(x,y) - 1
        BOX SHOW water$ at x-0.5,y-0.5
        FOR i = 1 to 4          ! find new perimeter sites
            LET xnew = x + nx(i)
            LET ynew = y + ny(i)
            IF ynew > Ly then      ! periodic boundary conditions in y
                LET ynew = 1
            ELSE IF ynew < 1 then
                LET ynew = Ly
            END IF
            IF site(xnew,ynew) < 1 then      ! new perimeter site
                LET site(xnew,ynew) = site(xnew,ynew) + 2
                LET nper = nper + 1
                CALL insert(site(),perx(),pery(),nper,xnew,ynew)
        
```

```

        END IF
NEXT i
LOOP until x >= Lx           ! until cluster reaches right boundary
END SUB

SUB average(site(,),Lx,Ly,#1)
    ! compute probability density P(r)
    DIM P(0 to 20),nr(0 to 20)
    LET Lmin = Lx/3
    LET Lmax = 2*Lmin
    LET n = (Lmax - Lmin + 1)*Ly ! # sites in middle half of lattice
    LET dr = 0.05
    LET nbin = 1/dr
    FOR x = Lmin to Lmax
        FOR y = 1 to Ly
            LET ibin = nbin*(mod(site(x,y),1))
            LET nr(ibin) = nr(ibin) + 1
            IF site(x,y) >= 1 and site(x,y) < 2 then
                LET occupied = occupied + 1 ! total # of occupied sites
                LET P(ibin) = P(ibin) + 1
            END IF
        NEXT y
    NEXT x
    WINDOW #1
    PRINT "# occupied sites ="; occupied
    OPEN #2: screen 0.66,1.0,0.01,0.99
    PRINT " r," P(r)"
    PRINT
    FOR ibin = 0 to nbin
        LET r = dr*ibin
        IF nr(ibin) > 0 then PRINT r,P(ibin)/nr(ibin)
    NEXT ibin
    PRINT
END SUB

```

Problem 14.7. Invasion percolation

- Use **Program invasion** to generate an invasion percolation cluster on a 20×40 lattice and describe the qualitative nature of the cluster.
- Modify **Program invasion** so that $M(L)$, the number of sites occupied by the invader in the central $L \times L$ region of the $L \times 2L$ lattice at the time that the invader first reaches the right edge, is averaged over at least twenty trials. Assume that $M(L) \sim L^D$ and estimate D from a plot of $\ln M$ versus $\ln L$. Compare your estimate for D with the fractal dimension of ordinary percolation. (The published results for $M(L)$ by Wilkinson and Willemse are for 2000 realizations each for L in the range 20 to 100.)

- c. Determine the probability $P(r) dr$ that a site with a random number between r and $r + dr$ is occupied. It is sufficient to choose $dr = 0.05$. Plot $P(r)$ versus r for $L = 20$ and also for larger values of L up to $L = 50$. Can you define a critical value of r near which $P(r)$ changes rapidly? How does this critical value of r compare to the value of p_c for ordinary site percolation on the square lattice? On the basis of your numerical estimate for the exponent D found in part (b) and the qualitative behavior of $P(r)$, make an hypothesis about the relation between the nature of the geometrical properties of the invasion percolation cluster and the spanning percolation cluster at $p = p_c$.
- d. Explain the nature of the two searching subroutines given in [Program invasion](#). Which method yields the fastest results on a 30×60 lattice? Verify that the CPU time for a linear and binary search is proportional to n and $\log n$ respectively, where n is the number of items in the list to be searched. Hence, for sufficiently large n , a binary search usually is preferred.
- e. Modify your program so that the invasion percolation clusters are grown from a seed at the origin. Grow a cluster until it either occupies a given fraction of the lattice or it reaches a boundary of the lattice. Estimate the fractal dimension as you did for the spanning percolation clusters in [Problem 14.3](#) and compare your two estimates. On the basis of this estimate and your results from part (b) and (c), can you conclude that the spanning cluster is a fractal? Note that this process of occupying the minimum number of sites to obtain a spanning cluster is an example of a self-organized critical phenomenon (see Chapter ??).

Diffusion in disordered media. In Chapters 7 and 12 we considered random walks on perfect lattices and on simple continuum systems. We found that the mean-square displacement of a random walker, $\langle R^2(t) \rangle$, is proportional to the time t for sufficiently large t . (For a simple random walk this relation holds for all t .) Now let us suppose that the random walker is restricted to a disordered lattice, for example, the occupied sites of a percolation cluster. What is the asymptotic t -dependence of $\langle R^2(t) \rangle$ in this case? This simple model of a random walk on a percolation cluster is known as the “ant in the labyrinth” problem.

There are at least two reasons for our interest in random walks on disordered lattices. Just as a random walk on a lattice is a simple model of diffusion, a random walk on a disordered lattice is a simple example of the general problem of diffusion and transport in disordered media. Because most materials of interest are noncrystalline and disordered, there are many physical phenomena that can be related to the motion of an ant in the labyrinth. Another reason for the interest in diffusion in disordered media is that the diffusion coefficient is proportional to the electrical conductivity of the medium. This relation between the conductivity and the diffusion coefficient is known as the Einstein relation (cf. Reif). We can understand this relation as follows. Consider for example, a system of electrons. Classically, we can follow the individual motion of the electrons and determine their mean square displacement. In the absence of external forces we can measure the self-diffusion coefficient D . In the presence of a “small” electric field, we can measure the electron’s mean velocity in the direction of the field and deduce the electron’s *mobility* μ , the ratio of the mean velocity to the applied force. Einstein’s contribution was to show that μ is proportional to D , that is, the linear response of the system is related to an equilibrium quantity. Because the mean velocity of the electrons is proportional to the electron current and the applied force is proportional to the voltage, the mobility and the electrical conductivity are proportional. Hence, we conclude that the conductivity is proportional to the self-diffusion coefficient.

In the usual formulation of the ant in the labyrinth problem we place a walker (the ant) at random on one of the occupied sites of a percolation cluster generated with probability p . At each time step, the ant tosses a coin with four possible outcomes (on a square lattice). If the outcome corresponds to a step to an occupied site, the ant moves; otherwise it remains in its present position. Either way, the time t is increased by one unit. The main quantity of interest is $R^2(t)$, the square of the distance between the ant's position at $t = 0$ and its position at time t . We can generate many walks with different initial positions on the same cluster as well as over many percolation clusters to obtain the ant's mean square displacement $\langle R^2(t) \rangle$. How does $\langle R^2(t) \rangle$ depend on p and t ? How do the laws of diffusion change on a fractal lattice (for example, the percolation cluster at $p = p_c$)? We consider these questions in Problem 14.8.

Problem 14.8. The ant in the labyrinth

- a. For $p = 1$, the ants walk on a perfect lattice, and hence, $\langle R^2(t) \rangle \propto t$. Suppose that an ant does a random walk on a two-dimensional percolation cluster with $p > p_c$. Assume that $\langle R^2(t) \rangle \sim 4D_s(p)t$ for $p > p_c$. We have denoted the diffusion coefficient by D_s to remind ourselves that we are considering random walks on spanning clusters only and are not considering walks on the finite clusters that also exist for $p > p_c$. Generate a percolation cluster at $p = 0.7$ using the growth algorithm considered in Problem 14.3. Choose the initial position of the ant to be the seed site and modify your program to observe the motion of the ant on the screen. Where does the ant spend much of its time? If the ant diffuses, what can you say qualitatively about the ratio $D_s(p)/D(p = 1)$?
- b. Compute $\langle R^2(t) \rangle$ for $p = 0.4$ and confirm that for $p < p_c$, the clusters are finite, $\langle R^2(t) \rangle$ is bounded, and diffusion is impossible.
- c. As in part (a) compute the mean square displacement for $p = 1.0, 0.8, 0.7, 0.65$, and 0.62 with $L = 61$. If time permits, average over several clusters. Make a log-log plot of $\langle R^2(t) \rangle$ versus t . What is the qualitative t -dependence of $\langle R^2(t) \rangle$ for relatively short times? Decide whether $\langle R^2(t) \rangle$ is proportional to t for longer times. (Remember that the maximum value of $\langle R^2 \rangle$ is bounded by the finite size of the lattice.) If $\langle R^2(t) \rangle \sim t$, estimate $D_s(p)$. Plot the ratio $D_s(p)/D(p = 1)$ as a function of p and discuss its qualitative behavior.
- d. Because there is no diffusion for $p < p_c$, we might expect that D_s vanishes as $p \rightarrow p_c$, that is, $D_s(p) \sim (p - p_c)^{\mu_s}$ for $p \geq p_c$. Extend your calculations of part (c) to larger L and more values of p near p_c and estimate the dynamical exponent μ_s .
- e. At $p = p_c$, we might expect a different type of t -dependence of $\langle R^2(t) \rangle$ to be observed, for example, $\langle R^2(t) \rangle \sim t^{2/z}$ for large t . Do you expect the exponent z to be greater or less than two? Do a Monte Carlo simulation of $\langle R^2(t) \rangle$ at $p = p_c$ and estimate z . Choose $L \geq 61$ and average over several spanning clusters.
- f. The chicken wire measurements by Watson and Leath (see Section 13.1) found that the dc electrical conductivity σ vanishes near the percolation threshold as $\sigma \sim (p - p_c)^\mu$, with $\mu \approx 1.38 \pm 0.12$. More precise estimates give $\mu \approx 1.30$. The difficulty of doing a direct Monte Carlo calculation of σ was considered in Project 13.16. From the Einstein relation we know that the electrical conductivity and the self-diffusion coefficient behave in the same way. However, we measured the self-diffusion coefficient D_s by always placing the ant on a spanning cluster rather

than on *any* cluster. In contrast, the conductivity is measured for the entire system including all finite clusters. Hence, the self-diffusion coefficient D that enters into the Einstein relation should be determined by placing the ant at random anywhere on the lattice, including sites that belong to the spanning cluster and sites that belong to the many finite clusters. Because only those ants that start on the spanning cluster can contribute to D , D is related to D_s by $D = P_\infty D_s$, where P_∞ is the probability that the ant would land on a spanning cluster. Since P_∞ scales as $P_\infty \sim (p - p_c)^\beta$, we have that $(p - p_c)^\mu \sim (p - p_c)^\beta (p - p_c)^{\mu_s}$ or $\mu_s = \mu - \beta$. Use your result for μ_s found in part (d) and the exact result $\beta = 5/36$ (see Table 13.1) to estimate μ and compare your result to the critical exponent for the dc electrical conductivity given above.

- g. We have found that $D_s(p) \sim (p - p_c)^{\mu_s}$ for $p > p_c$ and $\langle R^2(t) \rangle \sim t^{2/z}$ for $p = p_c$. We now give a simple scaling argument to find a relation between z and μ_s . For $p > p_c$, we know that $\langle R^2(t) \rangle \sim (p - p_c)^{\mu_s} t$ in the limit of $t \gg 1$ such that the root mean square displacement is much larger than the connectedness length ξ . We also know that $\langle R^2(t) \rangle \sim t^{2/z}$ for shorter time scales that satisfy the condition $\langle R^2(t) \rangle \ll \xi^2$. We expect that the crossover between the two dependencies on t occurs when $\langle R^2 \rangle \sim \xi^2$ or when $t \sim \xi^z$. Hence we have $\xi^2 \sim (p - p_c)^{\mu_s} \xi^z$, or since $\xi \sim (p - p_c)^{-\nu}$, we have $(p - p_c)^{\nu(z-2)} \sim (p - p_c)^{\mu_s}$. If we equate powers of $(p - p_c)$, we have

$$z = 2 + \frac{\mu_s}{\nu} = 2 + \frac{\mu - \beta}{\nu}. \quad (14.10)$$

Is it easier to determine μ_s or z accurately from a Monte Carlo simulation on a finite lattice? That is, if our real interest is estimating the best value of the critical exponent μ for the conductivity, should we determine the conductivity directly or should we measure the self-diffusion coefficient at $p = p_c$ or at $p > p_c$? What is your best estimate of the conductivity exponent μ ?

- h. A better method for treating random walks on a random lattice is to use an exact enumeration approach. The essence of the exact enumeration method is that $W_{t+1}(i)$, the probability that the ant is at site i at time $t + 1$, is determined solely by the probabilities of the ant being at the neighbors of site i at time t . Store the positions of the occupied sites in an array and introduce two arrays corresponding to $W_{t+1}(i)$ and $W_t(i)$ for all sites i in the cluster. Use the probabilities $W_t(i)$ to obtain $W_{t+1}(i)$ (see Figure 14.11). Spatial averages such as the mean square displacement can be calculated from the probability distribution function at different times. Details of the method and the results are discussed in Majid et al. These workers considered walks of 5000 steps on clusters with $\sim 10^3$ sites and averaged their results over 1000 different clusters.

Diffusion-limited aggregation (DLA). Many objects in nature grow by the random addition of subunits. Examples include snow flakes, lightning, crack formation along a geological fault, and the growth of bacterial colonies. Although it might seem unlikely that such phenomena have much in common, the behavior observed in many models that have been developed in recent years gives us clues that these and many other natural phenomena can be understood in terms of a few unifying principles. One model that has provided much insight is known as *diffusion limited aggregation* or DLA. The model provides an example of how random motion can give rise to beautiful self-similar clusters.

The first step is to occupy a site with a seed particle. Next, a particle is released from the perimeter of a large circle whose center coincides with the seed. The particle undergoes a random walk, that is, diffuses, until it reaches a perimeter site of the seed and sticks. Then another random walker is released and allowed to walk until it reaches a perimeter site of one of the two particles in the cluster and sticks. The process is repeated many times (typically on the order of several thousand to several million) until a large cluster is formed. A typical DLA cluster is shown in Figure 14.12. Some of the properties of DLA clusters are explored in Problem 14.9.

Problem 14.9. Diffusion limited aggregation

- a. Write a program to generate diffusion limited aggregation clusters on a square lattice. Let each walker begin at a random site on a circle of radius $r = R_{\max} + 2$, where R_{\max} is the maximum distance of any cluster particle from the origin. To save computer time, assume that a walker that reaches a distance $2R_{\max}$ from the seed site is removed and a new walker is placed at random on the circle of radius $r = R_{\max} + 2$. Choose a lattice of linear dimension $L \geq 31$. Color code the cluster sites according to their time of arrival, for example, choose the first 100 sites to be blue, the next 100 sites to be yellow, etc. Which parts of the cluster grow faster? If the clusters appear to be fractals, make a visual estimate of the fractal dimension. (Experts can make a visual estimate of D to within a few percent!)
- b. At $t = 0$ the four perimeter (growth) sites on the square lattice each have a probability $p_i = 1/4$ of growing, that is, of becoming part of the cluster. At $t = 1$, the cluster has mass two and six perimeter sites. Identify the perimeter sites and convince yourself that their growth probabilities are not uniform. Do a Monte Carlo simulation and verify that two perimeter sites have $p_i = 2/9$ and the other four have $p_i = 5/36$. We discuss a more direct way of determining the growth probabilities in Problem 14.10.
- c. It is likely that your program generates DLA clusters inefficiently, because most of the CPU time is spent while the random walker is wandering far from the perimeter sites of the cluster. There are several ways of overcoming this problem. One way is to let the walker take bigger steps the further the walker is from the cluster. For example, if the random walker is at a distance $R > R_{\max}$, a step of length greater than or equal to $R - R_{\max} - 1$ may be permitted if this distance is greater than one lattice unit. If the walker is very close to the cluster, the step length is one lattice unit. Other possible modifications are discussed by Meakin (see references). Modify your program (or see Program d1a listed below) and estimate the fractal dimension of diffusion limited clusters generated on a square lattice.
- d. Modify your program so that DLA clusters are generated on a triangular lattice. Do the clusters have the same visual appearance as on the square lattice? Estimate the fractal dimension and compare your estimate to your result for the square lattice.
- e. In Chapter 13 we found that the exponents describing the percolation transition are independent of the symmetry of the lattice, for example, the exponents for the square and triangular lattices are the same. We might expect that the fractal dimension of DLA clusters would also show such universal behavior. However, the presence of a lattice introduces a small anisotropy that becomes apparent only when very large clusters with on the order of 10^6 sites are grown. We again are reminded of the difficulty of extrapolating from finite L to infinite L . The best estimates of D for the square and triangular lattices are $D \approx 1.5$ and $D \approx 1.7$ respectively. We consider the growth of diffusion-limited aggregation clusters in a continuum in Project 14.16.

The following program provides a reasonably efficient simulation of DLA. Walkers begin just outside a circle of radius R_0 enclosing the existing cluster and centered at the seed site $(0, 0)$. If the walker moves away from the cluster, the step size for the random walker increases. If the walker wonders too far away (further than twice R_0), then the walk is started over.

```

PROGRAM dla
DIM site(-100 to 100,-100 to 100)
LIBRARY "csgraphics"
CALL initial(site(,),L)
CALL grow_cluster(site(,),L)
END

SUB initial(site(,),L)
  RANDOMIZE
  INPUT prompt "L = ": L
  CALL compute_aspect_ratio(L+2,xwin,ywin)
  SET WINDOW -xwin,xwin,-ywin,ywin
  BOX LINES -L-0.5,L+0.5,-L-0.5,L+0.5
  MAT site = 0           ! initialize sites
  FOR y = -L to L
    FOR x = -L to L
      PLOT POINTS: x,y
    NEXT x
  NEXT y
  SET COLOR "red"
END SUB

SUB grow_cluster(site(,),L)
  LET R0 = 3           ! start walker at distance R0 from origin
  LET site(0,0) = 1     ! seed site
  LET N = 1             ! number of particles in cluster
  BOX AREA -0.5,0.5,-0.5,0.5
  DO
    ! find random initial position of new walker
    LET theta = 2*pi*rnd
    LET x = int(R0*cos(theta))
    LET y = int(R0*sin(theta))
    CALL walk(site(,),L,x,y,R0,N)   ! random walk
  LOOP until key input
END SUB

SUB walk(site(,),L,x,y,R0,N)
  ! walk until on a perimeter site of cluster
  ! or walker strays too far from cluster
  DO
    LET onperimeter$ = "no"

```

```

LET r = sqr(x*x + y*y)
IF r < R0 + 1 then
    ! test if walker on perimeter
    CALL test(site(,),L,x,y,r,R0,N,onperimeter$)
END IF
LET step = int(r - R0) - 1      ! big step
IF step < 1 then LET step = 1
IF onperimeter$ = "no" then
    LET random = rnd
    IF random < 0.25 then
        LET x = x + step
    ELSE IF random < 0.5 then
        LET x = x - step
    ELSE IF random < 0.75 then
        LET y = y + step
    ELSE
        LET y = y - step
    END IF
END IF
LOOP until onperimeter$ = "yes" or r > 2*R0
END SUB

SUB test(site(,),L,x,y,r,R0,N,onperimeter$)
LET sum = site(x,y+1) + site(x,y-1) + site(x+1,y) + site(x-1,y)
IF sum > 0 then          ! walker on perimeter site
    LET site(x,y) = 1
    LET onperimeter$ = "yes"
    IF abs(x) <= L and abs(y) <= L then
        BOX AREA x-0.5,x+0.5,y-0.5,y+0.5
        LET N = N + 1
        SET CURSOR 2,1
        PRINT using "N = #####": N
    ELSE
        STOP          ! cluster outside box
    END IF
    IF r >= R0 then LET R0 = int(r+2)
END IF
END SUB

```

***Laplacian growth model.** As we discussed in Section 10.3, we can formulate the solution of Laplace's equation in terms of a random walk. We now do the converse and formulate the DLA algorithm in terms of a solution to Laplace's equation. Consider the probability $P(\mathbf{r})$ that a random walker reaches a site \mathbf{r} between the external boundary and the growing cluster without

having visited the cluster or the external boundary. This probability satisfies the relation

$$p(\mathbf{r}) = \frac{1}{4} \sum_{\mathbf{a}} p(\mathbf{r} + \mathbf{a}), \quad (14.11)$$

where the sum in (14.11) is over the four nearest neighbor sites (on a square lattice). If we set $p = 1$ on the boundary and $p = 0$ on the cluster, then (14.11) also applies to sites that are neighbors of the external boundary and the cluster. A comparison of the form of (14.11) with the form of (10.13) shows that the former is a discrete version of Laplace's equation, $\nabla^2 p = 0$. Hence $p(\mathbf{r})$ has the same behavior as the electrical potential between two electrodes connected to the outer boundary and the cluster respectively, and the growth probability at a perimeter site of the cluster is proportional to the value of the potential at that site.

**Problem 14.10.* Laplacian growth models

- a. Solve the discrete Laplace equation (14.11) by hand for the growth probabilities of a DLA cluster of mass 1, 2, and 3. Set $p = 1$ on the boundary and $p = 0$ on the cluster.
- b. You are probably familiar with the complicated and random nature of electrical discharge patterns that occur in atmospheric lightning. Although this phenomenon, known as *dielectric breakdown*, is complicated, we will see that a simple model leads to discharge patterns that are similar to those observed experimentally. Because lightning occurs in an inhomogeneous medium with differences in the density, humidity and conductivity of air, we want to develop a model of an electrical discharge in an inhomogeneous insulator. We know that when an electrical discharge occurs, the electrical potential ϕ satisfies Laplace's equation $\nabla^2 \phi = 0$. One version of the model (see Family et al.) is specified by the following steps:
 - (a) Consider a large boundary circle of radius R and place a charge source at the origin. Choose the potential $\phi = 0$ at the origin (occupied site) and $\phi = 1$ for sites on the circumference of the circle. The radius R should be larger than the radius of the growing pattern.
 - (b) Use the relaxation method (see Chapter 10) to compute the values of the potential ϕ_i for (empty) sites within the circle.
 - (c) Assign a random number r to each empty site within the boundary circle. The random number r_i at site i represents a breakdown coefficient and the random inhomogeneous nature of the insulator.
 - (d) The perimeter sites are the nearest neighbor sites of the discharge pattern (occupied sites). Form the product $r_i \phi_i^a$ for each perimeter site i , where a is an adjustable parameter. (Because the potential for the discharge pattern is zero, ϕ_i for perimeter site i can be interpreted as the magnitude of the potential gradient at site i .)
 - (e) The perimeter site with the maximum value of the product $r_i \phi_i^a$ breaks down, that is, set ϕ for this site equal to zero. (We can say that the bond between the discharge pattern and the perimeter site breaks down.)
 - (f) Use the relaxation method to recalculate the values of the potential at the remaining unoccupied sites and repeat steps (4)–(6).

Choose $a = \frac{1}{4}$ and analyze the structure of the discharge pattern. Does the pattern appear qualitatively similar to lightning? Does the pattern appear to have a fractal geometry? Estimate the fractal dimension by counting $M(b)$, the average number of sites belonging to the discharge pattern that are within a $b \times b$ box. Consider other values of a , for example, $a = \frac{1}{6}$ and $a = \frac{1}{3}$, and show that the patterns have a fractal structure with a tunable fractal dimension. Published results (Family et al.) are for patterns generated with 800 occupied sites.

- c. The usual version of the dielectric breakdown model associates a growth probability $p_i = \phi_i^a / \sum_j \phi_j^a$ with each perimeter site i , where the sum is over all perimeter sites. One of the perimeter sites is occupied with probability p_i . That is, choose a perimeter site at random and generate a random number r between 0 and 1. If $r \leq p_i$, the perimeter site i is occupied. As before, the exponent a is a free parameter. Convince yourself that $a = 1$ corresponds to diffusion-limited aggregation. (The boundary condition used in the latter corresponds to a zero potential at the perimeter sites.) To what type of cluster does $a = 0$ correspond? Choose $a = 1/2, 1$, and 2 and explore the dependence of the visual appearance of the clusters on a . If time permits, estimate the fractal dimension of the clusters.
- d. Consider a deterministic growth model for which *all* perimeter sites are tested for occupancy at each growth step. We adopt the same geometry and boundary conditions as in part (b) and use the relaxation method to solve Laplace's equation for ϕ_i . Then we find the perimeter site with the largest value of ϕ and set this value equal to ϕ_{\max} . Only those perimeter sites for which the ratio ϕ_i/ϕ_{\max} is larger than a parameter p become part of the cluster and ϕ_i is set equal to unity for these sites. After each growth step, the new perimeter sites are determined and the relaxation method is used to recalculate the values of ϕ_i at each unoccupied site. Choose $p = 0.35$ and determine the nature of the regular fractal pattern. What is the fractal dimension? Consider other values of p and determine the corresponding fractal dimension. These patterns have been termed *Laplace fractal carpets* (see Family et al.).

***Cluster-cluster aggregation.** Fractal structures commonly occur in aggregates that have been formed by the clustering of particles that are diffusing in a fluid. For example, colloids consist of particles that stick together in a liquid solvent, and aerosols are the analog in a gas. In DLA, all the particles that stick to a cluster are the same size (the growth occurs by cluster-monomer contact), and the cluster that is formed is motionless. In the following, we consider a cluster-cluster aggregation (CCA) model in which the clusters diffuse as they aggregate.

In a typical simulation we begin with a dilute collection of N particles. Each of these particles is a cluster with unit mass. The particles do random walks until one of them becomes a nearest neighbor of another particle. At that point they stick together to form a cluster of two particles. This new cluster now moves as a single random walker with a reduced diffusion coefficient. As this process continues, the clusters become larger and fewer in number. For simplicity, we assume a square lattice with periodic boundary conditions. The CCA algorithm can be summarized as follows:

1. Place N particles at random positions on the lattice. Do not allow a site to be occupied by more than one particle. Identify the i th particle with the i th cluster.
2. Check if any two clusters have particles that are nearest neighbors. If so, join these two clusters to form a single cluster.

3. Choose a cluster at random. Decide whether to move the cluster as discussed below. If so, move it randomly in one of the four possible directions. In the following, we discuss the strategy for deciding when to move a cluster.
4. Repeat steps 2 and 3 until the desired time or until there is only a single cluster.

What rule should we use to decide whether to move a cluster? One possibility is to select a cluster at random and simply move it. This possibility corresponds to all clusters having the same diffusion coefficient, regardless of the mass s of the cluster. A more realistic rule is to assume that the diffusion coefficient D_s is inversely related to the mass, for example as s^{-x} with $x \neq 0$. A common assumption in the literature is to assume $x = 1$. If we assume instead that D_s is inversely proportional to the linear dimension of the cluster, an assumption consistent with the Stokes-Einstein relation, it is reasonable to take $x = 1/d$. However because the clusters are fractals, we really should take $x = 1/D$, where D is the fractal dimension of the cluster. In Problem 14.11 we explore some of the possible forms of D_s .

To implement the cluster-cluster aggregation algorithm, we need to store the position of each particle and the cluster to which each particle belongs. In Program `cca` the position of a particle is given by its x and y coordinates and stored in the arrays `x` and `y` respectively. The array element `site(x,y)` equals zero if there is no particle at (x,y) ; otherwise the element equals the label of the cluster to which the particle at (x,y) belongs.

The labels of the clusters are found as follows. The array element `first_particle(k)` gives the particle label of the first particle in the k th cluster. To determine all the particles in a given cluster, we use a data structure called a *linked list*. This list is implemented using an array such that the value of an element of the array is the index for the next element in the linked list. The linked list is an example of a *circular linked list*, because the value of the last element in the linked list is the index for the first element. The array `next_particle` contains a series of circular linked lists, one for each cluster, such that `next_particle(i)` equals the particle label of another particle in the same cluster as the i th particle. If `next_particle(i) = i`, then the i th particle is a cluster with only one particle. To see how these arrays work, consider three particles 5, 9, and 16 which constitute cluster 4. We have `first_particle(4) = 5`, `next_particle(5) = 9`, `next_particle(9) = 16`, and `next_particle(16) = 5`.

As the clusters undergo a random walk, we need to check if any pair of particles in different clusters have become nearest neighbors. If such a situation occurs, their respective clusters have to be merged. The check for nearest neighbors is done in `SUB neighbor`. If for example, `site(x,y)` and `site(x+1,y)` are both nonzero and are not equal, then the two clusters associated with these sites need to be combined. To do so, we combine the two circular lists for each cluster into one circular list as is done in `SUB merge`. Hence if p_1 and p_2 are the first particles of their respective clusters, then

```
LET p1next = next_particle(p1)
LET p2next = next_particle(p2)
```

gives the second particle in each cluster. The following code merges the two clusters.

```
LET next_particle(p1) = p2next
LET next_particle(p2) = p1next
```

To complete the merger, all the entries in `site(x,y)` corresponding to the second cluster are relabeled with the label for the first cluster.

```

PROGRAM cca
PUBLIC site(50,50),x(1000),y(1000),L,N
LIBRARY "csgraphics"
DIM next_particle(1000),first_particle(1000)
CALL initial(first_particle(),next_particle(),ncl)
DO
    CALL move(next_particle(),first_particle(),ncl)
LOOP until ncl = 1
END

SUB initial(first_particle(),next_particle(),ncl)
PUBLIC nx(4),ny(4)
DECLARE PUBLIC site(,),x(),y()
DECLARE PUBLIC L,N
RANDOMIZE
DATA 1,0,-1,0,0,1,0,-1
FOR nn = 1 to 4
    READ nx(nn),ny(nn)
NEXT nn
INPUT prompt "L = ": L
INPUT prompt "N = ": N
CALL compute_aspect_ratio(L+1,xwin,ywin)
SET WINDOW 0,xwin,0,ywin
CLEAR
BOX LINES 1,L+1,1,L+1
SET COLOR "blue"
LET ncl = 0           ! number of clusters
FOR i = 1 to N
    DO
        LET x(i) = int(L*rnd) + 1
        LET y(i) = int(L*rnd) + 1
    LOOP until site(x(i),y(i)) = 0
    LET ncl = ncl + 1
    LET site(x(i),y(i)) = ncl
    BOX AREA x(i),x(i)+1,y(i),y(i)+1
    LET first_particle(ncl) = i
    LET next_particle(i) = i
    LET xi = x(i)
    LET yi = y(i)
    CALL neighbor(xi,yi,next_particle(),first_particle(),ncl)
NEXT i
END SUB

```

```

SUB neighbor(xi,yi,next_particle(),first_particle(),ncl)
  DECLARE PUBLIC site(,),x(),y()
  DECLARE PUBLIC L,nx(),ny()
  DECLARE DEF pbc
  FOR nn = 1 to 4
    LET px = pbc(xi + nx(nn),L)
    LET py = pbc(yi + ny(nn),L)
    LET perim = site(px,py)
    LET part = site(xi,yi)
    IF perim <> 0 and perim <> part then
      CALL merge(perim,part,first_particle(),next_particle(),ncl)
    END IF
  NEXT nn
END SUB

SUB merge(c1,c2,first_particle(),next_particle(),ncl)
  DECLARE PUBLIC site(,), x(), y()
  LET p1 = first_particle(c1)
  LET p2 = first_particle(c2)
  LET p1next = next_particle(p1)
  LET p2next = next_particle(p2)
  LET next_particle(p1) = p2next
  LET next_particle(p2) = p1next
  DO
    LET site(x(p2next),y(p2next)) = c1
    LET p2next = next_particle(p2next)
  LOOP until p2next = p1next
  LET plast = first_particle(ncl)
  IF c2 <> ncl then
    LET p = plast
    DO
      LET site(x(p),y(p)) = c2
      LET p = next_particle(p)
    LOOP until p = plast
    LET first_particle(c2) = plast
  END IF
  LET ncl = ncl - 1
END SUB

SUB move(next_particle(),first_particle(),ncl)
  DECLARE PUBLIC site(,),x(),y()
  DECLARE PUBLIC L,nx(),ny()
  DECLARE DEF pbc
  LET c = int(ncl*rnd) + 1
  LET direction = int(4*rnd) + 1
  LET p1 = first_particle(c)

```

```

LET i = p1
LET dx = nx(direction)
LET dy = ny(direction)
DO
    LET site(x(i),y(i)) = 0
    SET COLOR "white"
    BOX AREA x(i),x(i)+1,y(i),y(i)+1
    LET x(i) = pbc(x(i) + dx,L)
    LET y(i) = pbc(y(i) + dy,L)
    LET i = next_particle(i)
LOOP until i = p1
DO
    SET COLOR "blue"
    BOX AREA x(i),x(i) + 1, y(i),y(i) + 1
    LET site(x(i),y(i)) = c
    LET i = next_particle(i)
LOOP until i = p1
DO
    LET xi = x(i)
    LET yi = y(i)
    CALL neighbor(xi,yi,next_particle(),first_particle(),ncl)
    LET i = next_particle(i)
LOOP until i = p1
SET COLOR "black"
BOX LINES 1,L + 1,1,L + 1
END SUB

FUNCTION pbc(s,L)
    IF s > L then
        LET pbc = 1
    ELSE IF s < 1 then
        LET pbc = L
    ELSE
        LET pbc = s
    END IF
END DEF

```

*Problem 14.11. Cluster-cluster aggregation

- Program cca assumes that the diffusion coefficient is independent of the cluster mass. Run Program cca with $L = 40$ and $N = 300$ and describe the qualitative appearance of the clusters as they form. Do they appear to be fractals? Compare their appearance to DLA clusters.
- Choose $L = 50$ and $N = 500$ and compute the fractal dimension of the final cluster. (Speed up the program by eliminating the visual display of the formation of the clusters.) Use the center of mass, \mathbf{r}_{cm} , as the origin of the cluster, where $\mathbf{r}_{cm} = (1/N)(\sum_i \mathbf{r}_i)$ and (x_i, y_i) is the position of the i th particle. Average your results over at least ten final clusters. Do the same

for other values of L and N . Are the clusters formed by cluster-cluster aggregation more or less space filling than DLA clusters?

- c. Assume that the diffusion coefficient of a cluster of s particles varies as $D_s \propto s^{-1/d}$, where d is the spatial dimension. Let D_{\max} be the diffusion coefficient of the largest cluster. Add to your program an array that tracks the mass of each cluster. Choose a random number r between 0 and 1 and move the cluster if $r < D_s/D_{\max}$. Repeat the above simulations and discuss any changes in your results. What effect does this dependence of D on s have on the motion of the clusters? In any case increase the time by one step. The time-dependence of the cluster size distribution is investigated in Project 14.19.

Surface growth. The fractal objects we have discussed so far are self-similar, that is, if we look at a small piece of the object and magnify it isotropically to the size of the original, then the original and the magnified object look similar (on the average). In the following, we introduce some simple models that generate a class of fractals that are self-similar only for scale changes in certain directions.

One of the problems in surface science is understanding the formation of rough surfaces. Suppose that we have a flat surface at time $t = 0$. Let us ask how the surface grows as a result of vapor deposition and sedimentation. For example, consider a surface which initially is a line of L occupied sites. Growth is confined to the vertical direction (see Figure 14.13).

As before, we simply choose a perimeter site at random and occupy it. The average height of the cluster is given by

$$\bar{h} = \frac{1}{N_s} \sum_{i=1}^{N_s} h_i, \quad (14.12)$$

where h_i is the distance of the i th surface site from the substrate, and the sum is over all surface sites N_s . (The precise definition of a surface site for the Eden model is discussed in Problem 14.12.)

Each time a particle is deposited, the time t is increased by unity. Our main interest is how the “width” of the surface changes with t . We define the width of the surface by

$$w^2 = \frac{1}{N_s} \sum_{i=1}^{N_s} (h_i - \bar{h})^2. \quad (14.13)$$

In general, the surface width w , which is a measure of the surface roughness, depends on L and t . Initially w grows with time. We expect that

$$w(L, t) \sim t^\beta. \quad (14.14)$$

The exponent β describes the growth of the correlations with time along the vertical direction. Figure 14.13 illustrates the evolution of the surface generated according to the Eden model. After a characteristic time, the length over which the fluctuations are correlated becomes comparable to L , and the width reaches a steady state value that depends only on L . We write

$$w(L, t \gg 1) \sim L^\alpha, \quad (14.15)$$

where α is known as the roughness exponent.

From (14.15) we see that in the steady state, the width of the surface in the direction perpendicular to the substrate grows as L^α . This steady-state behavior of the width is characteristic of a *self-affine fractal*. Such a fractal is invariant (on the average) under anisotropic scale changes, that is, different scaling relations exist along different directions. For example, if we rescale the surface by a factor b in the horizontal direction, then the surface must be rescaled by a factor of b^α in the direction perpendicular to the surface to preserve the similarity along the original and rescaled surfaces.

Note that on short length scales, that is, lengths shorter than the width of the interface, the surface is rough and its roughness can be characterized by the exponent α . (Imagine an ant walking on the surface.) However on length scales much larger than the width of the surface, the surface appears to be flat and, in our example, it is a one-dimensional object. The properties of the surface as given by several growth models are explored in Problem 14.12.

Problem 14.12. Growing surfaces

- Eden model.* In the Eden model a perimeter site is chosen at random and occupied. In this model there can be “overhangs” as shown in Figure 14.13, and the height h_x corresponds to the maximum distance of any perimeter site in column x from the surface. Use periodic boundary conditions in the horizontal directions to determine the perimeter sites. Note that the growth rule is the same as the usual Eden model, but the growth is started from the top of a strip of length L . Choose a square lattice with $L = 100$. Describe the visual appearance of the surface as the surface grows. Is the surface well-defined visually? Where are most of the perimeter sites? We have defined the surface sites as a subset of the perimeter sites (that is, those with maximum h for a given x). Do you think our results would be qualitatively different if we included all perimeter sites?
- Plot the width $w(t)$ as a function of t for $L = 32, 64$, and 128 on the same graph and estimate the exponents α and β for the Eden model. What type of plot is most appropriate? Does the width initially grow as a power law? If so, estimate the exponent β . Is there a L -dependent crossover time after which the width of the surface approaches its steady state value? How can you estimate the exponent α ? The best numerical estimates for β and α are consistent with the presumed exact values $\beta = 1/3$ and $\alpha = 1/2$, respectively.
- The dependence of $w(L, t)$ on t and L can be combined into the scaling form

$$w(L, t) \approx L^\alpha f(t/L^{\alpha/\beta}) \quad (14.16)$$

where

$$f(x) \approx x^\beta \quad \text{for } x \ll 1 \quad (14.17a)$$

$$f(x) = \text{constant} \quad \text{for } x \gg 1 \quad (14.17b)$$

Verify the existence of the scaling form (14.16) by plotting the ratio $w(L, t)/L^\alpha$ versus $t/L^{\alpha/\beta}$ for the different values of L considered in part (b). If the scaling forms holds, the results for w for the different values of L should fall on a universal curve. Use either the estimated values of α and β that you found in part (b) or the exact results.

- d. *Random deposition.* The Eden model is not really a surface growth model, because any perimeter site can become part of the cluster. In the simplest deposition model, a column is chosen at random and a particle is deposited at the top of the column of already deposited particles. There is no horizontal correlation between neighboring columns. Do a simulation of this growth model and visually inspect the surface of the interface. Show that the heights of the columns follow a Poisson distribution (see (7.16)) and that $\bar{h} \sim t$ and $w \sim t^{1/2}$. This structure does not depend on L and hence $\alpha = 0$.
- e. *Ballistic deposition.* In this model a column is chosen at random and a particle is assumed to fall vertically until it reaches the first perimeter site that is a nearest neighbor of a site that already is part of the surface. This condition allows for growth parallel to the substrate. Only one particle falls at a time. How do the rules for this growth model differ from those of the Eden model? How does the deposit that you obtain compare to that of the Eden model? Suppose that instead of the particle falling vertically, we let it do a random walk as in DLA. Would the resultant surface be the same?

14.4 Fractals and Chaos

In Chapter 6 we explored dynamical systems that exhibited chaos under certain conditions. We found that after an initial transient, the trajectory of a dynamical system consists of a set of points in phase space called an attractor. For chaotic motion this attractor often is an object that can be described by a fractal dimension. Such attractors are called *strange attractors*.

We first consider the familiar logistic map (see (6.1)), $x_{n+1} = 4rx_n(1 - x_n)$. For most values of the control parameter $r > r_\infty = 0.892486417967\dots$, the trajectories are chaotic. Are these trajectories fractals? We explore this question in Problem 14.13.

To calculate the fractal dimension for dynamical systems, we use the *box counting* method in which space is divided into d -dimensional boxes of length ℓ . Let $N(\ell)$ equal the number of boxes that contain a piece of the trajectory. The fractal dimension is defined by the relation

$$N(\ell) \sim \lim_{\ell \rightarrow 0} \ell^{-D}. \quad (\text{box counting dimension}) \quad (14.18)$$

Equation (14.18) is accurate only when the number of boxes is much larger than $N(\ell)$ and the number of points on the trajectory is sufficiently large. If the trajectory moves through many dimensions, that is, the phase space is very large, box counting becomes too memory intensive because we need an array of size $\propto \ell^{-d}$. This array becomes very large for small ℓ and large d .

A more efficient approach is to calculate the *correlation dimension*. In this approach we store in an array the position of N points on the trajectory. We compute the number of points $N_i(r)$, and the fraction of points $f_i(r) = N_i(r)/(N - 1)$ within a distance r of the point i . The correlation function $C(r)$ is defined by

$$C(r) \equiv \frac{1}{N} \sum_i f_i(r), \quad (14.19)$$

and the *correlation dimension* D_c is defined by

$$C(r) \sim \lim_{r \rightarrow 0} r^{D_c}. \quad (\text{correlation dimension}) \quad (14.20)$$

From (14.20) we see that the slope of a log-log plot of $C(r)$ versus r yields an estimate of the correlation dimension. In practice, small values of r must be discarded because we cannot sample all the points on the trajectory, and hence there is a cutoff value of r below which $C(r) = 0$. In the large r limit, $C(r)$ saturates to unity if the trajectory is localized as it is for chaotic trajectories. We expect that for intermediate values of r , there is a scaling regime where (14.20) holds.

In Problems 14.13–14.15 we consider the fractal properties of some of the dynamical systems that we considered in Chapter 6.

Problem 14.13. Fractal dimension of logistic map trajectories

- a. Write a program that uses box counting to determine the fractal dimension of the attractor for the logistic map. Compute $N(\ell)$, the number of boxes of length ℓ that have been visited by the trajectory. Test your program for $r < r_\infty$. How does the number of boxes containing a piece of the trajectory change with ℓ ? What does this dependence tell you about the dimension of the trajectory?
- b. Compute $N(\ell)$ for $r = 0.9$ using at least five different values of ℓ , for example, $1/\ell = 100, 300, 1000, 3000, \dots$. Iterate the map at least 1000 times before determining $N(\ell)$. What is the fractal dimension of the attractor? Repeat for $r \approx r_\infty$, $r = 0.95$, and $r = 1$.
- c. Generate points at random in the unit interval and estimate the fractal dimension using the same method as in part (b). What do you expect to find? Use your results to estimate the accuracy of the fractal dimension that you found in part (b).
- d. Write a program to compute the correlation dimension for the logistic map and repeat the calculations for parts (b) and (c).

Problem 14.14. Strange attractor of the Hénon map

- a. Use two-dimensional boxes of linear dimension ℓ to estimate the fractal dimension of the strange attractor of the Hénon map (see (6.25)) with $a = 1.4$ and $b = 0.3$. Iterate the map at least 100 times before computing $N(\ell)$. Does it matter what initial condition you choose?
- b. Compute the correlation dimension for the same parameters used in part (a) and compare D_c with the box dimension computed in part (a).
- c. Iterate the Hénon map and view the trajectory on the screen by plotting x_{n+1} versus x_n in one window and y_n versus x_n in another window. Do the two ways of viewing the trajectory look similar? Estimate the correlation dimension, where the i th data point is defined by (x_i, x_{i+1}) and the distance R_{ij} between the i th and j th data point is given by $R_{ij}^2 = (x_i - x_j)^2 + (x_{i+1} - x_{j+1})^2$.
- d. Estimate the correlation dimension with the i th data point defined by x_i , and $R_{ij}^2 = (x_i - x_j)^2$. What do you expect to obtain for D_c ? Repeat the calculation for the i th data point given by (x_i, x_{i+1}, x_{i+2}) and $R_{ij}^2 = (x_i - x_j)^2 + (x_{i+1} - x_{j+1})^2 + (x_{i+2} - x_{j+2})^2$. What do you find for D_c ?

**Problem 14.15.* Strange attractor of the Lorenz model

- a. Use three-dimensional graphics or three two-dimensional plots of $x(t)$ versus $y(t)$, $x(t)$ versus $z(t)$, and $y(t)$ versus $z(t)$ to view the structure of the Lorenz attractor. Use $\sigma = 10$, $b = 8/3$, $r = 28$, and the time step $\Delta t = 0.01$. Then compute the correlation dimension for the Lorenz attractor.
- b. Repeat the calculation of the correlation dimension using $x(t)$, $x(t + \tau)$, and $x(t + 2\tau)$ instead of $x(t)$, $y(t)$, and $z(t)$. Choose the delay time τ to be at least ten times greater than the time step Δt .
- c. Compute the correlation dimension in the two-dimensional space of $x(t)$ and $x(t + \tau)$. Do the same calculation in four dimensions using $x(t)$, $x(t + \tau)$, $x(t + 2\tau)$, and $x(t + 3\tau)$. What can you conclude about the results for the correlation dimension using two, three, and four-dimensional spaces. What do you expect to see for $d > 4$?

Problems 14.14 and 14.15 illustrate a practical method for determining the underlying structure of systems when, for example, the data consists only of a single time series, that is, measurements of a single quantity over time. The dimension $D_c(d)$ computed by increasing the dimension of the space, d , using the delayed coordinate τ eventually saturates when d is approximately equal to the number of variables that actually determine the dynamics. Hence, if we have extensive data for a single variable, for example, the atmospheric pressure, we can use this method to determine the number of independent variables that determine the dynamics of the pressure. This information can then be used to help create models of the atmosphere.

14.5 Many Dimensions

So far we have discussed three ways of defining the fractal dimension: the mass dimension (14.1), the box counting dimension (14.18), and the correlation dimension (14.20). These methods do not always give the same results for the fractal dimension. Indeed, there are many other dimensions that we could compute. For example, instead of just counting the boxes that contain a part of an object, we can count the number of points of the object in each box, n_i , and compute $p_i = n_i/N$, where N is the total number of points. A generalized dimension D_q can be defined as

$$D_q = \frac{1}{q-1} \lim_{\ell \rightarrow 0} \frac{\ln \sum_{i=1}^{N(\ell)} p_i^q}{\ln \ell}. \quad (14.21)$$

The sum in (14.21) is over all the boxes and involves the probabilities raised to the q th power. For $q = 0$, we have

$$D_0 = - \lim_{\ell \rightarrow 0} \frac{\ln N(\ell)}{\ln \ell}. \quad (14.22)$$

If we compare the form of (14.22) with (14.18), we can identify D_0 with the box-counting dimension. For $q = 1$, we need to take the limit of (14.21) as $q \rightarrow 1$. Let

$$u(q) = \ln \sum_i p_i^q, \quad (14.23)$$

and do a Taylor-series expansion of $u(q)$ about $q = 1$. We have

$$u(q) = u(1) + (q - 1)\frac{du}{dq} + \dots \quad (14.24)$$

The quantity $u(1) = 0$ because $\sum_i p_i = 1$. The first derivative of $u(q)$ is given by

$$\frac{du}{dq} = \frac{\sum_i p_i^q \ln p_i}{\sum_i p_i^q} = \sum_i p_i \ln p_i, \quad (14.25)$$

where the last equality follows by setting $q = 1$. If we use the above relations, we find that D_1 is given by

$$D_1 = \lim_{\ell \rightarrow 0} \frac{\sum_i p_i \ln p_i}{\ln \ell}. \quad (\text{information dimension}) \quad (14.26)$$

D_1 is called the *information dimension* because of the similarity of the $p \ln p$ term in the numerator of (14.25) to the information form of the entropy.

It is possible to show that D_2 as defined by (14.21) is the same as the mass dimension defined in (14.1) and the correlation dimension D_c . That is, box counting gives D_0 and correlation functions give D_2 (cf. Sander et al. 1994).

There are many objects in nature that have similar fractal dimensions, but which nevertheless differ in appearance. An example of this difference is the visual appearance in three spatial dimensions of the clusters generated by diffusion-limited aggregation and the percolation clusters generated by the Leath algorithm at the percolation threshold. (Both objects have a fractal dimension of approximately 2.5.) In some cases this difference can be accounted for by *multifractal* properties of an object. For objects called *multifractals* the various D_q are different, in contrast to *monofractals* for which the different measures are the same. Percolation clusters are an example of a monofractal, because $p_i \sim \ell^{D_0}$, the number of boxes $N(\ell) \sim \ell^{-D_0}$, and from (14.21), $D_q = D_0$ for all q . Multifractals occur when the quantities p_i are not the same throughout the object, as frequently happens for the strange attractors produced by chaotic dynamics. DLA might be an example of a multifractal, and the appropriate probabilities p_i might correspond to the probability that the next perimeter site to be occupied is at i .

14.6 Projects

Although the kinetic growth models yield beautiful pictures and fractal objects, there is much we do not understand. Why do the fractal dimensions have the values that we found by various numerical experiments? Can we trust our numerical estimates of the various exponents or is it necessary to consider much larger systems to obtain their true asymptotic values? Can we find unifying features for the many kinetic growth models that presently exist? What is the relation of the various kinetic growth models to physical systems? What are the essential quantities needed to characterize the geometry of an object?

One of the reasons that growth models are difficult to understand is that typically the end product depends on the history of the growth. We say that these models are examples of “nonequi-

librium behavior.” This combination of simplicity, beauty, complexity, and relevance to many experimental systems suggests that the study of fractal objects will continue to involve a wide range of workers in many disciplines.

Project 14.16. Off-lattice DLA

- a. In the continuum (off-lattice) version of diffusion-limited aggregation, the diffusing particles are assumed to be disks of radius a . A particle executes a random walk until its center is within a distance $2a$ of the center of a particle already attached to the DLA cluster. At each step the walker changes its position by $(r \cos \theta, r \sin \theta)$, where r is the step size, and θ is a random variable between 0 and 2π . Modify your DLA program or [Program dla](#) to simulate off-lattice DLA.
- b. Compare the appearance of an off-lattice DLA cluster with one generated on a square lattice. It is necessary to grow very large clusters (approximately 10^6 particles) to see any real differences.
- c. Use the mass dimension to estimate the fractal dimension of the off-lattice DLA cluster and compare its value with the value you found for the square lattice. Explain why the fractal dimensions might be different. $D \approx 1.71$ for off-lattice DLA in two dimensions while $D \approx 1.55$ for a square lattice (also see Problem 14.9). However, it is necessary to grow very large clusters to determine the effect of the lattice.

Project 14.17. More efficient simulation of DLA

In [Program dla](#) we use a variable step size for the walkers to improve the efficiency of the algorithm. However, when the walker is within the distance R_0 of the seed, no optimization is used. Because there can be a great deal of empty space within this distance, we describe an additional optimization technique (see Ball and Brady). The basic idea is to use a simple geometrical object (a circle or square) centered at the walker such that none of the cluster is within the object. Then in one step move the walker to the perimeter of the object. For a circle the walker can move to any location with equal probability on the circle. For the square you need the probability of moving to various locations on the square. The major difficulty is to find the largest object that does not contain a part of the DLA cluster. To do this we consider coarse grained lattices. For example, each 2×2 group of sites on the original lattice corresponds to one site on the coarser lattice, and then each 2×2 group of sites on the coarse lattice corresponds to a site on an even coarser lattice, etc. If a site is occupied then any coarse site made from this site also is occupied.

- a. Because we have considered DLA clusters on a square lattice, we use squares centered at a walker. First we must find the probability $p(\Delta x, \Delta y, s)$ that a walker centered on a square of length $l = 2s + 1$, will be displaced by the vector $(\Delta x, \Delta y)$. This probability can be computed by simulating a random walk starting at the origin and ending at the edge of the square. These simulations are then repeated for many walkers, and then for each value of s . $p(\Delta x, \Delta y, s)$ is the fraction of walkers that reached the position $(\Delta x, \Delta y)$. Determine $p(\Delta x, \Delta y, s)$ for $s = 1$ to 16. Store your results in a file.
- b. We next need to produce an array such that for a given value of s and a random number r between 0 and 1, we can quickly find $(\Delta x, \Delta y)$. To do so create four arrays. The first array lists the probability distribution determined from p in part (a) such that the values for $s = 1$ are listed first, then the values for $s = 2$, etc. Call this array p . For example, $p(1) = p(-1, -1, 1)$,

$p(2) = p(1) + p(-1, 0, 1)$, $p(3) = p(2) + p(-1, 1, 1)$, etc. Next create an array `start` that tells you where to start in the array `p` for each value of `s`. Then create the two arrays `dx(i)` and `dy(i)` which give the values of Δx and Δy corresponding to `p(i)`. To see how these arrays are used, consider a walker located at (x, y) , centered on a square of size $2s + 1$. First compute a random number r and find `i = start(s)`. If $p(i) > r$, then the walker moves to $(x + dx(i), y + dy(i))$. If not, increment i by unity and check again. Repeat until $p(i) > r$. Write a program to create these four arrays and store them in a file.

- c. Write a subroutine to determine the maximum value of the parameter s such that a square of size $2s + 1$ centered at the position of the walker does not contain any part of the DLA cluster. Use coarse grained lattices to do this determination more efficiently.
- d. Modify Program `dla` to incorporate the subroutine from part (c) and read in the arrays from part (b). How much faster is your modified program than the original Program `dla` for clusters of size 500 and 5000 particles? What is the largest cluster you can grow on your computer in one hour?
- e. Grow as large a cluster as you can. Is there any evidence for anisotropy? For example, does the cluster tend to extend further along the axes or along any other direction?

The following two projects introduce some of the important ideas associated with scaling.

Project 14.18. Scaling properties of the percolation cluster size distribution

- a. The scaling hypothesis for n_s , the number of percolation clusters of size s , near the percolation threshold p_c is

$$n_s = s^{-\tau} f_{\pm}(|p - p_c|^{1/\sigma} s), \quad (s \gg 1) \quad (14.27)$$

where the indices + and – refer to $p > p_c$ and $p < p_c$, respectively. The critical exponents τ and σ are the same above and below p_c . To understand the scaling form of n_s , first note that (14.27) implies that $n_s \sim s^{-\tau}$ at $p = p_c$ (compare with Table 13.1). We now show that the exponent σ can be related to ν and τ . Recall that the connectedness length ξ is given in terms of n_s by (see (13.11)):

$$\xi^2 = \frac{\sum_s s^2 n_s R_s^2}{\sum_s s^2 n_s}, \quad (14.28)$$

where R_s is the radius of gyration of the clusters. Close to p_c , the large clusters dominate the sum in (14.28). On length scales less than ξ , the clusters have no way of telling that the system is not at p_c and hence the large clusters are fractals with $R_s \sim s^{1/D}$. If we substitute this dependence and the scaling form (14.27) for n_s in (14.28), we obtain

$$\xi^2 \sim \frac{\sum_s s^{2-\tau+2/D} f_{\pm}(|p - p_c|^{1/\sigma} s)}{\sum_s s^{2-\tau} f_{\pm}(|p - p_c|^{1/\sigma} s)}. \quad (14.29)$$

For an infinite system the sums in (14.29) range from $s = 1$ to $s = \infty$. To calculate these sums, we transform them into integrals. For example, we can write the numerator of (14.29) as

$$\begin{aligned} \sum_{s=1}^{\infty} s^{2-\tau+2/D} f_{\pm}(|p - p_c|^{1/\sigma} s) &\sim \int_1^{\infty} s^{2-\tau+2/D} f_{\pm}(|p - p_c|^{1/\sigma} s) ds \\ &\propto (p - p_c)^{(\tau D - 3D - 2)/(D\sigma)} \int_{(p-p_c)^{1/\sigma}}^{\infty} x^{2-\tau+2/D} f_{\pm}(x) dx, \end{aligned} \quad (14.30)$$

where $x = (p - p_c)^{1/\sigma} s$. The denominator can be written in a similar form. If we assume that the integrands are nonsingular, then to lowest order in $(p - p_c)$, the lower integration limit can be set equal to zero. Show that we obtain

$$\xi^2 \sim |p - p_c|^{-2/(D\sigma)}. \quad (14.31)$$

Because $\xi \sim |p - p_c|^{-\nu}$, we obtain the desired relation between ν , σ , and D :

$$\nu = 1/(D\sigma). \quad (14.32)$$

If we write the argument $x = |p - p_c|^{1/\sigma} s$ as $x = s/\xi^D$, we can express the scaling form of n_s (14.27) in a more physical form:

$$n_s \sim s^{-\tau} f_{\pm}(s/\xi^D). \quad (14.33)$$

Equation (14.33) implies that n_s depends on s only through the ratio s/ξ^D or R_s/ξ . That is, the connectedness length represents the only important length near p_c . Show that the integrands in (14.30) are nonsingular if $2 - \tau > -1$ and fill in the missing algebra leading to (14.33).

- b. Let us try to verify the scaling form of n_s by generating percolation clusters of all sizes as we did in Chapter 13. We use the Leath algorithm to generate clusters of size s that are grown from a seed. Modify Program `perc_cluster` so that many clusters are generated and n_s is computed for a given input probability p . Remember that the number of clusters of size s that are grown from a seed is the product sn_s , rather than n_s itself (see Problem 14.3a). Run your program at $p = p_c \approx 0.592746$ and grow at least 100 clusters for a square lattice with $L \geq 61$. From (14.27) we see that at $p = p_c$, $n_s \sim s^{-\tau} f(0) \sim s^{-\tau}$. Hence a log-log plot of n_s versus s should give a straight line for $s \gg 1$ with a slope of $-\tau$. Estimate τ from your data. If time permits, use a bigger lattice and average over more clusters, and also estimate the accuracy of your estimate of τ .
- c. Determine n_s for at least three different values of p close to p_c , for example, 0.57, 0.58, and 0.61. Plot the product $s^\tau n_s$ versus the product $|p - p_c|^\sigma s$ using the value of τ found in part (b). Try various values of σ until your points for all values of p follow the same curve. Initially, try $\sigma = 0.4, 0.45$, and 0.5.
- d. The mean cluster size $S(p)$ is related to n_s by (see (13.5)):

$$S(p) = \frac{\sum_s s^2 n_s}{\sum_s sn_s}. \quad (14.34)$$

Note that $S(p)$ can be regarded as the second moment of n_s . Use arguments similar to those used in part (a) to show that $S(p) \sim |p - p_c|^{(\tau-3)/\sigma}$ for p near p_c . Since $S(p) \sim |p - p_c|^{-\gamma}$, we have the relation

$$\gamma = (3 - \tau)/\sigma. \quad (14.35)$$

Use the values of τ and σ that you found in parts (b) and (c) to estimate γ .

Similar arguments can be made for P_∞ . Every site in the lattice is either empty with probability $1 - p$, or occupied and part of the spanning cluster with probability pP_∞ , or occupied but not part of the spanning cluster with probability $p(1 - P_\infty) = \sum_s sn_s$. Hence we have the exact relation

$$P_\infty(p) = 1 - \frac{1}{p} \sum_s sn_s. \quad (14.36)$$

Note that P_∞ can be regarded as the first moment of n_s . Hence using the same argument that led to (14.35), we find

$$\beta = (\tau - 2)/\sigma. \quad (14.37)$$

Compare this relation to the form of (14.35). A careful derivation of this scaling relation can be found on page 70 of Bunde and Havlin. Use (14.37) to estimate the critical exponent β .

Project 14.19. Dynamical scaling properties of the cluster size distribution in cluster-cluster aggregation

- a. The dynamical scaling assumption for the number of clusters of size s at time t for cluster-cluster aggregation is

$$n_s(t) \sim s^{-\theta} f(s/t^z), \quad (14.38)$$

where θ and z are exponents and $f(x)$ is a scaling function. The density of the particles is fixed and is related to n_s by the normalization condition:

$$\rho = \sum_s sn_s(t) \sim \int sn_s(t) ds. \quad (14.39)$$

Use the normalization condition (14.39) to show that $\theta = 2$. The scaling form (14.38) is expected to be applicable in the low density limit at large s and t .

- b. Show that the mean cluster size $S(t)$ given by

$$S(t) = \frac{\sum_s s^2 n_s(t)}{\sum_s sn_s(t)} \quad (14.40)$$

diverges for $t \rightarrow \infty$ as

$$S(t) \sim t^z \quad (14.41)$$

Hence the scaling form of $n_s(t)$ can be written as

$$n_s(t) \sim t^{-2} f(s/S(t)) \quad (14.42)$$

- c. Modify Program `cca` so that the cluster size distribution, $n_s(t)$, is computed for $t = 2^p$ with $p = 1, 2, 3 \dots$ and $s = 1, 3, 10, 30$, and 100. For simplicity, assume that the diffusion coefficient of the clusters is size independent. Remember that $n_s = N_s(t)/L^2$, where $N_s(t)$ is the number of clusters of s particles at time t . The unit of time can be defined in various ways. The easiest way is to increase t by unity after a cluster has been moved one lattice unit. However, this choice would lead to the time changing faster when the number of clusters decreases. A better choice is to increase the time by an amount $\Delta t = s/N$, where s is the number of sites in the selected cluster and N is the (original) number of particles in the lattice. Choose $L = 50$ and $N = 500$ and average over at least ten runs. One way to test the dynamical scaling form of $n_s(t)$ is to plot $s^2 n_s(t)$ versus s/t^z for different choices of z . Another way is to make a log-log plot of $S(t)$ versus t and extract the exponent z from the slope of the linear portion of the log-log plot.
- d. Repeat part (c) for other values of L and N and discuss the accuracy of your results.
- e. A similar type of dynamics occurs in one dimension. What do you think the fractal dimension of the final cluster would be? In one dimension the surface of a cluster is two sites regardless of its size. As a result, the large clusters do not grow as fast as they do for higher dimensions. Can $n_s(t)$ still be described by a scaling form?

References and Suggestions for Further Reading

We have considered only a few of the models that lead to self-similar patterns. Use your imagination to design your own model of real-world growth processes. You are encouraged to read the research literature and recent books on growth models.

R. C. Ball and R. M. Brady, "Large scale lattice effect in diffusion-limited aggregation," *J. Phys. A* **18**, L809 (1985). The authors discuss the optimization algorithm used in Project 14.17.

Albert-László Barabási and H. Eugene Stanley, *Fractal Concepts in Surface Growth*, Cambridge University Press (1995).

K. S. Birdi, *Fractals in Chemistry, Geochemistry, and Biophysics*, Plenum Press (1993).

Armin Bunde and Shlomo Havlin, editors, *Fractals and Disordered Systems*, Springer-Verlag (1991).

Fereydoon Family and David P. Landau, editors, *Kinetics of Aggregation and Gelation*, North-Holland (1984). A collection of research papers that give a wealth of information, pictures, and references on a variety of growth models.

Fereydoon Family, Daniel E. Platt, and Tamás Vicsek, "Deterministic growth model of pattern formation in dendritic solidification," *J. Phys. A* **20**, L1177 (1987). The authors discuss the nature of Laplace fractal carpets.

Fereydoon Family and Tamás Vicsek, editors, *Dynamics of Fractal Surfaces*, World Scientific (1991). A collection of reprints.

Fereydoon Family, Y. C. Zhang, and Tamás Vicsek, "Invasion percolation in an external field: dielectric breakdown in random media," *J. Phys. A* **19**, L733 (1986).

- Jens Feder, *Fractals*, Plenum Press (1988). This text discusses the applications as well as the mathematics of fractals.
- J.-M. Garcia-Ruiz, E. Louis, P. Meakin, and L. M. Sander, editors, *Growth Patterns in Physical Sciences and Biology*, NATO ASI Series B304, Plenum (1993).
- J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods*, Methuen (1964). The chapter on percolation processes discusses a growth algorithm for percolation.
- Shlomo Havlin and Daniel Ben-Avraham, "Diffusion in disordered media," *Adv. Phys.* **36**, 695 (1987).
- H. J. Herrmann, "Geometrical Cluster Growth Models and Kinetic Gelation," *Phys. Repts.* **136**, 154 (1986).
- Robert C. Hilborn, *Chaos and Nonlinear Dynamics*, Oxford University Press (1994).
- Benoit B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freeman (1983). An influential and beautifully illustrated book on fractals.
- Imtiaz Majid, Daniel Ben-Avraham, Shlomo Havlin, and H. Eugene Stanley, "Exact enumeration approach to random walks on percolation clusters in two dimensions," *Phys. Rev. B* **30**, 1626 (1984).
- P. Meakin, "The growth of rough surfaces and interfaces," *Physics Reports* **235**, 189 (1993). The author has written many seminal articles on DLA and other aggregation models.
- Paul Meakin, *Fractals, Scaling and Growth Far From Equilibrium*, Cambridge University Press (1995).
- L. Niemeyer, L. Pietronero, and H. J. Wiesmann, "Fractal dimension of dielectric breakdown," *Phys. Rev. Lett.* **52**, 1033 (1984).
- H. O. Peitgen and P. H. Richter, *The Beauty of Fractals*, Springer-Verlag (1986).
- Luciano Pietronero and Erio Tosatti, editors, *Fractals in Physics*, North-Holland (1986). A collection of research papers, many of which are accessible to the motivated reader.
- Mark Przyborowski and Mark van Woerkom, "Diffusion of many interacting random walkers on a three-dimensional lattice with a personal computer," *Eur. J. Phys.* **6**, 242 (1985). This work was done while the authors were high school students in West Germany.
- F. Reif, *Fundamentals of Statistical and Thermal Physics*, McGraw-Hill (1965). Einstein's relation between the diffusion and mobility is discussed in Chapter 15.
- John C. Russ, *Fractal Surfaces*, Plenum Press (1994). A disk also is included.
- Evelyn Sander, Leonard M. Sander, and Robert M. Ziff, "Fractals and Fractal Correlations," *Computers in Physics* **8**, 420 (1994). An introduction to fractal growth models and the calculation of their properties. One of the authors, Leonard Sander, is a co-developer of the diffusion limited aggregation model (see Problem 14.9).

H. Eugene Stanley and Nicole Ostrowsky, editors, *On Growth and Form*, Martinus Nijhoff Publishers, Netherlands (1986). A collection of research papers at approximately the same level as the Family and Landau collection. The article by Paul Meakin on DLA was referenced in the text.

Hideki Takayasu, *Fractals in the Physical Sciences*, John Wiley & Sons (1990).

David D. Thornburg, *Discovering Logo*, Addison-Wesley (1983). The book is more accurately described by its subtitle, *An Invitation to the Art and Pattern of Nature*. The nature of recursive procedures and fractals are discussed using many simple examples.

Donald L. Turcotte, *Fractals and Chaos in Geology and Geophysics*, Cambridge University Press (1992).

Tamás Vicsek, *Fractal Growth Phenomena*, second edition, World Scientific Publishing (1991). This book contains an accessible introduction to diffusion limited and cluster-cluster aggregation.

Bruce J. West, *Fractal Physiology and Chaos in Medicine*, World Scientific Publishing (1990).

David Wilkinson and Jorge F. Willemsen, “Invasion percolation: a new form of percolation theory,” *J. Phys. A* **16**, 3365 (1983).

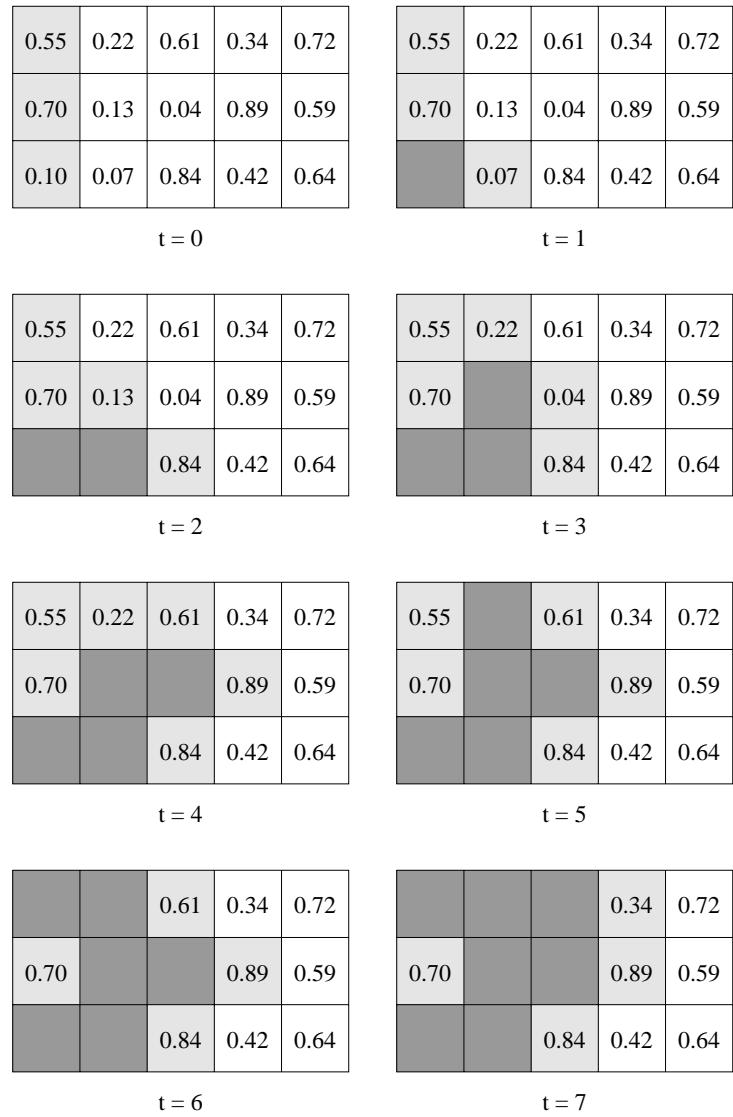


Figure 14.10: Example of a cluster formed by invasion percolation. The lattice at $t = 0$ shows the random numbers that have been assigned to the sites. The darkly shaded sites are occupied by the invader that occupies the perimeter site (lightly shaded) with the smallest random number.

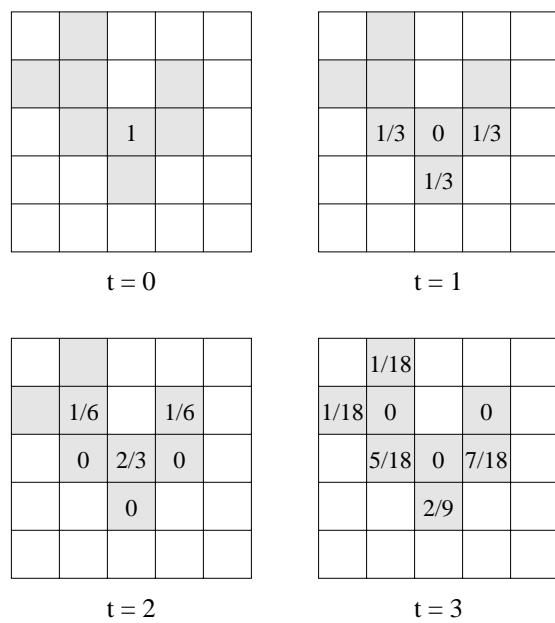


Figure 14.11: The time evolution of the probability distribution function $W_t(i)$ for three successive time steps.



Figure 14.12: An example of a DLA cluster of 1000 particles on a square lattice.

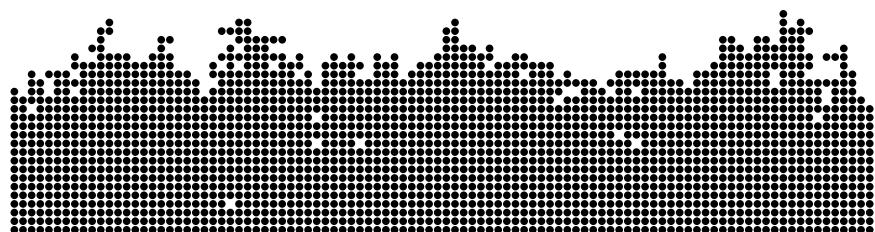


Figure 14.13: Surface growth according to the Eden model. The surface site in column x is the perimeter site with the maximum value h_x in the vertical direction. The average height for this surface is 20.46 and the width is 2.33.

Chapter 15

Complexity

We introduce cellular automata models, neural networks, genetic algorithms, and explore the concepts of self-organization and complexity.

15.1 Cellular Automata

Part of the fascination of physics is that it allows us in many cases to reduce natural phenomena to a few simple laws. It is perhaps even more fascinating to think about how a few simple laws can produce the enormously rich behavior that we see in nature. In this chapter we will discuss several models that illustrate some of the new ideas that are emerging from the study of “complex systems.”

The first class of models we discuss are known as *cellular automata*. Cellular automata were originally introduced by von Neumann and Ulam in 1948 as an idealization of biological self-reproduction, and are examples of discrete dynamical systems that can be simulated exactly on a digital computer. A cellular automaton can be thought of as a checkerboard with colored squares (the cells). Each cell changes its color at the tick of an external clock according to a rule based on the present configuration (microstate) of the cells in its neighborhood.

More formally, cellular automata are mathematical idealizations of dynamical systems in which space and time are discrete and the quantities of interest have a finite set of discrete values that are updated according to a local rule. The important characteristics of cellular automata include the following:

1. Space is discrete, and there is a regular array of sites (cells). Each site has a finite set of values.
2. Time is discrete, and the value of each site is updated in a sequence of discrete time steps.
3. The rule for the new value of a site depends only on the values of a *local* neighborhood of sites near it.

t:	111	110	101	100	011	010	001	000
t + 1:	0	1	0	1	1	0	1	0

Figure 15.1: Example of a local rule for the time evolution of a one-dimensional cellular automaton. The variable at each site can have values 0 or 1. The top row shows the $2^3 = 8$ possible combinations of three sites. The bottom row gives the value of the central site at the next time step. This rule is termed 01011010 in binary notation (see the second row), the modulo-two rule, or rule 90. Note that 90 is the base ten (decimal) equivalent of the binary number 01011010, that is, $90 = 2^1 + 2^3 + 2^4 + 2^6$.

4. The variables at each site are updated *simultaneously* (“synchronously”) based on the values of the variables at the previous time step.

Because the original motivation for studying cellular automata was their biological aspects, the lattice sites frequently are referred to as cells. More recently, cellular automata have been applied to a wide variety of physical systems ranging from fluids to galaxies. We will refer to sites rather than cells, except when we are explicitly discussing biological systems.

We first consider one-dimensional cellular automata with the neighborhood of a given site assumed to be the site itself and the sites immediately to the left and right of it. Each site also is assumed to have two states (a Boolean automata). An example of such a rule is illustrated in Fig. 15.1, where we see that a rule can be labeled by the binary representation of the update for each of the eight possible neighborhoods and by the base ten equivalent of the binary representation. Because any eight digit binary number specifies an one-dimensional cellular automata, there are $2^8 = 256$ possible rules.

Program ca1 takes as input the decimal representation of the rule and produces the rule matrix (array update). This array is used to update each site on the lattice using periodic boundary conditions. On a single processor computer, it is necessary to use an additional array so that the state of each site can be updated using the previous values of the sites in its local neighborhood. The state of the sites as a function of time is shown on the screen with time running downwards.

```

PROGRAM ca1
! one-dimensional Boolean cellular automata
DIM update(0 to 7),site(0 to 501)
CALL setrule(update())
CALL initial(site(),L,tmax,#2)
CALL iterate(site(),L,update(),tmax,#2)
END

SUB setrule(update())
INPUT prompt "rule number = ": rule
OPEN #1: screen 0,0.5,0.2,0.8
SET BACKGROUND COLOR "black"
SET COLOR "white"
FOR i = 7 to 0 step -1

```

```

LET update(i) = int(rule/2^i) ! find binary representation
LET rule = rule - update(i)*2^i
LET bit2 = int(i/4)
LET bit1 = int((i - 4*bit2)/2)
LET bit0 = i - 4*bit2 - 2*bit1
! show possible neighborhoods
PRINT using "#": bit2,bit1,bit0;
PRINT " ";
NEXT i
PRINT
FOR i = 7 to 0 step -1
    PRINT using "##": update(i); ! print rules
    PRINT " ";
NEXT i
CLOSE #1
END SUB

SUB initial(site(),L,tmax,#2)
RANDOMIZE
OPEN #2: screen 0.5,1,0.1,0.9
ASK PIXELS px,py
SET WINDOW 1,px,py,1
SET COLOR "yellow"
LET L = 2*int(px/8) - 8
LET tmax = L
LET site(L/2) = 1 ! center site
BOX AREA 1+2*L,2*L+4,1,4 ! each site 4 x 4 pixels
END SUB

SUB iterate(site(),L,update(),tmax,#2)
! update lattice
! need to introduce additional array, sitenew, to temporarily
! store values of newly updated sites
DIM sitenew(0 to 501)
FOR t = 1 to tmax
    FOR i = 1 to L
        LET index = 4*site(i-1) + 2*site(i) + site(i+1)
        LET sitenew(i) = update(index)
        IF sitenew(i) = 1 then BOX AREA 1+i*4,i*4+4,1+t*4,t*4+4
    NEXT i
    MAT site = sitenew
    LET site(0) = site(L) ! periodic boundary conditions
    LET site(L+1) = site(1)
NEXT t
END SUB

```

The properties of all 256 one-dimensional cellular automata have been cataloged (see Wolfram). We explore some of the properties of one-dimensional cellular automata in Problems 15.1 and 15.2.

Problem 15.1. One-dimensional cellular automata

- a. Use **Program ca1** and rule 90 shown in Fig. 15.1. This rule also is known as the “modulo-two” rule, because the value of a site at step $t + 1$ is the sum modulo 2 of its two neighbors at step t . Choose the initial configuration to be a single nonzero site (seed) at the midpoint of the lattice. It is sufficient to consider the time evolution for approximately twenty steps. Is the resulting pattern of nonzero sites self-similar? If so, characterize the pattern by a fractal dimension.
- b. Consider the properties of a rule for which the value of a site at step $t + 1$ is the sum modulo 2 of the values of its neighbors *plus* its own value at step t . This rule is termed rule 10010110 or rule $150 = 2^1 + 2^2 + 2^4 + 2^7$. Start with a single seed site.
- c. Choose a random initial configuration for which the independent probability for each site to have the value 1 is 50%; otherwise, the value of a site is 0. Consider the time evolution of rule 90, rule 150, rule $18 = 2^1 + 2^4$ (00010010), rule $73 = 2^0 + 2^3 + 2^6$ (01001001), and rule 136 (10001000). How sensitive are the patterns that are formed to changes in the initial conditions? Does the nature of the patterns depend on the use or nonuse of periodic boundary conditions?

Because the dynamical behavior of many of the 256 one-dimensional Boolean cellular automata is uninteresting, we also consider one-dimensional Boolean cellular automata with larger neighborhoods. The larger neighborhood implies that there are many more possible update rules, and it is convenient to place some reasonable restrictions on the rules. First, we assume that the rules are symmetric, for example, the neighborhood 100 produces the same value for the central site as 001. We also assume that the zero neighborhood 000 yields 0 for the central site, and that the value of the central site depends only on the sum of the values of the sites in the neighborhood, for example, 011 produces the same value for the central site as 101 (Wolfram 1984).

A simple way of coding the rules that is consistent with these requirements is as follows. Call the size of the neighborhood z if the neighborhood includes $2z + 1$ sites. Each rule is labeled by $\sum_m a_m 2^m$, where $a_m = 1$ if the central cell is 1 when the sum of all values in the neighborhood equals m ; else $a_m = 0$. As an example, take $z = 2$ and suppose that the central site equals one when two or four sites are unity. This rule is labeled by $2^2 + 2^4 = 20$.

Problem 15.2. More one-dimensional cellular automata

- a. Modify **Program ca1** so that it incorporates the possible rules discussed in the text for a neighborhood of $2z + 1$ sites. How many possible rules are there for $z = 1$? Choose $z = 1$ and a random initial configuration, and determine if the long time behavior for each rule belongs to one of the following classes:
 - (a) A homogeneous state where every site is either 0 or 1. An example is rule 8.
 - (b) A pattern consisting of separate stable or periodic regions. An example is rule 4.
 - (c) A chaotic, aperiodic pattern. An example is rule 10.
 - (d) A set of complex, localized structures that may not live forever. There are no examples for $z = 1$.

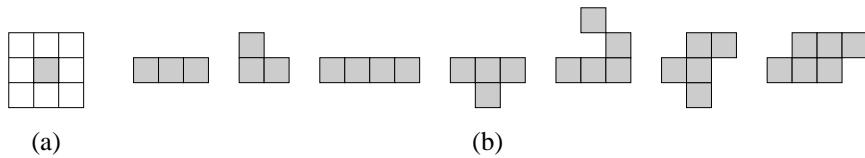


Figure 15.2: (a) The local neighborhood of a site is given by the sum of its eight neighbors. (b) Examples of initial configurations for the Game of Life, some of which lead to interesting patterns. Live cells are shaded.

- b. Modify your program so that $z = 2$. Wolfram (1984) claims that rules 20 and 52 are the only examples of complex behavior (class 4). Describe how the behavior of these two rules differs from the behavior of the other rules. Determine the fraction of the rules belonging to the four classes.
- c. Repeat part (b) for $z = 3$.
- d. Assume that sites can have three values, 0, 1, and 2. Classify the behavior of the possible rules for the case $z = 1$.

The results of Problem 15.2 suggest that an important feature of cellular automata is their capability for “self-organization.” In particular, the class of complex localized structures is distinct from regular as well as aperiodic structures. This intermediate structure is the focus of *complexity theory* whose goal is to explain complex phenomena in nature.

One-dimensional models are too limited to study the complexity of nature, and we now consider several two-dimensional models. The philosophy is the same except that the neighborhood contains more sites. Program `ca2` sets up the rule matrix and updates sites using the eight neighbor sites shown in Fig. 15.2a. There are now $2^9 = 512$ possible configurations for the eight neighbors and the center site, and 2^{512} possible rules. Clearly, we cannot go through all these rules in any systematic fashion as we did for one-dimensional cellular automata. For this reason, we will set up our rule matrix based on other considerations.

The rule matrix incorporated in Program `ca2` implements the best known two-dimensional cellular automata model: the *Game of Life*. This model, invented in 1970 by the mathematician John Conway, produces many fascinating patterns. The rules of the game are simple. For each cell determine the sum of the values of its four nearest and four next-nearest neighbors (see Fig. 15.2a). A “live” cell (value 1) remains alive only if this sum equals 2 or 3. If the sum is greater than 3, the cell will “die” (become 0) at the next time step due to overcrowding. If the sum is less than 2, the cell will die due to isolation. A dead cell will come to life only if the sum equals 3.

```

PROGRAM ca2
LIBRARY "csgraphics"
DIM update(0 to 511),cell(50,50)
CALL setrule(update(),L)
LET flag$ = ""
DO
```

```

CALL initial(cell(,),L)
DO
  CALL iterate(cell(,),update(),L)
  IF key input then
    GET KEY k
    IF (k = ord("s")) or (k = ord("S")) then
      LET flag$ = "stop"
    END IF
  END IF
  LOOP UNTIL k <> 0
  LET k = 0
LOOP until flag$ = "stop"
END

SUB setrule(update(),L)
  ! rule for Game of Life
  FOR i = 0 to 511
    LET update(i) = 0
  NEXT i
  ! three neighbors alive
  FOR nn1 = 0 to 5
    FOR nn2 = nn1+1 to 6
      FOR nn3 = nn2+1 to 7
        LET index = 2^nn1 + 2^nn2 + 2^nn3
        LET update(index) = 1 ! center dead
        LET update(index+256) = 1 ! center alive
      NEXT nn3
    NEXT nn2
  NEXT nn1
  ! two neighbors and center alive
  FOR nn1 = 0 to 6
    FOR nn2 = nn1+1 to 7
      LET index = 256 + 2^nn1 + 2^nn2
      LET update(index) = 1
    NEXT nn2
  NEXT nn1
  SET BACKGROUND COLOR "black"
  SET COLOR "white"
  INPUT prompt "lattice size = ": L
  CALL compute_aspect_ratio(L,xwin,ywin)
  SET WINDOW -0.2*xwin,1.2*xwin,-0.2*ywin,1.2*ywin
END SUB

SUB initial(cell(,),L)
  FOR i = 1 to L
    FOR j = 1 to L

```

```

LET cell(i,j) = 0
SET COLOR "yellow"
BOX AREA i,i+1,j,j+1
SET COLOR "black"
BOX LINES i,i+1,j,j+1
NEXT j
NEXT i
SET CURSOR 1,1
! click on cell to change its state or outside of lattice
! to update cells
SET COLOR "white"
PRINT "click on cell to toggle or outside of lattice to continue."
DO
    GET POINT x,y
    IF x > 1 and x < L and y > 1 and y < L then
        LET i = truncate(x,0)
        LET j = truncate(y,0)
        IF cell(i,j) = 0 then
            SET COLOR "black"
            BOX AREA i,i+1,j,j+1
            LET cell(i,j) = 1
        ELSE
            SET COLOR "yellow"
            BOX AREA i,i+1,j,j+1
            LET cell(i,j) = 0
            SET COLOR "black"
            BOX LINES i,i+1,j,j+1
        END IF
    END IF
LOOP until x < 1 or x > L or y < 1 or y > L
SET CURSOR 1,1
SET COLOR "white"
PRINT "Hit any key for new lattice, 's' to stop";
PRINT ""
END SUB

SUB iterate(cell(),update(),L)
    DIM cellnew(50,50)
    FOR i = 1 to L
        FOR j = 1 to L
            CALL neighborhood(cell(),i,j,sum,L)
            LET cellnew(i,j) = update(sum)
            IF cell(i,j) = 1 and cellnew(i,j) = 0 then
                SET COLOR "yellow"
                BOX AREA i,i+1,j,j+1
                SET COLOR "black"

```

```

        BOX LINES i,i+1,j,j+1
    ELSE IF cell(i,j) = 0 and cellnew(i,j) = 1 then
        SET COLOR "black"
        BOX AREA i,i+1,j,j+1
    END IF
NEXT j
NEXT i
MAT cell = cellnew
END SUB

SUB neighborhood(cell(),i,j,sum,L)
    LET ip = i + 1
    LET im = i - 1
    LET jp = j + 1
    LET jm = j - 1
    IF i = 1 then
        LET im = L
    ELSE IF i = L then
        LET ip = 1
    END IF
    IF j = 1 then
        LET jm = L
    ELSE IF j = L then
        LET jp = 1
    END IF
    LET sum = cell(i,jp) + 2*cell(i,jm) + 4*cell(im,j)
    LET sum = sum + 8*cell(ip,j)+ 16*cell(ip,jp) + 32*cell(ip,jm)
    LET sum = sum + 64*cell(im,jp) + 128*cell(im,jm) + 256*cell(i,j)
END SUB

```

Program ca2 allows the user to use any update rule by changing **SUB setrule**. **Program ca2** has not been optimized for the Game of Life and is written so that it can be easily modified for any cellular automata rule.

Problem 15.3. The Game of Life

- Program ca2** allows the user to determine the initial configuration interactively. Choose several initial configurations with a small number of live cells and investigate the different types of patterns that emerge. Some suggested initial configurations are shown in Fig. 15.2b. Does it matter whether you use fixed or periodic boundary conditions?
- Modify **Program ca2** so that each cell is initially alive with a 50% probability. What types of patterns typically result after a long time? What happens for 20% live cells? What happens for 70% live cells?
- Assume that each cell is initially alive with probability p . Given that the density of live cells at time t is $\rho(t)$, what is $\rho(t+1)$, the expected density at time $t+1$? Do the simulation and plot $\rho(t+1)$ versus $\rho(t)$. If $p=0.5$, what is the steady-state density of live cells?

- d. As we found in part (b), the system will develop structure even if each cell is randomly populated at $t = 0$. One measure of the increasing order in the system has been introduced by Schulman and Seiden and is analogous to the entropy in statistical mechanics. The idea is to divide the $L \times L$ system into boxes of linear dimension b and determine n_i , the number of live cells in the i th box. The quantity S is given by

$$S = \frac{1}{L^2} \log_2 \prod_{i=1}^{(L/b)^2} \binom{b^2}{n_i}. \quad (15.1)$$

The argument of the logarithm in (15.1) is the total number of microscopic states associated with a given sequence of n_i . Roughly speaking, S measures the extent to which live cells are correlated. Determine S as a function of time starting from a 50% random configuration. First consider $L = 50$ and $b = 2, 3, 4$, and 5. Average over at least ten separate runs. Describe how the behavior of the entropy depends on the level of “coarse graining” determined by the value of b . Does S decrease monotonically with time? Does it reach an equilibrium value? Increase L and the number of independent runs, and repeat your averages.

The Game of Life is an example of a universal computing machine. That is, we can set up an initial configuration of live cells to represent any possible program and any set of input data, run the Game of Life, and in some region of the lattice the output data will appear. The proof of this result (see Berlekamp et al.) involves showing how various configurations of cells represent the components of a computer including wires, storage, and the fundamental components of a CPU — the digital logic gates that perform **and**, **or**, and other logical and arithmetic operations.

Problem 15.4. Other two-dimensional cellular automata

- Consider a Boolean automaton with each site labeled by 1 (“on”) and 0 (“off”). We adopt an update rule such that the value of each site at time $t + 1$ is determined by the vote of its four nearest neighbors (on a square lattice) at time t . The update rule is that a site becomes on if 2, 3, or 4 of its four neighbors are on. Consider initial configurations for which 1-sites occur with probability p and 0-sites occur with probability $1 - p$. Because the voting rule favors the growth of 1-sites, it is interesting to begin with a minority of 1-sites. Choose $p = 0.1$ and determine what happens to isolated 1-sites. How do they grow initially? For what shape (convex or concave) does a cluster of 1-sites stop growing? What happens to clusters of 1-sites such as those shown in Fig. 15.3? (If necessary, create such a configuration.) Show that for $p = 0.1$, the system eventually freezes in a pattern made of 1-site rectangular islands in a sea of 0-sites. What happens for $p = 0.14$? Can you define a “critical density” p_c at which the behavior of the system changes? Consider square lattices with linear dimension $L = 128$ and $L = 256$ (see Vichniac).
- Suppose that the update rule is determined by the sum of the center cell and its eight nearest and next-nearest neighbors. If this sum equals 4 or more, then the center site equals 1 at the next time step (see Vichniac). This rule also favors the growth of 1-sites and leads to a phenomenon similar to that found in part (a). Consider an initial configuration for which 1-sites occur with probability p and 0-sites occur with probability $1 - p$. Choose $L = 128$ and show that for $p = 0.2$, the system eventually freezes. What is the shape of the 1-clusters? Show that if a single 0-site is changed to a 1-site at the surface of the largest 1-cluster, the cluster of

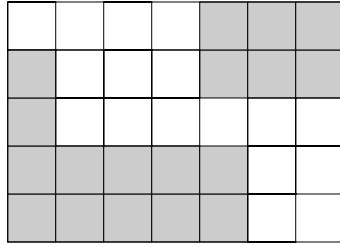


Figure 15.3: What is the evolution of these clusters using the rule discussed in Problem 15.4a. The shaded squares correspond to the 1-sites.

1-sites grows. What is the eventual state of the system? What is the behavior of the system for $p = 0.3$? Is it possible to define a “critical density” p_c such that for $p \geq p_c$, the growth of the 1-clusters continues until all sites change to the 1-state? Consider larger values of L and show that the value of p_c appears to be insensitive to the value of L . What is your estimated value of p_c in the limit of an infinite lattice?

- c. There is one problem with the conclusions that you were able to reach in parts (a) and (b) — they are incorrect! The finite lattice results are misleading and p_c is zero in the limit of an infinite lattice. The probability of *any* configuration of 1-sites is unity for an infinite lattice. That is, somewhere in the lattice there is a “critical cluster” that will grow indefinitely until all sites in the lattice change to the 1-state. The moral of the story is, “Do not trust a simulation without a theory” (a paraphrase of a quote usually attributed to Eddington).

**Problem 15.5.* Ising model cellular automata

- a. One of the most frequently studied models in statistical physics is the Ising model. In this model each cell has two states s_i that are labeled by ± 1 instead of 0 and 1. The energy is given by

$$E = -J \sum_{i,j=\text{nn}(i)}^N s_i s_j. \quad (15.2)$$

The notation $j = \text{nn}(i)$ indicates that j is a nearest neighbor of i . If we think of s_i as representing the magnetic moment or spin at cell i , then the Ising model can be understood as a model of magnetism with J being the strength of the interaction between nearest neighbor spins. For $J > 0$, the lowest energy state occurs when all the spins are either all up ($s_i = +1$) or all down ($s_i = -1$). The magnetization m per spin is given by

$$m = \frac{1}{N} \sum_{i=1}^N s_i. \quad (15.3)$$

Monte Carlo algorithms for the simulation of the Ising model are discussed in Chapters ?? and ??.

In the cellular automata implementation of the Ising model, the energy of the entire lattice is fixed, and a spin may flip only if the energy of the lattice does not change. Such a situation occurs if a spin has two up neighbors and two down neighbors. Hence the update rule is to flip the spin at i , that is, $s_i \rightarrow -s_i$, if it has precisely two up neighbors; otherwise do not change s_i . But because we want to update all sites simultaneously, we have a problem. That is, if two neighboring spins have opposite signs and each has a total of two up neighbors, then flipping both spins would change the total energy. Why? The trick is to divide the lattice into two kinds of sites corresponding to the red and black squares of a checkerboard. First we update simultaneously all the red squares (hence keeping their neighbors, the black squares fixed), and then we update simultaneously all the black squares. Implement this cellular automata update of the Ising model by modifying **Program ca2** and write a subroutine to compute the mean magnetization per spin and the total energy per spin E/N . (The latter should not change with time.) The average is over the different configurations generated by the cellular automata updates. Use periodic boundary conditions.

- b. Compute the mean magnetization as a function of E/N for a square lattice with $L = 20$. One way to generate an initial configuration is to let each spin be up with probability p . Allow the system to “equilibrate” before accumulating values of the magnetization. Does the mean magnetization change from being close to zero to being close to unity as E/N is lowered? If such a qualitative change occurs, we say that there is a phase transition. To improve your results, average over many different initial configurations with the same value of E/N and determine the dependence of the mean magnetization on E/N . Because the same value of p will occasionally lead to different initial energies, write a subroutine that flips spins until the desired energy is obtained.
- c. Repeat part (b) for a 40×40 lattice. Do your qualitative conclusions change?
- d. One difficulty with the cellular automata version of the Ising model is that it is not ergodic, that is, there are configurations with the same energy that cannot be reached for a given initial condition. In Chapter ?? we will see how to avoid this problem using Monte Carlo methods. Here we might allow the total energy E_0 to vary a little, say $\pm nJ$, where n is an integer. During a run we can periodically flip a spin at random such that the total energy is in the range $E_0 \pm nJ$. Try this method for different values of n . Do your results for the mean magnetization change significantly?

Program ca1 and **ca2** are inefficient due in part to the limitations imposed by the True BASIC language which does not have *bit* manipulation capability. The smallest element of computer memory contains a bit, which is a 0 or a 1. A *byte* is the size of memory needed to hold a single character, for example, a letter or a digit. More precisely, a byte is eight bits. Because there are $2^8 = 256$ possible arrangements of 1's and 0's in a byte, a byte can represent the ASCII character set, including all upper and lower case letters, numerals, punctuation, and other control characters such as a line feed. A computer *word* is usually two, four, or eight bytes, and is the unit of storage that can be accessed simultaneously and moved back and forth from the central processing unit (CPU) to various storage devices. If we could manipulate bits directly, then we could represent each site in a Boolean cellular automaton by a bit and update a whole word of sites (32 sites on a 32 bit machine) simultaneously. This type of update is a simple example of parallel processing on a single processor machine.

The C language has intrinsic bit manipulation operations; most Fortran 77 and all Fortran 90 compilers also have intrinsic functions that perform bit manipulation. One of the enticements of cellular automata is that they can run very quickly on parallel processors. Special processing boards exist for personal computers and workstations that run cellular automata models faster than on a general purpose supercomputer. A major interest in cellular automata is how they can be used as a general paradigm for massively parallel computation (cf. Toffoli and Margolus).

15.2 Lattice Gas Models of Fluid Flow

One of the most promising applications of cellular automata models is to simulations of fluid flow. Fluid flow is very difficult to simulate because the partial differential equation describing fluid flow, the Navier-Stokes equation, is nonlinear. As we have found, nonlinear equations can lead to the breakdown of standard numerical algorithms. In addition, there are typically many length scales that must be considered simultaneously. These length scales include the microscopic motion of the fluid particles, the length scales associated with fluid structures such as vortices, and the length scales of macroscopic objects such as pipes or obstacles. Because of all these considerations, simulations of fluid flow based on direct numerical solutions of the Navier-Stokes equation typically require sophisticated numerical methods (cf. Oran and Boris).

The cellular automata models of fluids are known as *lattice gas* models. These models are based on the idea that if we maintain the fundamental conservation laws and symmetries associated with fluids, then we can produce the correct physics at the macroscopic level if we average over many particles. In a lattice gas model the positions of the particles are restricted to the sites of a lattice and the velocities are restricted to a small number of velocity vectors. A microscopic model needs to include two processes, the free motion between collisions and the collisions. In the simplest models, the particles move freely to their nearest neighbor lattice site in one time step. Then the velocities of the particles at each lattice site are updated according to a collision rule that conserves mass, momentum, and kinetic energy. The free motion and the collisions for all sites are computed simultaneously.

To understand the nature of lattice gas models, it is easier to discuss a specific model. Three-dimensional models are being studied, but they are much more difficult to visualize and to understand theoretically, and we will consider only two-dimensional models. We assume a triangular lattice, because its symmetry is more closely related to that of a continuum than a square lattice. In addition, collision rules for square lattices do not typically mix the horizontal and vertical motions of the particles. All particles are assumed to have the same speed and mass. The possible velocity vectors lie only along the links connecting sites, and hence there are only six possible velocities (labeled 0 to 5 because the first bit in a computer byte is in the 0 position):

$$\begin{aligned} \mathbf{v}_0 &= (1, 0) & \mathbf{v}_1 &= (1, -\sqrt{3})/2 & \mathbf{v}_2 &= -(1, \sqrt{3})/2 \\ \mathbf{v}_3 &= (-1, 0) & \mathbf{v}_4 &= (-1, \sqrt{3})/2 & \mathbf{v}_5 &= (1, \sqrt{3})/2. \end{aligned}$$

In some models a rest particle also is allowed. Each site can have at most one particle moving in a particular direction. The update process proceeds by first moving all the particles in the direction of their velocity to a neighboring site. Then at each lattice site the velocity vectors are changed according to a collision rule. Particle number and kinetic energy are easily conserved because all particles have the same speed, and we need only insure momentum conservation. There

is some flexibility in the choice of rules. One set of collision rules is illustrated in Fig. 15.4. These rules are deterministic with only one possible set of velocities after a collision for each possible set of velocities before a collision.

Because True BASIC does not provide intrinsic bit manipulation functions, we do not list a program in True BASIC. Instead, we list a Fortran and C lattice gas program in Appendices ?? and ??, respectively. You can convert either program to True BASIC by using the fact that `mod(int(A/2xx (caret) N),2)` is the n th bit of the integer A. We can determine if there is a particle at site x,y with velocity v_2 by testing whether `mod(int(lat(x,y)/2 xx caret N),2)` is equal to 1. Or it is possible to introduce a string of eight characters, each of which can be 0 or 1 as a model for a computer byte. For example, the n th lattice site with two particles of velocity v_2 and v_4 can be represented by the string array `lat$(n) = "001010"`. The string segment of `astring` between the p th and q th character is given by `astring$[p:q]`. We can determine if there is a particle with velocity v_2 by testing whether `lat$(n)[3:3]` is equal to "1".

We now describe the procedure for storing information about the particles in an array of integers, `lat`. The eight bits in a byte are labeled from 0 to 7. In principle, we could use a four byte integer to update four sites simultaneously, but to avoid complications we will not do so. Instead each site of the lattice is represented by one element of `lat`. We use the first six bits from 0 to 5 of an integer to represent particles moving in the six possible directions with bit 0 corresponding to a particle moving with velocity v_0 . For example, if bit 0 equals unity, we know there is a particle at this site with velocity v_0 . If there are three particles with velocities v_0 , v_2 , and v_4 at a site, then this situation is represented by 00010101 in bit notation; the corresponding decimal equivalent is 21. From Fig. 15.4 we see that after the collision there are three particles with velocities v_1 , v_3 , and v_5 corresponding to 42. We can express this collision as `rule(21) = 42`. Similar decimal equivalents can be expressed for all the other possible collisions.

We reserve bit 6 for a possible rest particle. If we want to occupy a site with a fixed particle to represent a barrier, we use bit 7, that is, we set the value of barrier sites equal to $2^7 = 128$. What boundary condition should we use when a particle is adjacent to a barrier site and heading toward it? The simplest rule that insures that we do not lose any particles is to set the velocity v of such a particle equal to $-v$. Other possibilities are to set the angle of incidence equal to the angle of reflection or to set the velocity to an arbitrary value. The latter case corresponds to a rough surface and is more difficult to implement because we need to insure that no site has more than one particle with the same velocity.

An important use of lattice gas models is to simulate flow in and around various geometries. The fluid velocity field can be seen to develop vortices, wakes, and other fluid structures near obstacles. Typically, particles are injected at one end and absorbed as they reach the other end. Periodic boundary conditions are used in the other direction. Very large lattices are required to obtain quantitative results, because it is necessary to average the velocity over many sites. The density of the fluid is determined by the average number of particles per site, and the pressure can be varied by changing the flux of particles that are injected at one end. We discuss some typical applications in Problems 15.6–15.8.

Problem 15.6. Approach to equilibrium

- To maintain zero total velocity or momentum at all times, use an initial configuration where all sites contain either no particles or six particles whose net momentum is zero. The number density $\rho = 6N/(L_x L_y)$, where N is the number of sites with six particles and the total number of

sites is $L_x \times L_y$. Use `Program latgas.f` in Appendix ?? or `Program latgas.c` in Appendix ?? as a basis for your program for simulating a lattice gas. Use periodic boundary conditions in both directions. The output of the program should be a pictorial representation of the average velocity at each site, for example, an arrow pointing in the direction of the average velocity with a size proportional to the magnitude of the average velocity. Begin with a dilute gas with $N = 10$ on a 10×10 triangular lattice. Plot the velocity vector field after each iteration of the lattice gas to check that your program is working correctly.

- b. We now consider the approach to equilibrium. Use a 30×30 lattice and place six particles at every site in a 4×4 region. Describe qualitatively what happens to the particles as a function of time. Approximately how many time steps does it take for the particles to fill the box? Do the particles appear to be at random positions with random velocities? Describe your visual algorithm for determining when equilibrium has been reached.
- c. Repeat part (b) for an initial $b \times b$ region of particles with $b = 2, 6, 8$, and 10 . Estimate the equilibration time in each case. What is the qualitative dependence of the equilibration time on b ? How does the equilibration time depend on the number density ρ ?
- d. Repeat part (b) for an initial 4×4 region of particles, but vary the size of the box. Try $L_x = L_y = 10, 20$, and 40 . Estimate the equilibration time in each case. How does the equilibration time depend on the number density ρ ?

Problem 15.7. Flow past a barrier

- a. Modify your program from Problem 15.6 so that at each time step three particles are injected with velocities v_0 , v_1 , and v_5 at each site on the left-hand side. The particles are removed on the right-hand side. Include barrier sites in the middle of the cell representing a $b_x \times b_y$ rectangular barrier. Use the barrier boundary condition that the directions of the reflected particles are reversed, $\mathbf{v} \rightarrow -\mathbf{v}$. Use periodic boundary conditions in the vertical direction. Besides the left-hand column and the barrier sites, all other sites are initially empty. Represent the velocity field visually as described in Problem 15.6a.
- b. Choose $L_x = 50$ and $L_y = 20$ with a barrier of dimensions $b_x = 5$ and $b_y = 1$. Describe the flow once a steady state velocity field begins to appear. Can you see a wake appearing behind the obstacle? Are there vortices (circular fluid flow)?
- c. Repeat part (b) with different size obstacles. Are there any systematic trends?
- d. Reduce the pressure by injecting particles at the left every other time step. Are there any noticeable changes in behavior from parts (b) and (c)? Reduce the pressure still further and describe any changes in the fluid flow.
- e. Increase the size of the lattice by a factor of 10 in each direction and average the velocity in each 5×5 region. Compare the flow patterns that you obtain with those obtained in parts (b)–(d).

Problem 15.8. Fluid flow in porous media

- a. Modify your lattice gas program so that instead of a rectangular barrier, the barrier sites are placed at random in the lattice. Add a subroutine that sums the horizontal velocity of those

particles that reach the right edge of the lattice. The current density is the average of this sum per unit height of the lattice. Compute the current density as a function of porosity, the fraction of sites not containing barriers. If time permits, average over at least ten pore configurations for each value of the porosity. Use $L_x = 50$ and $L_y = 20$.

- b. Vary the size of the lattice and use the finite size scaling procedure discussed in Section 13.4 to estimate the critical exponent μ defined by the dependence of the current density J on the porosity ϕ . That is, $J \sim (\phi - \phi_c)^\mu$. Assume that you know the value of the percolation exponent ν defined by the critical behavior of the connectedness length $\xi \sim |p - p_c|^{-\nu}$ (see Table 13.1).

The principle virtues of lattice gas models are their use of simultaneous updating, which makes them very fast on parallel computers, and their use of integer or boolean arithmetic, which might be faster than floating point arithmetic. Their major limitation is that many sites must be averaged over to obtain quantitative results. It is not yet clear whether lattice gas models are more efficient than standard simulations of the Navier-Stokes equation. The greatest promise for lattice gas models may not be with simple single component fluids, but with multicomponent fluids such as binary fluids and fluids containing bubbles.

15.3 Self-Organized Critical Phenomenon

In nature we rarely see very large events such as a magnitude eight earthquake, an avalanche on a snow covered mountain, the sudden collapse of an empire (for example, the Soviet Union), or the crash of the stock market. When such rare events occur, are they due to some special set of circumstances or are they a part of a more general pattern of events that would occur without any specific external intervention? The idea of *self-organized criticality* is that in many cases very large events are part of a distribution of events and do not depend on special conditions or external forces.

If s represents the magnitude of an event, such as the energy released in an earthquake or the amount of snow in an avalanche, then a system is said to be *critical* if the number of events $N(s)$ follows a power law:

$$N(s) \sim s^{-\alpha}. \text{(no characteristic scale)} \quad (15.4)$$

One implication of the form (15.4) is that there is no characteristic scale. Systems whose correlation or distribution functions decay as power laws are said to be *scale invariant*. This terminology reflects the fact that power laws look the same on all scales. For example, the replacement $s \rightarrow bs$ in the function $N(s) = As^{-\alpha}$ yields a function $\tilde{N}(s)$ that is indistinguishable from $N(s)$, except for a change in the amplitude A by the factor $b^{-\alpha}$. If $\alpha \approx 1$, there would be one large event of size 1000 for every 1000 events of size one.

Contrast the power law dependence of $N(s)$ in (15.4) to the result of combining a large number of independently acting random events. In this case we know that the distribution of the sum is a Gaussian (see Problem 12.8) and that $N(s)$ has the form

$$N(s) \sim e^{-(s/s_0)^2}. \text{(characteristic scale)} \quad (15.5)$$

Scale invariance does not hold for functions that decay exponentially, because the replacement $s \rightarrow bs$ in the function $e^{-(s/s_0)^2}$ changes s_0 (the characteristic scale of s) by the factor b^2 . We note that for a power law distribution, there are events of all sizes, but for a Gaussian distribution, there are practically speaking no large events.

A common example of self-organized critical phenomena is an idealized sand pile. Suppose that we construct a sand pile by randomly adding one grain at a time onto a flat surface with open edges. Initially, the grains will stay more or less where they land, but after a while there will be small avalanches during which the grains move so that the local slope of the pile is not too large. Eventually, the pile reaches a statistically stationary (time-independent) state and the amount of sand added balances the sand that falls off the edge (on the average). When a single grain of sand is added to a configuration belonging to this state, a rearrangement might occur that triggers an avalanche of any size (up to the size of the system), so that the mean slope again equals the critical value. We say that the statistically stationary state is critical because there are avalanches of all sizes. The stationary state is self-organized because no external parameter (such as the temperature) needs to be tuned to force the system to this state. In contrast, the concentration of fissionable material in a nuclear chain reaction has to be carefully controlled for the nuclear chain reaction to be exactly critical.

The nature of self-organized critical phenomena can be understood by considering some simple models. We begin with a one-dimensional model based on the simplified behavior of a sand pile. Consider a lattice of L sites and let the height at each site be represented by the array element $h(i)$. One grain of sand is added to the left-most site, $h(1) = h(1) + 1$, at each time step. During this time step all the sites are checked to see if $h(i) - h(i+1) > 1$. All sites that satisfy this condition are marked for “toppling.” Next each marked site is toppled. A simple rule is to let $h(i) = h(i) - 1$ and $h(i+1) = h(i) + 1$, that is, the sand falls to the right. Any grains of sand that go beyond $i = L$ are lost forever. Program `sandpile` implements this simple model. We will find in Problem 15.9 that slightly more complicated rules are necessary to find nontrivial behavior.

```

PROGRAM sandpile
! simple one-dimensional sandpile model
DIM height(51),move(100),N(0:51)
CALL initial(height(),N(),L,sand$,#1,#2)
LET grain = 0
DO
    LET height(1) = height(1) + 1 ! add grain to first site
    WINDOW #1
    BOX SHOW sand$ at 1,height(1) - 1
    LET topple = 0           ! number of grains that topple
    LET grain = grain + 1   ! number of grains added
    DO
        CALL check(height(),L,move(),unstable)
        IF unstable = 1 then
            CALL slide(height(),L,move(),sand$,#1)
            LET topple = topple + 1
        END IF
    LOOP until unstable = 0

```

```

LET N(topple) = N(topple) + 1
CALL show_distribution(N(),L,grain,#2)
LOOP until key input
END

SUB initial(height(),N(),L,sand$,#1,#2)
  OPEN #1: screen 0,0.7,0,1
  INPUT prompt "lattice size L = ": L      ! suggest L = 20
  FOR i = 1 to L
    LET height(i) = 0
  NEXT i
  LET height(L+1) = 0          ! boundary condition
  SET WINDOW -1,L+1,-2,40
  SET COLOR "blue"
  BOX AREA 1.1,1.9,0.1,0.9
  BOX KEEP 1,2,0,1 in sand$
  CLEAR
  SET COLOR "black"
  PLOT LINES: 1,-0.05;L+1,-0.05
  ! initialize array
  FOR topple = 0 to 20
    LET N(topple) = 0
  NEXT topple
  OPEN #2: screen 0.72,1,0.4,1
  SET WINDOW -0.1,L+1,-0.05,1.01
  PLOT LINES: 0,-0.01;L+0.2,-0.01
  PLOT LINES: -0.05,-0.01;-0.05,1
END SUB

SUB check(height(),L,move(),unstable)
  LET unstable = 0
  FOR i = 1 to L
    IF height(i) - height(i+1) > 1 then
      LET move(i) = 1
      LET unstable = 1
    ELSE
      LET move(i) = 0
    END IF
  NEXT i
END SUB

SUB slide(height(),L,move(),sand$,#1)
  WINDOW #1
  FOR i = 1 to L
    IF move(i) = 1 then
      LET height(i) = height(i) - 1      ! sand topples

```

```

BOX CLEAR i,i + 1,height(i),height(i) + 1
IF i < L then
    ! next site receives grain
    LET height(i+1) = height(i+1) + 1
    BOX SHOW sand$ at i+1,height(i+1) - 1
END IF
END IF
NEXT i
END SUB

SUB show_distribution(N(),L,grain,#2)
WINDOW #2
! erase previous graph
SET COLOR "white"
BOX AREA 0,L+0.2,0,1
SET COLOR "black"
FOR topple = 0 to L
IF N(topple) > 0 then
    BOX AREA topple,topple+0.2,0,N(topple)/grain
END IF
NEXT topple
END SUB

```

Problem 15.9. One-dimensional sand piles

- Use `Program sandpile` with $L = 20$. Where is the sand added? What is the slope of the sand pile after a long time?
- `Program sandpile` computes the number of sites s that topple during each time step and plots the distribution of toppling sizes, $N(s)$, versus s . Modify the program so that $N(s)$ is computed only after the sand pile reaches a steady state. Is the behavior of $N(s)$ interesting for this model?
- Modify the rules so that a site which topples loses *two* grains of sand, one to its nearest neighbor to the right and the other to its next nearest neighbor to the right. Plot the distribution $N(s)$ versus s after steady state behavior is reached. Is there a range of values of s for which $N(s)$ shows power law behavior? If so, make the appropriate plot and estimate the critical exponent α .
- Introduce the variable $m(i) = h(i + 1) - h(i)$, and convince yourself that the same results are obtained by replacing $h(i)$ by $m(i)$ and using the rule $m(i) > 1$ for toppling. The variable $m(i)$ is called the local slope. Modify your program so that $m(i)$ is used instead of $h(i)$ and adopt the toppling rule used in part (c). Do your results change if you add a grain of sand at each time step at random anywhere in the lattice?
- Use the same rule that you considered in parts (c) and (d) and compute $N(s)$ for different values of L and systematically investigate the importance of finite size effects. Average $N(s)$ over many updates and obtain your best estimate for the critical exponent α .

In Problem 15.9 we saw that the fundamental variable is the local slope. Most sand pile models are described using this variable. In the literature many authors refer to the height when they really mean the local slope. In Problem 15.10 we explore the behavior of a two-dimensional model of a sand pile.

Problem 15.10. Two-dimensional sand pile

- a. Write a program to simulate a two-dimensional sand pile using the rule that a site topples if $m(i) > 3$. The pile is grown by choosing a site at random and adding one grain, that is, $m(i) \rightarrow m(i) + 1$. If site i topples (exceeds its critical value), it distributes four grains of sand to its four nearest neighbors (on a square lattice), that is, $m(i) \rightarrow m(i) - 4$. At the edges or corners, only three or two neighbors, respectively, are affected, and the sand that goes outside the boundary of the lattice is lost. Color code the sites according to their value of $m(i)$. In `Program sandpile` we checked all sites for stability, but in two dimensions such a check would take too much time if the size of the lattice were sufficiently large. For this reason we suggest that you maintain two arrays `posx` and `posy` which contain the x and y coordinates of those sites that need to be checked for stability. Each time a site topples, add its neighbors' positions to these arrays. Then remove the last site from the arrays and check its stability. Continue this process of removing sites and checking their stability, and adding neighbors to the array until there are no more sites to check.
- b. After the critical state has been reached, compute the number of sites s that topple in response to the addition of a single grain. Then compute the distribution of toppling sizes $N(s)$ and determine if $N(s)$ exhibits power law behavior. Begin with $L = 10$ and then consider larger values of L .
- c. Although the model in part (a) might seem reasonably realistic, it is of course over simplified. Laboratory experiments indicate that real sand piles show power law behavior if the piles are small, but larger sand piles do not (see Jaeger et al.). Modify the model to make its behavior more realistic.

Do model sand piles have anything in common with earthquakes? The Gutenberg-Richter law for $N(E)$, the number of earthquakes with energy release E , has been observed empirically and is consistent with power law behavior:

$$N(E) \sim E^{-b}, \quad (15.6)$$

with $b \approx 0.5$. The magnitude of earthquakes on the Richter scale is approximately the logarithm of the energy release. One implication of the power law dependence in (15.6) is that there is nothing special about large earthquakes. That is, if we could wait a couple of million years, we would likely observe earthquakes of size ten following the same Gutenberg-Richter law. In Problem 15.11 we explore the behavior of a model of tectonic plate motion which suggests that the Gutenberg-Richter law is a consequence of self-organized criticality.

Problem 15.11. Earthquake model

Given the long time scales between earthquakes and the complexity of the historical record, there is considerable interest in developing ways of studying earthquakes using simulations. The Burridge and Knopoff model of fault systems consists of a system of coupled masses in contact with a moving

rough surface. The masses are subjected to static and dynamic frictional forces, and also are pulled by an external force corresponding to slow tectonic plate motion. The major difficulty with this model and similar ones that have been proposed is that the numerical solution of the corresponding Newton's equations of motion is computationally intensive. For this reason we first study a simple cellular automaton model that retains some of the basic physics of the Burridge and Knopoff type models. A more realistic cellular automaton model is discussed in Project 15.19.

Define a real variable $F(i, j)$ on a square lattice, where F represents the force on the block at position (i, j) . The linear dimension of the lattice is L , and the number of sites N is given by $N = L^2$. The initial state of the lattice at time $t = 0$ is found by assigning small random values to $F(i, j)$. The lattice is updated according to the following rules:

1. Increase F everywhere by a small amount ΔF , for example, choose $\Delta F = 10^{-5}$, and increase the time t by 1. This increase represents the effect of the driving force due to the slow motion of the tectonic plate.
2. Check if $F(i, j)$ is greater than F_c , the critical threshold value of the force. If not, the system is stable and step 1 is repeated. If the system is unstable, go to step 3. Choose $F_c = 4$ for convenience.
3. The release of force due to slippage of a block is represented by letting $F(i, j) = F(i, j) - F_c$. The transfer of force is represented by updating the force at the sites of the four neighbors at $(i, j \pm 1)$ and $(i \pm 1, j)$: $F \rightarrow F + 1$.

As an example, let the linear dimension of the lattice $L = 10$ so that the number of sites $N = L^2 = 100$. Do the simulation and show that the system eventually comes to a statistically stationary state, where the average value of the force at each site stops growing. Monitor the distribution of the size s , where s is the total number of sites (blocks) that are affected by an instability. Increase L to $L = 30$ and repeat your simulations.

The behavior of some other simple models is explored in various contexts in the following four problems.

Problem 15.12. Forest fire model

- a. Consider the following simple model of the spread of a forest fire. Suppose that at $t = 0$ the $L \times L$ sites of a square lattice either have a tree or are empty with probability p_t and $1 - p_t$ respectively. Those sites which have a tree, are on fire with probability f . At each time step an empty site grows a tree with probability p , a tree that has a nearest neighbor site on fire catches fire, and a site that is already on fire dies and becomes empty. Note that the changes in each site occur synchronously, and that this model is an example of a probabilistic cellular automaton. Write a program to simulate this model and color code the three types of sites. Use periodic boundary conditions.
- b. Use $L \geq 30$ and determine the values of p for which the forest maintains fires indefinitely. Note that as long as $p > 0$, new trees will always grow.
- c. Choose the value of p that you found in part (b) and compute the distribution of the number of sites s_f on fire. If the distribution is critical, determine the exponent α that characterizes this

distribution. Also compute the distribution for the number of trees, s_t . Is there any relation between these two distributions?

- d. To obtain reliable results it is frequently necessary to average over many initial configurations. However, it is possible that the behavior of a system is independent of the initial configuration and averaging over many initial configurations is unnecessary. This latter possibility is called *self-averaging*. Repeat parts (b) and (c), but average your results over ten initial configurations. Is this forest fire model self-averaging?

Problem 15.13. Another forest fire model

- a. Consider a simple variation of the model discussed in Problem 15.12. At $t = 0$ each site is occupied by a tree with probability p_t ; otherwise, it is empty. The system is updated in successive time steps as follows:
- Randomly grow new trees at time t with a small probability p from sites that are empty at time $t - 1$;
 - A tree that is not on fire at $t - 1$ catches fire due to lightning with probability f ;
 - Trees on fire ignite neighboring trees, which in turn ignite their neighboring trees, etc. The spreading of the fire occurs instantaneously.
 - Trees on fire at time $t - 1$ die (become empty sites) and are removed at time t (after they have set their neighbors on fire);

As in Problem 15.12, the changes in each site occur synchronously. Determine $N(s)$, the number of clusters of trees of size s that catch fire in each time step. Two trees are in the same cluster if they are nearest neighbors.

- b. Do the simulation and determine if the behavior of $N(s)$ is consistent with $N(s) \sim s^{-\alpha}$. If so, estimate the exponent α for several values of p and f .
- c. The balance between the mean rate of birth and burning of trees in the steady state suggests a value for the ratio f/p at which this model is likely to be scale invariant. If the average steady state density of trees is ρ , then at each time step the mean number of new trees appearing is $pN(1 - \rho)$, where $N = L^2$ is the total number of sites. In the same spirit, we can say that for small f , the mean number of trees destroyed by lightning is $f\rho N\langle s \rangle$, where $\langle s \rangle$ is the mean number of trees in a cluster. Is this reasoning consistent with the results of your simulation? If we equate these two rates, we find that $\langle s \rangle \sim ((1 - \rho)/\rho)(p/f)$. Because $0 < \rho < 1$, it follows that $\langle s \rangle \rightarrow \infty$ in the limit $f/p \rightarrow 0$. Given the relation $\langle s \rangle = \sum_{s=1}^{\infty} sN(s)/\sum_s N(s)$ and the divergent behavior of $\langle s \rangle$, why does it follow that $N(s)$ must decay more slowly than exponentially with s ? This reasoning suggests that $N(s) \sim s^{-\alpha}$ with $\alpha < 2$. Is this expectation consistent with the results that you obtained in part (b)?

In this model there are three well separated time scales, that is, the time for lightning to strike ($\propto f^{-1}$), the time for trees to grow ($\propto p^{-1}$), and the instantaneous spreading of fire through a connected cluster. This separation of time scales seems to be an essential ingredient for self-organized criticality (cf. Grinstein and Jayaprakash).

**Problem 15.14.* Is “Life” critical?

Despite the simplicity of the rules of the Game of Life (see Problem ??), the dynamics of the game are not well understood. For example, if each cell is randomly populated at $t = 0$, how does the system develop structure and how can we characterize it? The existence of self-organized criticality in various cellular automata models has led several workers to investigate if similar phenomena exist in Life. The results are not yet definitive, and this problem will require considerable computational power and most likely some insight before it is solved. The idea is to begin with a random distribution ($p = 0.5$) of live cells and let the system evolve until only static or simple local periodic activity is found. Then a single dead cell is chosen at random and changed to a live cell. This change is analogous to adding a grain of sand to the sand pile model. The system is allowed to evolve until it again reaches a stable or periodic configuration. The quantities of interest include s , the total number of sites that are changed after the initial change, and T , the number of updates that are needed to return to a stable or periodic configuration. Then choose another dead cell at random, and repeat the above procedure. Compute $N(s)$ and $D(T)$, the distribution of s and T , respectively. If a site is changed several times after a perturbation, each change counts as part of the response. The difficult part of the program is determining whether a configuration is part of a periodic cycle. For small periods the lattice can be stored for a few times and then compared to previous configurations to check whether the state has repeated itself. It is very difficult to check for all types of periodic states, but if the system is started from a random distribution of live sites, cyclic structures with long periods are very rare. Another way of improving the efficiency is to change a dead cell to a live cell only if there is at least one live cell in its neighborhood (for example, the cell’s twenty nearest neighbors). In this way we do not waste time counting small avalanches, because if there are very few live cells in a neighborhood, then usually the system will return to a stable or periodic state very quickly.

Consider at least a 50×50 lattice with periodic boundary conditions. Are your results for $N(s)$ and $D(T)$ consistent with power law behavior? Consider progressively larger lattices and compute the mean values $\langle s \rangle$ and $\langle T \rangle$ in addition to $N(s)$ and $D(T)$. If $N(s)$ and $D(T)$ exhibit critical behavior, then the corresponding mean values would increase with the size of the lattice. Why? At present, some workers believe that $N(s)$ and $D(T)$ exhibit critical behavior, while others believe that this apparent behavior is an artifact resulting from the relatively small lattices that have been considered. (The largest lattices considered at present are 1024^2 .) Can you reach any tentative conclusions on the basis of your results?

Problem 15.15. Model of punctuated equilibrium

- a. The idea of *punctuated equilibrium* is that biological evolution occurs episodically rather than as a steady, gradual process. That is, most of the major changes in life forms occur in relatively short periods of time. Bak and Sneppen have proposed a simple model that exhibits some of the dynamical behavior expected of punctuated equilibrium. The model consists of a one-dimensional cellular automata of size L , where cell i represents the biological fitness of species i , normalized to unity. Initially, all cells receive a random fitness f_i between 0 and 1. Then the cell with the lowest fitness and its two nearest neighbors are randomly given new fitness values. This update rule is repeated indefinitely. Write a program to simulate the behavior of this model. Use periodic boundary conditions, and show the fitness of each cell as a bar of height f_i .
- b. Begin with $L = 64$ and describe what happens to the distribution of fitness values after a long

time. We can crudely think of the update process as replacing a species and its neighbors by three new species. In this sense the fitness represents a barrier to creating a new species. If the barrier is low, it is easier to create a new species. Do the low fitness species die out? What is the average value of fitness of the species after the model is run for a long time (10^3 , 10^4 , or more time steps)? Compute the distribution of fitness values, $N(f)$, averaged over all cells and over a long time. Allow the system to come to a fluctuating steady state before computing $N(f)$. Plot $N(f)$ versus f . Is there a critical value f_c below which $N(f)$ is much less than the values above f_c ? Is the update rule reasonable from a evolutionary point of view?

- c. Modify your program to compute the distance x between successive fitness changes and the distribution of these distances $C(x)$. Make a log-log plot of $C(x)$ versus x . Is there any evidence of self-organized criticality?
- d. Another way to visualize the results is to make a plot of the time at which a cell changed versus the position of the cell. Is the distribution of the plotted points approximately uniform? We might expect that the time of survival of a species depends exponentially on its fitness, and hence each update corresponds to an elapsed time of e^{-cf_i} , where the constant c sets the time scale and f_i is the fitness of the cell which has been changed. Choose $c = 100$ for convenience and make a similar plot with the time axis replaced by the logarithm of the time, that is, the quantity $100f_i$. Is this plot more meaningful?
- e. Another way of visualizing the meaning of punctuated equilibrium is to plot the number of times groups of cells change as a function of time. Divide the time into units of 100 updates and compute the number of fitness changes for cells $i = 1$ to 10 as a function of time. Do you see any evidence of punctuated equilibrium?

Stuart Kauffman has devised a cellular automaton model of genetics in which each site interacts with K neighbors through a randomly selected cellular automata rule. The K neighbors of each site also are randomly chosen at the beginning of a run. For $K = 4$, there are $2^4 = 16$ neighbor configurations and $2^{16} = 65536$ possible rules. In the Kauffman model each site is initially assigned one of the 65536 possible rules (for $K = 4$). After some time the system goes either to a limit cycle (repeating itself after a finite number of states) or to a fixed point (the state does not change). The idea is to test if a change of the update rule of a single site drives the entire system out of its steady state, that is, if a small mutation drastically changes the genetic behavior. For $K = 2$, simulations show that the genetic behavior withstands the mutation. However, for large K , for example, $K = 10$, a single mutation leads to a very different state. Perhaps, Nature is an example of a complex system where small local changes can lead to large global changes.

15.4 Neural Networks

Can computers think? This question has occupied philosophers, computer scientists, cognitive psychologists, and many others ever since the first computer was imagined. The assumption of workers in “strong” *artificial intelligence* is that it will be possible someday for computers to think. The reasoning is that thinking is based on symbolic manipulation of inputs from the external world. Of course, everyone agrees that we are far from having a computer think like a human. Some contend that computers will be able to only simulate the human brain and not be able to think

like it. Part of the argument is that the brain is not analogous to the hardware of a computer, and the mind is not analogous to its software.

Recent developments in two classes of models, known as neural networks and genetic algorithms, tend to weaken one argument against AI. These models are based on the idea that the program can change itself based on the inputs, that is, the program statements and its data are the “mind” of the computer, and the program and its data can change depending on its own internal state and outside inputs. Indeed, we should consider the entire memory of the computer to constitute its mind. The result is that the state of the computer can change itself in ways that the programmer cannot anticipate. As we have learned, simple algorithms can lead to complex and unpredictable outcomes, and it probably comes as no surprise that the state of the computer can evolve through a sequence of states that can be very complex, that is, neither completely random nor completely ordered. Perhaps, the mind exists at the edge of chaos where the complex behavior just begins.

Neural networks model a piece of this ultimate computer mind. The idea is to store memories so that a computer can recall them when inputs are given that are close to a particular memory. As humans we have our own algorithms for doing so. For example, if we see someone more than once, the person’s face might provide input that helps us recall the person’s name. In the same spirit, a neural network can be given a pattern, for example, a string of 0’s and 1’s, that partially reflect a previously memorized pattern. The network then attempts to recall the memorized pattern. The significant difference between what the computer usually does to retrieve data from its memory and the memory recall of a neural network is that in the latter we consider *content addressable memory* in contrast to computer programs themselves which retrieve memory based on the address or location of the data, not on its content.

Neural network models have been motivated by how neurons in the brain might collectively store and recall memories. It is known that a neuron “fires” once it receives electrical inputs from other neurons whose strength reaches a certain threshold. An important characteristic of a neuron is that its output is a nonlinear function of the sum of its inputs. Usually, a neuron is in one of two states, a resting potential (not firing) or firing at the maximum rate. The assumption is that when memories are stored in the brain, the strengths of the connections between neurons change. Neural network models attempt to maintain the key functions of biological neurons without the specific biological substrate.

We now consider an example of a neural network due to Hopfield. The network consists of N neurons and the state of the network is defined by the potential at each neuron, V_i . The strength of the connection between the i th and j th neuron is denoted by T_{ij} and is determined by the M stored memories:

$$T_{ij} = \sum_{s=1}^M (2V_i^s - 1)(2V_j^s - 1). \quad (15.7)$$

V^s is the state of the s th stored memory. Given an initial state V_i^0 , the dynamics of the network is simple, that is, choose a neuron i at random and change its state according to its input. Its input strength, S_i , is defined as

$$S_i = \sum_{i \neq j} T_{ij} V_j, \quad (15.8)$$

where V_j represent the current state of the j th neuron. Change the state of neuron i by setting

$$V_i = \begin{cases} 1, & \text{if } S_i > 0 \\ 0, & S_i \leq 0. \end{cases} \quad (15.9)$$

Note that the threshold value has been set equal to zero, but other values could be used as well.

Program `hopfield`, listed in the following, implements this model of a neural network and stores memories and recalls them based on user input.

```

PROGRAM hopfield
DIM T(50,50)
CALL memorize(T(,),N)
DO
    CALL recall(T(,),N)
LOOP
END

SUB memorize(T(,),N)
    DIM V(50)
    RANDOMIZE
    INPUT prompt "number of stored memories = ": M
    ! N corresponds to number of neurons
    INPUT prompt "size of memories = ": N
    PRINT "enter M strings of N 0's and 1's"
    FOR memory = 1 to M
        LINE INPUT s$
        CALL convert(s$,N,V())
        FOR i = 1 to N
            FOR j = 1 to N
                IF i <> j then
                    LET T(i,j) = T(i,j) + (2*V(i) - 1)*(2*V(j) - 1)
                END IF
            NEXT j
        NEXT i
    NEXT memory
    PRINT
END SUB

SUB recall(T(,),N)
    DIM V(50)
    PRINT "enter a string of N 0's and 1's"
    LINE INPUT s$
    CALL convert(s$,N,V())
    DO
        FOR k = 1 to N
            LET i = int(N*rnd) + 1      ! choose neuron at random

```

```

LET sum = 0
FOR j = 1 to N
    IF i <> j then LET sum = sum + T(i,j)*V(j)
NEXT j
IF sum > 0 then
    LET V(i) = 1      ! above threshold
ELSE
    LET V(i) = 0      ! below threshold
END IF
NEXT k
FOR i = 1 to N
    PRINT using "#": V(i);
NEXT i
PRINT
PAUSE 1
LOOP until key input
GET KEY kk
END SUB

SUB convert(s$,N,V())
    ! convert string to array of 0's and 1's
    FOR i = 1 to N
        LET c$ = s$[i:i]
        IF c$ = "0" then
            LET V(i) = 0
        ELSE
            LET V(i) = 1
        END IF
    NEXT i
END SUB

```

Problem 15.16. Memory recall in the Hopfield model

- Use **Program hopfield** to explore the ability of the Hopfield neural network to store and recall memories. Begin by storing $M = 2$ memories of $N = 20$ characters each. For example, store 1111100000000011111 and 11001100110011001100. Then try to recall a memory using the input string 1111111000000111111. This input is similar to the first memory. Record the “Hamming” distance between the final state and the closest memory, where the Hamming distance is the number of characters that differ between two strings. Repeat the above procedure for a number of different values of the number of memories M and the memory length N .
- Estimate how many memories can be stored for a given sized string before recall becomes severely reduced. Make estimates for $N = 10, 20$, and 30 . What criteria did you adopt for correct recall?
- Modify **Program hopfield** so that two-dimensional patterns can be memorized. Instead of a string of 1’s and 0’s, you will need a grid of $L \times L$ cells each of which can be on or off. The

major change in your program is that the indices i and j for S_i and T_{ij} must now refer to a particular cell. For example, if (x, y) is the location of the i th cell, then $i = x + (y - 1)L$. T_{ij} will be represented by an $L^2 \times L^2$ array, and V_i will correspond to an array of length L^2 . Also, you will need to write input and output subroutines to show the patterns. Consider a grid with $L \geq 10$ and store three patterns. Patterns could be simple geometric shapes or symbols. Then input a pattern similar to one of the stored memories and see how well the Hopfield algorithm is able to recall the correct pattern. Repeat for a number of different input patterns, and then increase the number of stored memories.

In addition to helping us understand biological neural networks, neural networks can be used to determine an optimal solution to a difficult optimization problem. In Problem 15.17 we consider the problem of finding the minimum energy of a model spin glass.

Problem 15.17. Minimum energy of an Ising spin glass

- We can define an energy for the Hopfield model in analogy to the Ising model:

$$E = -\frac{1}{2} \sum_i \sum_{j \neq i} T_{ij} V_i V_j, \quad (15.10)$$

where we assume that $T_{ij} = T_{ji}$. Note that the form of (15.10) is very similar to (15.2). If we give the T_{ij} random values, then the model is an example of a “spin glass” with long-range interactions (see Project ??). Modify your program so that the T_{ij} are given random values between -1 and 1 with $T_{ii} = 0$. The program should ask for an input string of N characters. Have the program display the output string and the energy after every N attempts to change a neuron. Begin with $N = 20$.

- Describe what happens to the energy after a long time. For different initial states, but the same set of T_{ij} , is the energy the same after the system has evolved for a long time? Explain your results in terms of the number of local energy minima.
- What is the behavior of the states? Do you find periodic behavior and/or random behavior or do the states evolve to a state that does not change?

15.5 Genetic Algorithms

There are many people who find it difficult to accept that evolution is sufficiently powerful to generate the biological complexity seen in nature. Part of this difficulty arises from the inability of humans to intuitively grasp time scales that are much greater than their own lifetimes. Another reason is that it is very difficult to appreciate how random changes can lead to emergent complex structures. Genetic algorithms provide one way of understanding the nature of evolution. Their principal utility at present is in optimization problems, but they also are being used to model biological and social evolution. One of the important examples is due to the biologist Tom Ray (see Lewin) who used a genetic algorithm in conjunction with low level machine coding. The memory of the computer was loaded with code segments which reproduce with small changes in their code. Because each code segment requires memory and the computer’s processing time, the code segments compete with one another for memory and time. In what was a surprise to many,

the memory of the computer, which initially contained a few simple code segments, evolved into a complicated “ecosystem” of code segments of many different sizes and structures.

The idea of genetic algorithms is to model the process of evolution by natural selection. This process involves two steps: random changes in the genetic code during reproduction, and selection according to some fitness criteria. In biological organisms the genetic code is stored in the DNA. We will store the genetic code as a string of 0's and 1's. (In Fortran and C the genetic code can be stored as an integer variable and bit manipulation can be used.) The genetic code constitutes the *genotype*. The conversion of this string to the organism or *phenotype* depends on the problem. The selection criteria is applied to the phenotype. First we describe how change is introduced into the genotype.

Typically, nature changes the genetic code in two ways. The most obvious, but least often used method, is *mutation*. Mutation corresponds to changing a character at random in the genetic code string from 0 to 1 or from 1 to 0. The second and much more powerful method is associated with sexual reproduction. We can take two strings, remove a piece from one string and exchange it with the same length piece from the other string. For example, if string $A = 0011001010$ and string $B = 0001110001$, then exchanging the piece from position 4 to position 7 leads to two new strings $A' = 0011110010$ and $B' = 0001001001$. This type of change is called *recombination* or *crossover*. At each generation we produce changes using recombination and mutation. We then select from the enlarged population of strings (including strings from the previous generation), a new population for the next generation. Usually, a constant population size is maintained from one generation of strings to the next.

We next have to choose a selection criterion. If we want to model an actual ecosystem, we can include a physical environment and other sets of populations corresponding to different species. The fitness could depend on the interaction of the different species with one another, the interaction within each species, and the interaction with the physical environment. In addition, the behavior of the populations might change the environment from one generation to the next. For simplicity, we will introduce the idea of genetic algorithms with a single population of strings, a simple phenotype, and a simple criteria for fitness.

The phenotype we consider is a variant of the Ising spins considered in Problems 15.5 and 15.17. We consider a square lattice of linear dimension L occupied by spins that have one of the two values $s_i = \pm 1$. The energy of the system is given by

$$E = - \sum_{i,j=\text{nn}(i)} T_{ij} s_i s_j, \quad (15.11)$$

where the sum is over all pairs of spins that are nearest neighbors. Note that the energy function in (15.11) assumes that only nearest neighbor spins interact, in contrast to the energy function in (15.10) which assumes that every spin interacts with every other spin. The coupling constants T_{ij} can be either +1 (the ferromagnetic Ising model), -1 (the antiferromagnetic Ising model), randomly distributed (a spin glass), or have some other distribution. We adopt the energy as a measure of fitness. If we assume that $|T_{ij}| = 1$, then the minimum energy equals $-2L^2$ and the maximum energy is $2L^2$. More precisely, we choose the combination $2L^2 - E$, a quantity that is always positive, as our measure of fitness, and take the probability of selecting a particular string with energy E for the next generation to be proportional to the fitness $2L^2 - E$.

How does a genotype become “expressed” as a phenotype? The genotypes consists of a list or

string of length L^2 with 1's and 0's. Lattice site (i, j) corresponds to the n th position in the string where $n = (j - 1)L + i$. If the character in the string at position n is 0, then the spin at site (i, j) equals -1 . If the character is 1, then the spin equals $+1$.

We now have all the ingredients we need to apply the genetic algorithm. Program `genetic` chooses a random size and a random position for the process of recombination, and periodic boundary conditions are applied to the string for the purpose of recombination. We use the True BASIC concatenation operator, `&`, to piece two strings together. For example, "Magic" `&` "and" `&` "Larry" is equivalent to "MagicandLarry". Note that the selection of the population for the new generation uses the method of discrete nonuniform probability distributions discussed in Section 11.5.

```

PROGRAM genetic
DIM s$(1000),T(400,2),Eselect(1000)
CALL initial(s$(),L,L2,T(),npop,nrecombine,nmutation,ngeneration)
FOR igeneration = 1 to ngeneration
    LET ntot = npop
    FOR iswap = 1 to nrecombine
        CALL recombine(s$(),L2,npop,ntot)
    NEXT iswap
    FOR i = 1 to nmutation
        CALL mutate(s$(),L2,npop,ntot)
    NEXT i
    CALL selection(s$(),L,L2,T(),npop,ntot,Eselect())
NEXT igeneration
CALL showoutput(s$(),npop,Eselect())
END

SUB initial(s$(),L,L2,T(),npop,nrecombine,nmutation,ngeneration)
    RANDOMIZE
    INPUT prompt "string (lattice) size = ": L
    LET L2 = L*L
    INPUT prompt "population number = ": npop
    ! L is linear dimension of phenotype
    INPUT prompt "number of recombinations per generation = ": nrecombine
    INPUT prompt "number of mutations per generation = ": nmutation
    INPUT prompt "number of generations = ": ngeneration
    ! create random population of genotypes
    FOR ipop = 1 to npop
        LET s$(ipop) = ""
        FOR i = 1 to L2
            IF rnd > 0.5 then
                LET s$(ipop) = s$(ipop) & "1"
            ELSE
                LET s$(ipop) = s$(ipop) & "0"
            END IF
        NEXT i
    END SUB

```

```

NEXT ipop
! create random bond network of Tij's
FOR i = 1 to L2
    FOR j = 1 to 2
        IF rnd > 0.5 then
            LET T(i,j) = 1
        ELSE
            LET T(i,j) = -1
        END IF
    NEXT j
NEXT i
END SUB

SUB recombine(s$(),L2,npop,ntot)
! choose two strings (genotypes) to recombine
LET i = int(npop*rnd) + 1
DO
    LET j = int(npop*rnd) + 1
LOOP until i <> j
LET size = int(rnd*(L2/2)) + 1
LET pos = int(rnd*L2) + 1
LET s1$ = s$(i)
LET s2$ = s$(j)
IF pos + size <= L2 then
    LET s$(ntot+1) = s1$[1:pos-1]&s2$[pos:pos+size]&s1$[pos+size+1:L2]
    LET s$(ntot+2) = s2$[1:pos-1]&s1$[pos:pos+size]&s2$[pos+size+1:L2]
ELSE
    ! apply periodic eboundary conditions
    LET pbc = pos + size - L2
    LET s$(ntot+1) = s2$[1:pbc] & s1$[pbc+1:pos-1] & s2$[pos:L2]
    LET s$(ntot+2) = s1$[1:pbc] & s2$[pbc+1:pos-1] & s1$[pos:L2]
END IF
LET ntot = ntot + 2
END SUB

SUB mutate(s$(),L2,npop,ntot)
LET i = int(rnd*npop) + 1
LET pos = int(rnd*L2) + 1
LET c$ = s$(i)[pos:pos]
IF c$ = "1" then
    LET c$ = "0"
ELSE
    LET c$ = "1"
END IF
LET s$(ntot + 1) = s$(i)[1:pos-1] & c$ & s$(i)[pos+1:L2]
LET ntot = ntot + 1
END SUB

```

```

SUB convert(a$,L,s(,))
  ! converts strings of 0's and 1's to 2D array of spins
  ! that is, genotype to phenotype
  FOR i = 1 to L
    FOR j = 1 to L
      LET n = (j-1)*L + i
      IF a$[n:n] = "1" then
        LET s(i,j) = 1
      ELSE
        LET s(i,j) = -1
      END IF
    NEXT j
  NEXT i
END SUB

SUB energy(L,s(,),T(,),E)
  LET E = 0
  FOR i = 1 to L
    LET ip = i + 1
    IF ip > L then LET ip = 1
    FOR j = 1 to L
      LET jp = j + 1
      IF jp > L then LET jp = 1
      LET n = (j-1)*L + i
      LET E = E - T(n,1)*s(i,j)*s(ip,j) - T(n,2)*s(i,j)*s(i,jp)
    NEXT j
  NEXT i
END SUB

SUB selection(s$(,) ,L,L2,T(,) ,npop,ntot,Eselect())
  DIM s(30,30),save$(1000),Elist(0:1000)
  LET Esum = 0
  LET Elist(0) = 0
  FOR i = 1 to ntot
    CALL convert(s$(i),L,s(,))
    CALL energy(L,s(,),T(,),E)
    LET Esum = Esum - E + 2*L2    ! contribution to Esum > 0
    LET Elist(i) = Esum
  NEXT i
  MAT save$ = s$
  ! select new population
  FOR ipop = 1 to npop
    LET E = Esum*rnd
    LET i = 0
    DO

```

```

LET i = i + 1
LOOP until E < Elist(i)    ! choose according to energy
LET s$(ipop) = save$(i)
LET Eselect(ipop) = Elist(i-1) - Elist(i) + 2*L2
NEXT ipop
END SUB

SUB showoutput(s$(),npop,Eselect())
FOR ipop = 1 to npop
    PRINT ipop,s$(ipop),Eselect(ipop)
NEXT ipop
PRINT
END SUB

```

Problem 15.18. Ground state of Ising-like models

- a. Use **Program genetic** to find the ground state of the ferromagnetic Ising model for which $T_{ij} = 1$ and the ground state energy is $E = -2L^2$. Choose $L = 4$, and consider a population of 20 strings, with 10 recombinations and 4 mutations per generation. How long does it take to find the ground state energy? You might wish to modify the program slightly so that each new generation is shown on the screen and a pause statement is added so that you can look at the new generations as they appear.
- b. Find the mean number of generations needed to find the ground state for $L = 4, 6$, and 8 . Repeat each run at least twice. Use a population of 100, a recombination rate of 50% and a mutation rate of 20%. Are there any general trends as L is increased? How do your results change if you double the population size? What happens if you double the recombination rate or mutation rate? Use larger lattices if you have sufficient computer resources.
- c. Repeat part (b) for the antiferromagnetic model.
- d. Repeat part (b) for the spin glass model where $T_{ij} = \pm 1$ randomly. In this case we do not know the ground state energy in advance. What criteria can you use to terminate a run?

One of the important features of the genetic algorithm is that the change in the genetic code is selected not in the genotype directly, but in the phenotype. Note that the way we change the strings (particularly with recombination) is not closely related to the two-dimensional lattice of spins. Indeed, we could have used some other prescription for converting our string of 0's and 1's to a configuration of spins on a two-dimensional lattice. If the phenotype is a three-dimensional lattice, we could use the same procedure for modifying the genotype, but a different prescription for converting the genetic sequence (the string of 0's and 1') to the phenotype (the three-dimensional lattice of spins). The point is that it is not necessary for the genetic coding to mimic the phenotypic expression. This point becomes distorted in the popular press when a gene is tied to a particular trait, because specific pieces of DNA rarely correspond directly to any explicitly expressed trait in the phenotype.

15.6 Overview and Projects

All of the models we have discussed in this chapter have been presented in the form of a computer algorithm rather than in terms of a differential equation. These models are an example of the development of a “computer culture” and are a reflection of the way that technology affects the way we think (cf. Vichniac). Can you discuss the models in this chapter without thinking about their computer implementation? Can you imagine understanding these models without the use of computer graphics?

We have given only a brief introduction to cellular automata and other models that are relevant to the newly developing study of complexity, and there are many models and applications that we have not discussed. These models range from attempts to understand the most fundamental processes of nature such as biological evolution and fluid flow to practical studies of the setting of cement (cf. Bentz et al.). In addition, one of the major motivations for the study of cellular automata is their relation to theories of computation and the development of new computer architectures (cf. Hillis). Some researchers believe that ultimately all models of nature can be reduced to cellular automata. One of the attractive features of these models is that the complexity of nature can be ultimately understood as the result of simple and local rules of evolution.

Project 15.19. Cellular automata spring-block model of earthquakes

Mechanical models of earthquakes based on Newton’s equations of motion are difficult to simulate because of the wide range of time scales inherent in these models (cf. Carlson and Langer). For this reason various cellular automata models have been proposed (cf. Klein et al.) that approximate the dynamics on short time scales. The idea is that such approximations are not important because the time interval between earthquakes is much larger than the time of an individual slip. Consider a set of L blocks that are constrained to move in one dimension. Each block is connected to its two nearest neighbors by harmonic springs with stiffness constant k_c and to a loader plate by a leaf spring with stiffness constant k_L . The loader plate moves at velocity v . The stress σ_j on block j at time t after the external loader plate has advanced a distance nv is given by

$$\sigma_j = k_L(nv - x_j(t)) + k_c(x_{j+1}(t) + x_{j-1}(t) - 2x_j(t)), \quad (15.12)$$

where $x_j(t)$ is the displacement of block j at time t from its initial equilibrium position. The time t is associated with local rupture and stress release, which typically is the order of a few seconds. The integer n is the number of loader plate updates and is the time associated with the transmission of the tectonic loading force, typically the order of years or decades. We assume that the equilibrium distance between blocks is unity.

A block slips only if its stress exceeds the static friction threshold in which case it advances by an amount proportional to the stress. The displacement of block j at time $t+1$ is given by the equation of motion:

$$x_j(t+1) = x_j(t) + \frac{\sigma_j(t)}{k_j} \theta(\sigma_j(t) - S), \quad (15.13)$$

where

$$k_j = k_L + q_j k_c, \quad (15.14)$$

and S is the static friction threshold. The value of the coordination number q in (15.14) is two except if free boundary conditions are used in which case $q = 1$ for the end blocks. The step function $\theta(x)$ is unity for $x \geq 0$ and zero for $x < 0$. Note that the equation of motion sets the residual stress on block j equal to zero. The evolution of this model is given by the following steps.

1. Choose the initial displacements of the blocks at random, that is, let $x_j(t=0) = (r_j - 0.5)$, where r_j is a uniform random number in the unit interval. Set $n = 0$.
2. Compute the stress σ_j on each block according to (15.12).
3. Blocks j for which $\sigma_j \geq S$ slip according to the rule, $x_j \rightarrow x_j + \sigma_j/k_j$ (see (15.13)). Increment the “slip” time by unity, $t \rightarrow t + 1$.
4. Repeat steps 2 and 3 until $\sigma_j < S$ for all blocks.
5. Increase the stress on each block by the amount $k_L v \Delta n$ and increase n by $\Delta n = 1$.
6. Repeat steps 2–5 until a sufficient number of loader plate updates have been obtained.

Note that the evolution is deterministic and that randomness is introduced only in the initial conditions. The nonlinearity in the model is in the rule for slips. In the following, we set $k_L = k_c = 1$.

- a. Explain how the dynamics of the model can lead to avalanches of failed blocks. Each avalanche of failed blocks represents an earthquake. Write a program to implement the model. For your preliminary simulations, choose a system of $L = 64$ blocks. Equilibrate the system for a time long enough for each block to fail (slip) at least once. Choose $S = 1$ and $v = 0.02$. After equilibrium has been established, compute s , the number of blocks that slip during one loader update. (If a block slips more than once, count it each time it slips.) Compute the histogram $H(s)$ averaged over many loader plate updates. How would you characterize the dependence of $H(s)$ on s ? Do you see any evidence of power law behavior? Does the behavior of $H(s)$ depend on the size of the system? Choose either periodic or free boundary conditions and determine if the boundary conditions influence your results.
- b. Another quantity of interest is the average slip deficit ϕ_n defined as

$$\phi_n = \frac{1}{L} \sum_j (x_j - nv). \quad (15.15)$$

The average slip deficit is a measure of the lag of the blocks with respect to the movement of the loader plate. Describe the nature of the dependence of ϕ_n on n . Is the n -dependence of ϕ_n purely random? Compute the power spectrum of ϕ_n and characterize its dependence on ω . Try different values of the parameters v , S , and k_c . The properties of this and similar earthquake models are an area of much current interest.

References and Suggestions for Further Reading

- Preben Alstrøm and João Leão, "Self-organized criticality in the game of life," *Phys. Rev. E* **49**, R2507 (1994). The authors find evidence for self-organized criticality using finite size scaling, but suggest that further studies are needed.
- P. Bak, "Catastrophes and self-organized criticality," *Computers in Physics* **5**(4), 430 (1991). A good introduction to self-organized critical phenomena.
- Per Bak, Kan Chen, and Michael Creutz, "Self-organized criticality in the Game of Life," *Nature* **342**, 780 (1989). These workers consider lattices up to 150×150 and find evidence that the local configurations self-organize into a critical state.
- Per Bak and Kim Sneppen, "Punctuated equilibrium and criticality in a simple model of evolution," *Phys. Rev. Lett.* **71**, 4083 (1993); Henrik Flyvbjerg, Kim Sneppen, and Per Bak, "Mean field theory for a simple model of evolution," *Phys. Rev. Lett.* **71**, 4087 (1993);
- P. Bak, C. Tang, and K. Wiesenfeld, "Self-organized criticality," *Phys. Rev. A* **38**, 364 (1988).
- Per Bak and Michael Creutz, "Fractals and self-organized criticality," in *Fractals in Science*, Armin Bunde and Shlomo Havlin, editors, Springer-Verlag (1994).
- Charles Bennett and Marc S. Bourzutschky, "Life not critical?," *Nature* **350**, 468 (1991). The authors present evidence that the criticality of the Game of Life is an artifact resulting from the simulation of small lattices.
- Dale P. Bentz, Peter V. Coveny, Edward J. Garboczi, Michael F. Kleyn, and Paul E. Stutzman, "Cellular automaton simulations of cement hydration and microstructural development," *Modelling Simul. Mater. Sci. Eng.* **2**, 783 (1994).
- E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for your Mathematical Plays, Vol. 2*, Academic Press (1982). A discussion is given of how the Game of Life simulates a universal computer.
- J. M. Carlson and J. S. Langer, "Mechanical model of an earthquake fault," *Phys. Rev. A* **40**, 6470 (1989).
- John W. Clark, Johann Rafelski, and Jeffrey V. Winston, "Brain without mind: Computer simulation of neural networks with modifiable neuronal interactions," *Phys. Repts.* **123**, 215 (1985).
- Michael Creutz, "Deterministic Ising dynamics," *Ann. Phys.* **167**, 62 (1986). A deterministic cellular automaton rule for the Ising model is introduced.
- Gary D. Doolen, Uriel Frisch, Brosl Hasslacher, Steven Orszag, and Stephen Wolfram, editors, *Lattice Gas Methods for Partial Differential Equations*, Addison-Wesley (1990). A collection of reprints and original articles by many of the leading workers in lattice gas methods.

- Stephanie Forrest, editor, *Emergent Computation: Self-Organizing, Collective, and Cooperative Phenomena in Natural and artificial Computing Networks*, MIT Press (1991).
- Stephen I. Gallant, *Neural Network Learning and Expert Systems*, MIT Press (1993).
- M. Gardner, *Wheels, Life and other Mathematical Amusements*, W. H. Freeman (1983).
- G. Grinstein and C. Jayaprakash, “Simple models of self-organized criticality,” *Computers in Physics* **9**, 164 (1995).
- G. Grinstein, Terence Hwa, and Henrik Jeldtoft Jensen, “ $1/f^\alpha$ noise in dissipative transport,” *Phys. Rev. A* **45**, R559 (1992).
- B. Hayes, “Computer recreations,” *Sci. Amer.* **250**(3), 12 (1984). An introduction to cellular automata.
- Jan Hemmingsson, “Consistent results on ‘Life’,” *Physica D* **80**, 151 (1995). The author measures the same properties as Bak, Chen, and Creutz and finds that the power law behavior seen for smaller lattices disappears for larger lattices (1024×1024 with open boundary conditions).
- John Hertz, Anders Krogh, and Richard G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley (1991).
- J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proc. Natl. Acad. Sci. USA* **79**, 2554 (1982).
- W. Daniel Hillis, *The Connection Machine*, MIT Press (1985). A discussion of a new massively parallel computer architecture influenced in part by physical models of the type discussed in this chapter.
- H. M. Jaeger, Chu-heng Liu, and Sidney R. Nagel, “Relaxation at the angle of repose,” *Phys. Rev. Lett.* **62**, 40 (1989). The authors discuss their experiments on real sand piles.
- Stuart A. Kauffman, *The Origins of Order: Self-Organization and Selection in Evolution*, Oxford University Press (1993); “Cambrian explosion and Permian quiescence: Implications of rugged fitness landscapes,” *Evol. Ecol.* **3**, 274 (1989).
- W. Klein, C. Ferguson, and J. B. Rundle, “Spinodals and scaling in slider block models,” in *Reduction and Predictability of Natural Disasters*, J. B. Rundle, D. L. Turcotte, and W. Klein, editors, Addison-Wesley (1995).
- J. A. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press (1992).
- Chris Langton, “Studying artificial life with cellular automata,” *Physica D* **22**, 120 (1986). See also Christopher G. Langton, editor, *Artificial Life*, Addison-Wesley (1989); Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, Addison-Wesley (1989); Christopher G. Langton, editor, *Artificial Life III*, Addison-Wesley (1994);

Roger Lewin, *Complexity: life at the edge of chaos*, MacMillan (1992). A popular exposition on complexity theory.

Elaine S. Oran and Jay P. Boris, *Numerical Simulation of Reactive Flow*, Elsevier Science Publishing (1987). Although much of this book assumes an understanding of fluid dynamics, there is much discussion of simulation methods and of the numerical solution of the differential equations of fluid flow.

Sergei Maslov, Maya Paczuski, and Per Bak, “Avalanches and 1/f noise in evolution and growth models,” *Phys. Rev. Lett.* **73**, 2162 (1994).

William Poundstone, *The Recursive Universe*, Contemporary Books (1985). A book on the Game of Life that attempts to draw analogies between the patterns of Life and ideas of information theory and cosmology.

Daniel H. Rothman and Stéphane Zaleski, “Lattice-gas models of phase separation: interfaces, phase transitions, and multiphase flow,” *Rev. Mod. Phys.* **66**, 1417 (1994). A comprehensive review paper.

David E. Rumelhart and James L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: *Foundations*, MIT Press (1986). See also Vol. 2 on applications.

L. Schulman and P. Seiden, “Statistical mechanics of a dynamical system based on Conway’s Game of Life,” *J. Stat. Phys.* **19**, 293 (1978).

Dietrich Stauffer, “Cellular Automata,” Chapter 9 in *Fractals and Disordered Systems*, Armin Bunde and Shlomo Havlin, editors, Springer-Verlag (1991). Also see “Programming cellular automata,” *Computers in Physics* **5**(1), 62 (1991).

Daniel L. Stein, editor, *Lectures in the Sciences of Complexity*, Vol. 1, Addison-Wesley (1989); Erica Jen, editor, *Lectures in Complex Systems*, Vol. 2, Addison-Wesley (1990); Daniel L. Stein and Lynn Nadel, editors, *Lectures in Complex Systems*, Vol. 3, Addison-Wesley (1991).

Patrick Sutton and Sheri Boyden, “Genetic algorithms: A general search procedure,” *Amer. J. Phys.* **62**, 549 (1994). This readable paper discusses the application of genetic algorithms to Ising models and function optimization.

Tommaso Toffoli and Norman Margolus, *Cellular Automata Machines – A New Environment for Modeling*, MIT Press (1987). See also Norman Margolus and Tommaso Toffoli, “Cellular Automata Machines,” in the volume edited by Doolen et al. (see above).

D. J. Tritton, *Physical Fluid Dynamics*, second edition, Oxford Science Publications (1988). An excellent introductory text that integrates theory and experiment. Although there is only a brief discussion of numerical work, the text provides the background useful for simulating fluids.

Gérard Y. Vichniac, “Cellular automata models of disorder and organization,” in *Disordered Systems and Biological Organization*, E. Bienenstock, F. Fogelman Soulie, and G. Weisbuch, eds. Springer-Verlag (1986). See also Gérard Y. Vichniac, “Taking the computer seriously

in teaching science (an introduction to cellular automata)," in *Microscience*, Proceedings of the UNESCO Workshop on Microcomputers in Science Education, G. Marx and P. Szucs, editors, Balaton, Hungary (1985).

M. Mitchell Waldrop, *Complexity: the emerging science at the edge of order and chaos*, Simon and Schuster (1992). A popular exposition of complexity theory.

Stephen Wolfram, editor, *Theory and Applications of Cellular Automata*, World Scientific (1986).

A collection of research papers on cellular automata that range in difficulty from straightforward to specialists only. An extensive annotated bibliography also is given. Two papers in this collection that discuss the classification of one-dimensional cellular automata are S. Wolfram, "Statistical mechanics of cellular automata," *Rev. Mod. Phys.* **55**, 601 (1983), and S. Wolfram, "Universality and complexity in cellular automata," *Physica* **B10**, 1 (1984).

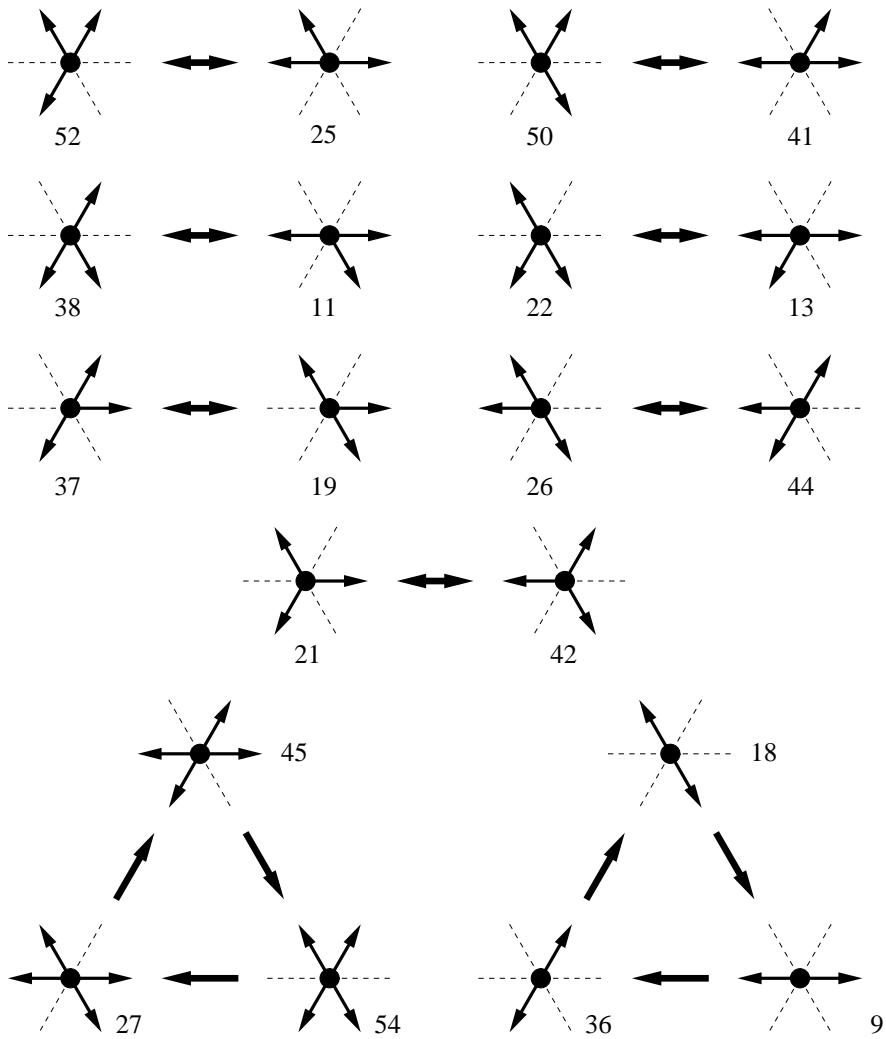


Figure 15.4: Examples of collision rules for a lattice gas on a triangular lattice. The rule for configurations that are not shown is that the velocities do not change after a collision. The numbers represent the way that the velocities at a lattice site are encoded.

Chapter 16

The Microcanonical Ensemble

©2002 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
22 April 2002

We simulate the microcanonical ensemble and “discover” the Boltzmann distribution for systems in thermal contact with a heat bath.

16.1 Introduction

The Monte Carlo simulations of the “particles in the box” problem discussed in Chapter 7 and the molecular dynamics simulations discussed in Chapter 8 have exhibited some of the important qualitative features of macroscopic systems, for example, the irreversible approach to equilibrium and the existence of equilibrium fluctuations in macroscopic quantities. In this chapter we apply Monte Carlo methods to the simulation of the equilibrium properties of systems with many degrees of freedom. This application will allow us to explore the methodology of statistical mechanics and to introduce the concept of temperature.

Due in part to the impact of computer simulations, the applications of statistical mechanics have expanded from the traditional areas of dense gases and liquids to the study of phase transitions, particle physics, and theories of the early universe. In fact, the algorithm introduced in this chapter was developed by a physicist interested in using computer simulations to predict experimentally verifiable quantities from lattice gauge theories, theories used to describe the interactions of fundamental particles.

16.2 The Microcanonical Ensemble

We first discuss an *isolated* system for which the number of particles N , the volume V , and the total energy E are fixed and the influence of external parameters such as gravitational and magnetic fields can be ignored. In general, an isolated macroscopic system tends to a time-independent equilibrium state of maximum randomness or entropy. The *macrostate* of the system is specified

by the values of E , V , and N . At the microscopic level there are a large number of different ways or *configurations* in which the macrostate (E, V, N) can be realized. A particular configuration or *microstate* is *accessible* if its properties are consistent with the specified macrostate.

All we know about the accessible microstates is that their properties are consistent with the known physical quantities of the system. Because we have no reason to prefer one microstate over another, it is reasonable to postulate that the system is *equally* likely to be in any one of its accessible microstates. To make this postulate of *equal a priori probabilities* more precise, imagine an isolated system with Ω accessible states. The probability P_s of finding the system in microstate s is

$$P_s = \begin{cases} 1/\Omega, & \text{if } s \text{ is accessible} \\ 0, & \text{otherwise.} \end{cases} \quad (16.1)$$

The sum of P_s over all Ω states is equal to unity.

The averages of physical quantities can be determined in two ways. In the usual laboratory experiment, physical quantities are measured over a time interval sufficiently long to allow the system to sample a large number of its accessible microstates. We already performed such time averages in Chapter 6, where we used the method of molecular dynamics to compute the time-averaged values of quantities such as the temperature and pressure. An interpretation of the probabilities in (16.1) that is consistent with such a time average is that during a sequence of observations, P_s yields the fraction of times that a single system is found in a given microscopic state.

Although time averages are conceptually simple, it is convenient to formulate statistical averages at a given instant of time. Instead of performing measurements on a single system, imagine a collection or *ensemble* of systems that are identical mental replicas characterized by the same macrostate. The number of systems in the ensemble equals the number of possible microstates. In this interpretation, the probabilities in (16.1) describe an ensemble of identical systems. An ensemble of systems specified by E , N , V is called a *microcanonical ensemble*. Suppose that a physical quantity A has the value A_s when the system is in the state s . Then the ensemble average of A is given by

$$\langle A \rangle = \sum_{s=1}^{\Omega} A_s P_s, \quad (16.2)$$

where P_s is given by (16.1).

To illustrate these ideas, consider a one-dimensional model of an ideal gas in which the particles are distinguishable, noninteracting, and have only two possible velocities v_0 and $-v_0$. Because the particles are noninteracting, the size of the system and the positions of the particles are irrelevant. In Table 16.1 we show the ensemble of systems consistent with $N = 4$ and $E = 2v_0^2$. The mass of the particles is assumed to be unity.

The enumeration of the sixteen systems in the ensemble allows us to calculate ensemble averages for the physical quantities of the system. For example, inspection of Table 16.1 shows that P_n , the probability that the number of particles moving to the right is n , is given by 1/16, 4/16, 6/16, 4/16, and 1/16 for $n = 0, 1, 2, 3$, and 4, respectively. Hence, the mean number of particles

L L L L	L L L R	L L R R	L R R R	R R R R
L L R L	L R L R	R L R R		
L R L L	L R R L	R R L R		
R L L L	R L L R	R R R L		
	R L R L			
	R R L L			

Table 16.1: The sixteen possible microstates for a one-dimensional system of $N = 4$ noninteracting particles. The letter R denotes a particle moving to the right and the letter L denotes a particle moving to the left. Each particle has speed v_0 . The mass of the particles is taken to be unity and the total (kinetic) energy $E = 4(v_0^2/2)$.

moving to the right is

$$\langle n \rangle = \sum n P_n = (0 \times 1 + 1 \times 4 + 2 \times 6 + 3 \times 4 + 4 \times 1)/16 = 2. \quad (16.3)$$

16.3 The Demon Algorithm

We found in Chapter 8 that we can do a time average of a system of many particles with E , V , and N fixed by integrating Newton's equations of motion for each particle and computing the time-averaged value of the physical quantities of interest. How can we do an ensemble average at fixed E , V , and N ? One way would be to enumerate all the microstates and calculate the ensemble average of the desired physical quantities as we did in the ideal gas example. However, this approach usually is not practical, because the number of microstates for even a small system is much too large to enumerate. In the spirit of Monte Carlo, we wish to develop a practical method of obtaining a representative sample of the total number of microstates. An obvious procedure is to fix V and N , change the positions and velocities of the individual particles at random, and retain the configuration if it has the desired total energy. However this procedure is very inefficient, because most configurations would not have the desired total energy and would have to be discarded.

An efficient Monte Carlo procedure has been developed by Creutz and coworkers. Suppose that we add an extra degree of freedom to the original macroscopic system of interest. For historical reasons, this extra degree of freedom is called a *demon*. The demon travels about the system transferring energy as it attempts to change the dynamical variables of the system. If a desired change lowers the energy of the system, the excess energy is given to the demon. If the desired change raises the energy of the system, the demon gives the required energy to the system if the demon has sufficient energy. The only constraint is that the demon cannot have negative energy. The demon algorithm for a classical system of particles is summarized in the following:

1. Choose a particle at random and make a trial change in its position.
2. Compute ΔE , the change in the energy of the system due to the change.
3. If $\Delta E \leq 0$, the system gives the amount $|\Delta E|$ to the demon, that is, $E_d = E_d - \Delta E$, and the trial configuration is accepted.

4. If $\Delta E > 0$ and the demon has sufficient energy for this change ($E_d \geq \Delta E$), then the demon gives the necessary energy to the system, that is, $E_d = E_d - \Delta E$, and the trial configuration is accepted. Otherwise, the trial configuration is rejected and the configuration is not changed.

The above steps are repeated until a representative sample of states is obtained. After a sufficient number of steps, the demon and the system will reach a compromise and agree on an average energy for each. The total energy remains constant, and because the demon is only one degree of freedom in comparison to the many degrees of freedom of the system, the energy fluctuations of the system will be order $1/N$.

How do we know that this Monte Carlo simulation of the microcanonical ensemble will yield results equivalent to the time-averaged results of molecular dynamics? The assumption that the two averages yield equivalent results is called the *ergodic hypothesis* (more accurately, the *quasi-ergodic hypothesis*). Although these two averages have not been shown to be identical in general, they have been found to yield equivalent results in all cases of interest.

16.4 One-Dimensional Classical Ideal Gas

We first apply the demon algorithm to the one-dimensional classical ideal gas. Of course, we do not need to use the demon algorithm for an ideal gas because a reduction in the energy of one particle can be easily compensated by the corresponding increase in the energy of another particle. However, it is a good idea to consider a simple example first.

For an ideal gas, the energy of a configuration is independent of the positions of the particles, and the total energy is the sum of the kinetic energies of the individual particles. Hence, for an ideal gas the only coordinates of interest are the velocity coordinates. To change a configuration, we choose a particle at random and change its velocity by a random amount.

Program ideal implements a microcanonical Monte Carlo simulation of an ideal classical gas in one dimension. The variable `mcs`, the number of Monte Carlo steps per particle, plays an important role in Monte Carlo simulations. On the average, the demon attempts to change the velocity of each particle once during each Monte Carlo step per particle. We frequently will refer to the number of Monte Carlo steps per particle as the “time,” even though this time has no obvious direct relation to a physical time.

```

PROGRAM ideal
! demon algorithm for the one-dimensional, ideal classical gas
DIM v(100)
CALL initial(N,v(),mcs,E,Ed,Ecum,Edcum,accept,dvmax)
FOR imcs = 1 to mcs
    CALL change(N,v(),E,Ed,Ecum,Edcum,accept,dvmax)
NEXT imcs
CALL averages(N,Ecum,Edcum,mcs,accept)
END

SUB initial(N,v(),mcs,E,Ed,Ecum,Edcum,accept,dvmax)
    RANDOMIZE

```

```

INPUT prompt "number of particles = ": N
INPUT prompt "initial energy of system = ": E
LET Ed = 0           ! initial demon energy
INPUT prompt "number of MC steps per particle = ": mcs
INPUT prompt "maximum change in velocity = ": dvmax
! divide energy equally among particles
LET vinitial = sqr(2*E/N)    ! mass unity
! all particles have same initial velocities
FOR i = 1 to N
  LET v(i) = vinitial
NEXT i
! initialize sums
LET Ecum = 0
LET Edcum = 0
LET accept = 0
END SUB

SUB change(N,v(),E,Ed,Ecum,Edcum,accept,dvmax)
FOR i = 1 to N
  LET dv = (2*rnd - 1)*dvmax      ! trial change in velocity
  LET ipart = int(rnd*N + 1)      ! select random particle
  LET vtrial = v(ipart) + dv      ! trial velocity
  ! trial energy change
  LET de = 0.5*(vtrial*vtrial - v(ipart)*v(ipart))
  IF de <= Ed then
    LET v(ipart) = vtrial
    LET accept = accept + 1
    LET Ed = Ed - de
    LET E = E + de
  END IF
NEXT i
! accumulate data after each Monte Carlo step per particle
LET Ecum = Ecum + E
LET Edcum = Edcum + Ed
END SUB

SUB averages(N,Ecum,Edcum,mcs,accept)
LET norm = 1/mcs
LET Edave = Edcum*norm      ! mean demon energy
LET norm = norm/N
LET accept_prob = accept*norm      ! acceptance probability
! system averages per particle
LET Esave = Ecum*norm      ! mean energy per system particle
PRINT "mean demon energy ="; Edave
PRINT "mean system energy per particle ="; Esave
PRINT "acceptance probability ="; accept_prob

```

```
END SUB
```

Problem 16.1. Monte Carlo simulation of an ideal gas

- a. We will use **Program ideal** to investigate some of the equilibrium properties of an ideal gas. Suppose that we assign the same initial velocity to all the particles. What is the mean value of the particle velocities after equilibrium has been reached? Choose the number of particles $N = 40$, the initial total energy $E = 10$, the initial demon energy $E_d = 0$, the maximum change in the velocity $\text{dvmax} = 2$, and the number of Monte Carlo steps per particle $\text{mcs} \geq 1000$. The mass of the particles is set equal to unity.
- b. The configuration corresponding to all particles having the same velocity is not very likely, and it would be better to choose an initial configuration that is more likely to occur when the system is in equilibrium. Because this choice is not always possible, we should let the system evolve for a number of Monte Carlo steps per particle before we accumulate data for the averages. We call this number the *equilibration time*. Modify **Program ideal** so that the changes are made for **nequil** Monte Carlo steps per particle before averages are taken. We can estimate this time from a plot of the time average of the demon energy or other quantity of interest versus the time. Determine the mean demon energy and mean system energy per particle for the parameters in part (a).
- c. Compute the mean energy of the demon and the mean system energy per particle for $E = 20$ and $E = 40$. Choose $\text{mcs} = 50000$ if possible. Use your result from part (b) and obtain an approximate relation between the mean demon energy and the mean system energy per particle.
- d. In the microcanonical ensemble the total energy is fixed with no reference to temperature. Define the temperature by the relation $\frac{1}{2}m\langle v^2 \rangle = \frac{1}{2}kT_{\text{kin}}$, where $\frac{1}{2}m\langle v^2 \rangle$ is the mean kinetic energy per particle. Use this relation to obtain T_{kin} . How is T_{kin} related to the mean demon energy? Choose energy units such that Boltzmann's constant k is equal to unity.
- e. A limitation of any Monte Carlo simulation is the finite number of particles. In part (d) we found that the mean demon energy is approximately twice the mean kinetic energy per particle. In the infinite particle limit this relation would hold exactly. Determine how close your results come to the infinite particle results for $N = 2$ and $N = 10$. If there is no statistically significant difference between your results for the two values of N , explain why finite N might not be an important limitation for the ideal gas.

16.5 The Temperature and the Canonical Ensemble

Although the microcanonical ensemble is conceptually simple, it does not represent the situation usually found in the laboratory. Most laboratory systems are not isolated, but are in thermal contact with their environment. This thermal contact allows energy to be exchanged between the laboratory system and its environment in the form of heat. The laboratory system is usually small relative to its environment. The larger system with many more degrees of freedom is referred to as the *heat reservoir* or *heat bath*.

We now consider the more realistic case for which the total energy of the *composite* system consisting of the laboratory system and the heat bath is constrained to be constant, but the energy

of the laboratory system can vary. Imagine a large number of mental copies of the laboratory system and the heat bath. Considered together, the composite system is isolated and can be described by the microcanonical ensemble. However, because we are interested in the equilibrium values of the laboratory system, we need to know the probability P_s of finding the laboratory system in the microstate s with energy E_s . The ensemble that describes the probability distribution of the laboratory system in thermal equilibrium with a heat bath is known as the *canonical ensemble*.

In general, the laboratory system can be any macroscopic system that is much smaller than the heat bath. The laboratory system can be as small as an individual particle if the latter can be clearly distinguished from the particles of the heat bath. An example of such a laboratory system is the demon itself. Hence, we can consider the demon to be a system whose microstate is specified only by its energy. The demon is a model of a laboratory system in equilibrium with a heat bath.

One way of finding the form of the probability distribution of the canonical ensemble is to simulate a demon exchanging energy with an ideal gas. The ideal gas serves as the heat bath. The main quantity of interest is the probability $P(E_d)$ that the demon has energy E_d . We will find in Problem 16.2 that the form of $P(E_d)$ is given by

$$P(E_d) = \frac{1}{Z} e^{-E_d/kT}, \quad (16.4)$$

where Z is a normalization constant such that the sum over all the states of the demon is unity. The parameter T in (16.4) is called the *absolute temperature* and is measured in Kelvin (K). Boltzmann's constant k is given by $k = 1.38 \times 10^{-23}$ J/K. The probability distribution (16.4) is called the *Boltzmann* or the *canonical distribution*, and Z is called the *partition function*.

The form (16.4) of the Boltzmann distribution provides a simple way of computing T from the mean demon energy $\langle E_d \rangle$ given by

$$\langle E_d \rangle = \frac{\int_0^\infty E e^{-E/kT} dE}{\int_0^\infty e^{-E/kT} dE} = kT. \quad (16.5)$$

We see that T is proportional to the mean demon energy. Note that the result $\langle E_d \rangle = kT$ in (16.5) holds only if the energy of the demon can take on a continuum of values and if the upper limit of integration can be taken to be ∞ .

Problem 16.2. The Boltzmann probability distribution

- Add a subroutine to **Program ideal** to compute the probability distribution $P(E_d)$ of the demon. Because E_d is a continuous variable, it is necessary to place the values of E_d in appropriate bins. Plot the natural logarithm of $P(E_d)$ versus E_d and verify the form (16.4) for the Boltzmann distribution. What is the slope of this plot? Choose units such that $k = 1$ and estimate the corresponding value of T . Choose the same parameters as were used in Problem 16.1. Be sure to determine $P(E_d)$ only after thermal equilibrium has been obtained.
- Determine the magnitude of T from the relation (16.5). Are your two estimates of T consistent?
- Compare the value of T obtained in parts (a) and (b) with the value of T found in Problem 16.1 using the kinetic definition of the temperature. Is the demon in thermal equilibrium with its heat bath?



Figure 16.1: The interaction energy between nearest neighbor spins in the absence of an external magnetic field.

16.6 The Ising Model

A popular model of a system of interacting variables in statistical physics is the *Ising* model. The model was proposed by Lenz and investigated by his graduate student, Ising, to study the phase transition from a paramagnet to a ferromagnet (cf. Brush). Ising computed the thermodynamic properties of the model in one dimension and found that the model does not have a phase transition. However, for two and three dimensions the Ising model does exhibit a transition. The nature of the phase transition in two dimensions and the diverse applications of the Ising model are discussed in Chapter 17.

To introduce the Ising model, consider a lattice containing N sites and assume that each lattice site i has associated with it a number s_i , where $s_i = +1$ for an “up” (\uparrow) spin and $s_i = -1$ for a “down” (\downarrow) spin. A particular configuration or microstate of the lattice is specified by the set of variables $\{s_1, s_2, \dots, s_N\}$ for all lattice sites.

The macroscopic properties of a system are determined by the nature of the accessible microstates. Hence, it is necessary to know the dependence of the energy on the configuration of spins. The total energy E of the Ising model is given by

$$E = -J \sum_{i,j=\text{nn}(i)}^N s_i s_j - H \sum_{i=1}^N s_i, \quad (16.6)$$

where H is proportional to a uniform external magnetic field. The first sum in (16.6) is over all nearest neighbor pairs. The *exchange constant* J is a measure of the strength of the interaction between nearest neighbor spins (see Fig. 16.1). The second sum in (16.6) represents the energy of interaction of the magnetic moments associated with the spins with an external magnetic field.

If $J > 0$, then the states $\uparrow\uparrow$ and $\downarrow\downarrow$ are energetically favored in comparison to the states $\uparrow\downarrow$ and $\downarrow\uparrow$. Hence for $J > 0$, we expect that the state of lowest total energy is *ferromagnetic*, that is, the spins all point in the same direction. If $J < 0$, the states $\uparrow\downarrow$ and $\downarrow\uparrow$ are favored and the state of lowest energy is expected to be *antiferromagnetic*, that is, alternate spins are aligned. If we subject the spins to an external magnetic field directed upward, the spins \uparrow and \downarrow possess an additional internal energy given by $-H$ and $+H$ respectively.

An important virtue of the Ising model is its simplicity. Some of its simplifying features are that the kinetic energy of the atoms associated with the lattice sites has been neglected, only nearest neighbor contributions to the interaction energy have been included, and the spins are allowed to have only two discrete values. In spite of the simplicity of the model, we will find that it exhibits very interesting behavior.

For the familiar case of classical particles with continuously varying position and velocity coordinates, the dynamics is given by Newton's laws. For the Ising model the dependence (16.6) of the energy on the spin configuration is not sufficient to determine the time-dependent properties of the system. That is, the relation (16.6) does not tell us how the system changes from one spin configuration to another and we have to introduce the dynamics separately.

In Problem ?? we simulated the Ising model using a cellular automata approach. The major limitation of this approach is that it is difficult for the system to sample a representative set of configurations. In addition, there is no simple measure of the temperature. The demon algorithm is much more effective at exploring the set of possible configurations, because the energy of the lattice can fluctuate slightly allowing the lattice to sample any configuration with nearly the same energy. We implement the demon algorithm by choosing a spin at random. The trial change corresponds to a flip of the spin from \uparrow to \downarrow or \downarrow to \uparrow .

Because we are interested in the properties of an infinite system, we have to choose appropriate boundary conditions. The simplest boundary condition in one dimension is to choose a “free surface” so that the spins at sites 1 and N each have one nearest neighbor interaction only. In general, a better choice is periodic boundary conditions. For this choice the lattice becomes a ring and the spins at sites 1 and N interact with one another and hence have the same number of interactions as do the other spins.

What are some of the physical quantities whose averages we wish to compute? An obvious physical quantity is the *magnetization* M given by

$$M = \sum_{i=1}^N s_i, \quad (16.7)$$

and the magnetization per spin $m = M/N$. Usually we are interested in the average values $\langle M \rangle$ and the fluctuations $\langle M^2 \rangle - \langle M \rangle^2$. We can determine the temperature T as a function of the energy of the system in two ways. One way is to measure the probability that the demon has energy E_d . Because we know that this probability is proportional to $\exp(-E_d/kT)$, we can determine T from a plot of the logarithm of the probability as a function of E_d . An easier way to determine T is to measure the mean demon energy. However, because the values of E_d are not continuous for the Ising model, T is not proportional to $\langle E_d \rangle$ as it is for the ideal gas. We show in Appendix 16.8 that for $H = 0$ and the limit of an infinite system, the temperature is related to $\langle E_d \rangle$ by

$$kT/J = \frac{4}{\ln(1 + 4J/\langle E_d \rangle)}. \quad (16.8)$$

The result (16.8) comes from replacing the integrals in (16.5) by sums over the possible demon energies. Note that in the limit $|J/E_d| \ll 1$, (16.8) reduces to $kT = E_d$ as expected.

Program `demon` implements the microcanonical simulation of the Ising model in one dimension using spin flip dynamics and periodic boundary conditions. Once the initial configuration is chosen, the demon algorithm is similar to that described in Section 16.3. However, in contrast to the ideal gas, the spins in the one-dimensional Ising model must be chosen randomly.

```
PROGRAM demon
! demon algorithm for the d = 1 Ising model in zero magnetic field
```

```

DIM spin(1000)
LIBRARY "mygraphics"
CALL initial(N,spin(),E,Ed,M,mcs,Ecum,Edcum,Mcum,M2cum,accept)
CALL setupscreen(N,spin(),up$,down$)
FOR imcs = 1 to mcs
    CALL change(N,spin(),E,Ed,M,accept,up$,down$)
    CALL data(E,Ed,M,Ecum,Edcum,Mcum,M2cum)
NEXT imcs
CALL averages(N,Ecum,Edcum,Mcum,M2cum,mcs,accept)
END

SUB initial(N,spin(),E,Ed,M,mcs,Ecum,Edcum,Mcum,M2cum,accept)
    RANDOMIZE
    INPUT prompt "number of spins = ": N
    ! choose total energy to be multiple of 4J
    ! coupling constant J is unity
    INPUT prompt "desired total energy = ": Etot
    LET Etot = 4*int(Etot/4)
    INPUT prompt "number of Monte Carlo steps per spin = ": mcs
    ! initial configuration of spins in minimum energy state
    FOR isite = 1 to N
        LET spin(isite) = 1
    NEXT isite
    LET M = N                      ! net magnetization
    ! compute initial system energy
    LET E = -N                      ! periodic boundary conditions
    LET Ed = (Etot - E)
    PRINT "total energy = "; E + Ed
    ! initialize sums
    LET Ecum = 0
    LET Edcum = 0
    LET Mcum = 0
    LET M2cum = 0
END SUB

SUB setupscreen(N,spin(),up$,down$)
    LET r = N/(2*pi)
    CALL compute_aspect_ratio(r+2,xwin,ywin)
    SET WINDOW -xwin,xwin,-ywin,ywin
    LET dtheta = 2*pi/N
    SET COLOR "red"
    BOX AREA 1,1+0.5,1,1+0.5
    BOX KEEP 1,1+0.5,1,1+0.5 in up$
    CLEAR
    SET COLOR "blue"
    BOX AREA 1,1+0.5,1,1+0.5

```

```

BOX KEEP 1,1+0.5,1,1+0.5 in down$
CLEAR
FOR i = 1 to N
    CALL showspin(N,spin(i),i,up$,down$)
NEXT i
END SUB

SUB change(N,spin(),E,Ed,M,accept,up$,down$)
FOR i = 1 to N
    LET isite = int(rnd*N + 1)      ! random spin
    ! determine neighboring spin values
    IF isite = 1 then
        LET left = spin(N)
    ELSE
        LET left = spin(isite - 1)
    END IF
    IF isite = N then
        LET right = spin(1)
    ELSE
        LET right = spin(isite + 1)
    END IF
    ! trial energy change
    LET de = 2*spin(isite)*(left + right)
    IF de <= Ed then
        ! spin flip dynamics
        LET spin(isite) = -spin(isite)
        LET M = M + 2*spin(isite)
        LET accept = accept + 1      ! number of changes accepted
        LET Ed = Ed - de
        LET E = E + de
    END IF
    CALL showspin(N,spin(isite),isite,up$,down$)
NEXT i
END SUB

SUB data(E,Ed,M,Ecum,Edcum,Mcum,M2cum)
    ! accumulate data
    LET Ecum = Ecum + E
    LET Edcum = Edcum + Ed
    LET Mcum = Mcum + M
    LET M2cum = M2cum + M*M
END SUB

SUB averages(N,Ecum,Edcum,Mcum,M2cum,mcs,accept)
    SET COLOR "black/white"
    LET norm = 1/mcs                ! collected data after every attempt

```

```

LET Edave = Edcum*norm
PRINT "mean demon energy ="; Edave
LET T = 4/log(1 + 4/Edave)
PRINT "T ="; T
LET Eave = Ecum*norm
PRINT "<E> = "; Eave
LET Mave = Mcum*norm
PRINT "<M> ="; Mave
LET M2ave = M2cum*norm
PRINT "<M*M> ="; M2ave
LET accept_prob = accept*norm/N
PRINT "acceptance probability ="; accept_prob
END SUB

SUB showspin(N,dir,isite,up$,down$)
LET r = N/(2*pi)
LET theta = isite/r
LET x = r*cos(theta)
LET y = r*sin(theta)
IF dir = 1 then
    BOX SHOW up$ at x,y
ELSE
    BOX SHOW down$ at x,y
END IF
END SUB

```

Note that for $H = 0$, the change in energy due to a spin flip is either 0 or $\pm 4J$. Hence the initial energy of the system plus the demon must be an integer multiple of $4J$. Because the spins are interacting, it is difficult to choose an initial configuration of spins with precisely the desired energy. The procedure followed in **SUB initial** is to choose the initial configuration to be all spins up, a minimum energy configuration. The demon energy is chosen so that the total energy of the system and the demon is equal to the desired multiple of $4J$.

Problem 16.3. One-dimensional Ising model

- Use **Program demon** with $N = 100$ and the desired total energy $E_{\text{tot}} = -20$. What is the initial energy assigned to the demon in **SUB initial**? Note that the program shows the spins as a ring. Describe the evolution of the spins. (It might be useful to insert some **PAUSE** statements in the program.) Change E_{tot} and describe any qualitative changes in the evolution.
- Compute the time average of the demon energy and the magnetization M as a function of the time. As usual, we interpret the time as the number of Monte Carlo steps per spin. What is the approximate time for these quantities to approach their equilibrium values?
- Modify the program so that initial nonequilibrium configurations are not used to determine the averages of physical quantities. What are the equilibrium values of $\langle E_d \rangle$, $\langle M \rangle$, and $\langle M^2 \rangle$? The choice of $mcs = 100$ is appropriate for testing the program and yields results of approximately 20% accuracy. To obtain better than 5% results, mcs should be the order of 1000.

- d. Compute T and E for $N = 100$, and the cases $E_{\text{tot}} = -20, -40, -60$, and -80 . Compare your results to the exact result for an infinite one-dimensional lattice, $E/N = -\tanh(J/kT)$. How do your computed results for E/N depend on N and on the number of Monte Carlo steps per spin?
- e. Use the same runs as in part (d) to compute $\langle M^2 \rangle$ as a function of T . Does $\langle M^2 \rangle$ increase or decrease with T ?
- f. Modify `Program demon` and verify the Boltzmann form (16.4) for the energy distribution of the demon.

Problem 16.4. Additional applications

- a. Modify `Program demon` so that the antiferromagnetic case ($J = -1$) is treated. Before doing the simulation describe how you expect the spin configurations to differ from the ferromagnetic case. What is the lowest energy or ground state configuration? Run the simulation with the spins initially in their ground state, and compare your results with your expectations. Compute the mean energy per spin versus temperature and compare your results with the ferromagnetic case.
- b. Modify `Program demon` to include a nonzero magnetic field, $H \neq 0$, and compute $\langle E_d \rangle$, $\langle M \rangle$, and $\langle M^2 \rangle$ as a function of H for fixed E . Read the discussion in Appendix 16.8 and determine the relation of $\langle E_d \rangle$ to T for your choices of H . Is the equilibrium temperature higher or lower than the $H = 0$ case for the same total energy?

**Problem 16.5.* The two-dimensional Ising model

- a. Simulate the two-dimensional Ising model on a square lattice with spin-flip dynamics in the microcanonical ensemble. The total number of spins $N = L^2$, where L is the length of one side of the lattice. Use periodic boundary conditions as shown in Fig. 16.2 so that spins in the left-hand column interact with spins in the right-hand column, etc. Do not include nonequilibrium configurations in your averages.
- b. Compute $\langle E_d \rangle$ and $\langle M^2 \rangle$ as a function of E . Convenient choices of parameters are $L = 10$ and $\text{mcs} = 500$. Assume $J = 1$ and $H = 0$. Use (16.8) to determine the dependence of T on E and plot E versus T .
- c. Repeat the simulations in part (b) for $L = 20$. If necessary, increase mcs until your averages are accurate to within a few percent. Describe how the energy versus temperature curve changes with lattice size.
- d. Modify your program to make “snapshots” of the spin configurations. Describe qualitatively the nature of the configurations at different energies or temperatures. Are they ordered or disordered? Are there domains of up or down spins?

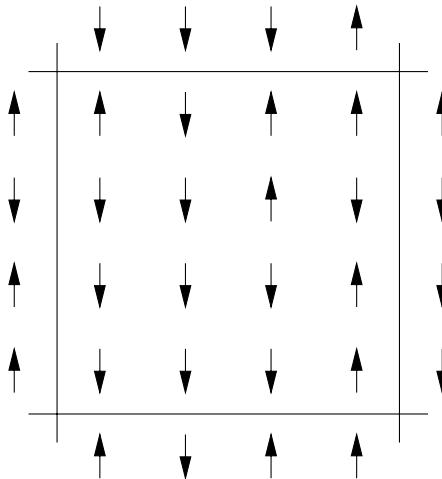


Figure 16.2: One of the 2^N possible configurations of a system of $N = 16$ Ising spins on a square lattice. Also shown are the spins in the four nearest periodic images of the central cell that are used to calculate the energy. An up spin is denoted by \uparrow and a down spin is denoted by \downarrow . Note that the number of nearest neighbors on a square lattice is four. The energy of this configuration is $E = -8J + 4H$ with periodic boundary conditions.

16.7 *Heat Flow

In our applications of the demon algorithm one demon shared its energy equally with all the spins. As a result the spins all attained the same mean energy of interaction. Many interesting questions arise when the system is not spatially uniform and is in a nonequilibrium but time-independent (steady) state.

Let us consider heat flow in a one-dimensional Ising model. Suppose that instead of all the sites sharing energy with one demon, each site has its own demon. We can study the flow of heat by requiring the demons at the boundary spins to satisfy different conditions than the demons at the other spins. The demon at spin 1 adds energy to the system by flipping this spin so that it is in its highest energy state, that is, in the opposite direction of spin 2. The demon at spin N removes energy from the system by flipping spin N so that it is in its lowest energy state, that is, in the same direction as spin $N - 1$. As a result, energy flows from site 1 to site N via the demons associated with the intermediate sites. In order that energy not build up at the “hot” end of the Ising chain, we require that spin 1 can only add energy to the system if spin N simultaneously removes energy from the system. Because the demons at the two ends of the lattice satisfy different conditions than the other demons, we do not use periodic boundary conditions.

The temperature is determined by the generalization of the relation (16.8), that is, the temperature at site i is related to the mean energy of the demon at site i . To control the temperature gradient, we can update the end spins at a rate different than the other spins. The maximum temperature gradient occurs if we update the end spins after every update of an internal spin. A

smaller temperature gradient occurs if we update the end spins less frequently. The temperature gradient between any two spins can be determined from the temperature profile, the spatial dependence of the temperature. The energy flow can be determined by computing the magnitude of the energy per unit time that enters the lattice at site 1.

To implement this procedure we modify **Program demon** by converting the variables **Ed** and **Edcum** to arrays. We do the usual updating procedure for spins 2 through $N - 1$ and visit spins 1 and N at regular intervals denoted by **nvisit**.

```

PROGRAM conduct
! many demon algorithm for Ising chain
! heat added at spin 1 and subtracted at spin N
DIM spin(1000),Ed(1000),Edsum(1000),Msum(1000)
CALL initial(N,spin(),nmcs,nvisit)
FOR imcs = 1 to nmcs
    CALL change(N,spin(),Ed(),accept)
    IF mod(imcs,nvisit) = 0 then CALL heat(N,spin(),Edsum())
    CALL data(N,spin(),Ed(),Edsum(),Msum())
NEXT imcs
CALL output(N,Edsum(),Msum(),nmcs,accept)
END

SUB initial(N,spin(),nmcs,nvisit)
    RANDOMIZE
    INPUT prompt "number of spins = ": N
    INPUT prompt "number of MC steps per spin = ": nmcs
    INPUT prompt "MC steps between updates of end spins = ": nvisit
    ! initial random configuration
    FOR i = 1 to N
        IF rnd > 0.5 then
            LET spin(i) = 1
        ELSE
            LET spin(i) = -1
        END IF
    NEXT i
END SUB

SUB change(N,spin(),Ed(),accept) ! spin flip dynamics
    ! do one Monte Carlo step
    FOR i = 2 to N - 1
        ! pick spin at random from spins 2 to N - 1
        LET isite = int(rnd*(N - 2) + 2)
        ! trial energy change
        LET de = 2*spin(isite)*(spin(isite - 1) + spin(isite + 1))
        IF de <= Ed(isite) then
            LET spin(isite) = - spin(isite)
            LET accept = accept + 1
        END IF
    NEXT i
END SUB

```

```

        LET Ed(isite) = Ed(isite) - de
    END IF
NEXT i
END SUB

SUB heat(N,spin(),Edsum())
    ! attempt to add energy at spin 1 and remove it at spin N
    ! possible only if spins 1 and 2 are aligned
    ! and spins N and N - 1 are not aligned
    IF (spin(1)*spin(2) = 1) and (spin(N)*spin(N-1) = -1) then
        LET Edsum(1) = Edsum(1) + 2
        LET Edsum(N) = Edsum(N) - 2
        LET spin(1) = -spin(1)
        LET spin(N) = -spin(N)
    END IF
END SUB

SUB data(N,spin(),Ed(),Edsum(),Msum())
    FOR i = 2 to N - 1
        LET Edsum(i) = Edsum(i) + Ed(i)
        LET Msum(i) = Msum(i) + spin(i)
    NEXT i
END SUB

SUB output(N,Edsum(),Msum(),nmcs,accept)
    LET norm = 1/nmcs
    LET accept_prob = accept*norm/(N - 2)
    PRINT "acceptance probability = "; accept_prob
    PRINT
    PRINT tab(2); "i"; tab(16); "Ed(i)"; tab(35); "T"; tab(46); "M(i)"
    PRINT
    FOR i = 2 to N-1
        LET edave = Edsum(i)*norm
        LET temperature = 0
        IF Edave <> 0 then
            IF (1 + 4/Edave) > 0 then
                LET temperature = 4/log(1 + 4/Edave)
            END IF
        END IF
        LET M = Msum(i)*norm
        PRINT i,Edave,temperature,M
    NEXT i
END SUB

```

Problem 16.6. One-dimensional heat flow

- As a check on Program conduct, modify the program so that all the demons are equivalent,

that is, impose periodic boundary conditions and do not use `SUB heat`. Compute the mean energy of the demon at each site and use (16.8) to define a local site temperature. Use $N \geq 22$ and $\text{mcs} \geq 1000$. Is the local temperature approximately uniform? How do your results compare with the single demon case?

- b. In `Program conduct` energy is added to the system at site 1 and is removed at site N . Determine the mean demon energy for each site and obtain the corresponding local temperature and the mean energy of the system. Draw the temperature profile by plotting the temperature as a function of site number. The temperature gradient is the difference in temperature from site $N - 1$ to site 2 divided by the distance between them. (The distance between neighboring sites is unity.) Because of local temperature fluctuations and edge effects, the temperature gradient should be estimated by fitting the temperature profile in the middle of the lattice to a straight line. Reasonable choices for the parameters are $N = 22$, $\text{mcs} = 4000$, and `nvisit` = 1.
- c. The heat flux Q is the energy flow per unit length per unit time. The energy flow is the amount of energy that demon 1 adds to the system at site 1. The time is conveniently measured in terms of Monte Carlo steps per spin. Determine Q for the parameters used in part (b).
- d. If the temperature gradient $\partial T / \partial x$ is not too large, the heat flux Q is proportional to $\partial T / \partial x$. We can determine the *thermal conductivity* κ by the relation

$$Q = -\kappa \frac{\partial T}{\partial x}. \quad (16.9)$$

Use your results for $\partial T / \partial x$ and Q to estimate κ . Because of the limited number of spins and Monte Carlo steps, your results should be accurate to only about 20%. More accurate results would require at least $N = 50$ spins and 10^4 to 10^5 Monte Carlo steps per spin.

- e. Determine Q , the temperature profile, and the mean temperature for different values of `nvisit`. Is the temperature profile linear for all `nvisit`? If the temperature profile is linear, estimate $\partial T / \partial x$ and determine κ . Does κ depend on the mean temperature?

Note that in Problem 16.6 we were able to compute a temperature profile by using an algorithm that manipulated only integer numbers. The conventional approach is to solve a heat equation similar in form to the diffusion equation.

Problem 16.7. Magnetization profile

- a. Modify `Program conduct` by removing `SUB heat` and constraining spins 1 and N to be +1 and -1 respectively. Estimate the magnetization profile by plotting the mean value of the spin at each site versus the site number. Choose $N = 22$ and $\text{mcs} \geq 1000$. How do your results vary as you increase N ?
- b. Compute the mean demon energy and hence the local temperature at each site. Does the system have a uniform temperature even though the magnetization is not uniform? Is the system in thermal equilibrium?
- c. The effect of this constraint is easier to observe in two and three dimensions than in one dimension. Write a program for a two-dimensional Ising model on a $L \times L$ square lattice.

Constrain the spins at site (i, j) to be $+1$ and -1 for $i = 1$ and $i = L$ respectively. Use periodic boundary conditions in the y direction. How do your results compare with the one-dimensional case?

- d. Remove the periodic boundary condition in the y direction and constrain all the boundary spins from $i = 1$ to $L/2$ to be $+1$ and the other boundary spins to be -1 . Choose an initial configuration where all the spins on the left half of the system are $+1$ and the others are -1 . Do the simulation and draw a configuration of the spins once the system has reached equilibrium. Draw a line between each pair of spins of opposite sign. Describe the curve separating the $+1$ spins from the -1 spins. Begin with $L = 20$ and determine what happens as L is increased.

16.8 Comment

One advantage of doing simulations using the demon algorithm is that it is not necessary to make any demands on the random number generator. We have done a Monte Carlo simulation without random numbers! (In the one-dimensional Ising model we have to choose the trial spins at random. However, the spins can be chosen sequentially in higher dimensions.) Very fast algorithms have been developed by using one computer bit per spin and multiple demons. There also are several disadvantages associated with the microcanonical ensemble. One disadvantage is the difficulty of establishing a system at the desired value of the energy. However, the most important disadvantage for us is conceptual. That is, it is more natural for us to think of the behavior of macroscopic physical quantities as functions of the temperature rather than the total energy. Hence, we postpone further consideration of the further properties of the Ising model to Chapter 17 in the context of the canonical ensemble.

Appendix A: Relation of the Mean Demon Energy to the Temperature

We know that the energy of the demon, E_d , is constrained to be positive and is given by $E_d = E_{total} - E$, where E is the energy of the system and E_{total} is the total energy. We have found in Problems 16.2 and ?? that the probability for the demon to have energy E_d is proportional to $e^{-E_d/kT}$. We assume that the same form of the probability distribution holds for any macroscopic system in thermodynamic equilibrium. Hence in general, $\langle E_d \rangle$ is given by

$$\langle E_d \rangle = \frac{\sum_{E_d} E_d e^{-E_d/kT}}{\sum_{E_d} e^{-E_d/kT}}, \quad (16.10)$$

where the summations in (16.10) are over the possible values of E_d . If an Ising spin is flipped in zero magnetic field, the minimum nonzero decrease in energy of the system is $4J$ (see Fig. 16.3). Hence the possible energies of the demon are $0, 4J, 8J, 12J, \dots$. We write $x = 4J/kT$ and perform the summations in (16.10). The result is

$$\langle E_d/kT \rangle = \frac{0 + xe^{-x} + 2xe^{-2x} + \dots}{1 + e^{-x} + e^{-2x} + \dots} = \frac{x}{e^x - 1}. \quad (16.11)$$

The form (16.8) can be obtained by solving (16.11) for T in terms of E_d . Convince yourself that the relation (16.11) is independent of dimension for lattices with an even number of nearest neighbors.

If the magnetic field is nonzero, the possible values of the demon energy are $0, 2H, 4J - 2H, 4J + 2H, \dots$. If J is a multiple of H , then the result is the same as before with $4J$ replaced by $2H$, because the possible energy values for the demon are multiples of $2H$. If the ratio $4J/2H$ is irrational, then the demon can take on a continuum of values, and thus $\langle E_d \rangle = kT$. The other possibility is that $4J/2H = m/n$, where m and n are relatively prime positive integers. In this case it can be shown that (see Mak)

$$kT/J = \frac{4/m}{\ln(1 + 4J/m\langle E_d \rangle)}. \quad (16.12)$$

You can test these relations for $H \neq 0$ by choosing values of J and H and computing the sums in (16.10) directly.

References and Suggestions for Further Reading

- S. G. Brush, “History of the Lenz-Ising model,” *Rev. Mod. Phys.* **39**, 883 (1967).
- Michael Creutz, “Microcanonical Monte Carlo simulation,” *Phys. Rev. Lett.* **50**, 1411 (1983). See also Gyan Bhanot, Michael Creutz, and Herbert Neuberger, “Microcanonical simulation of Ising systems,” *Nuc. Phys. B* **235**, 417 (1984).
- R. Harris, “Demons at work,” *Computers in Physics* 4(3), 314 (1990).
- S. S. Mak, “The analytical demon of the Ising model,” *Phys. Lett. A* **196**, 318 (1995).

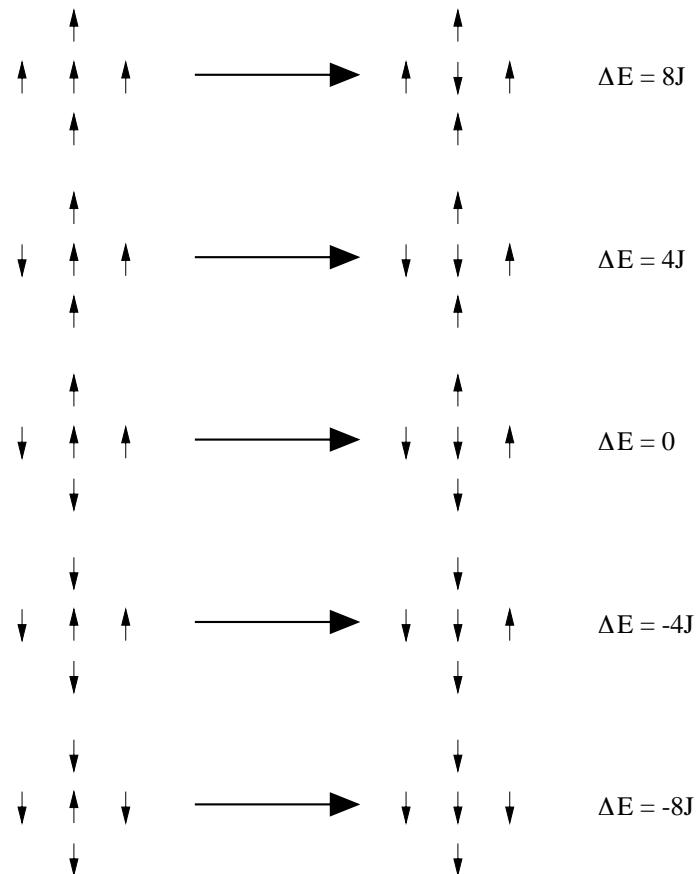


Figure 16.3: The five possible transitions of the Ising model on the square lattice with spin flip dynamics.

Chapter 17

Monte Carlo Simulation of the Canonical Ensemble

©2002 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
22 April 2002

We discuss Monte Carlo methods for simulating equilibrium systems. Applications are made to models of magnetism and simple fluids.

17.1 The Canonical Ensemble

Most physical systems are not isolated, but exchange energy with their environment. Because such systems are usually small in comparison to their environment, we assume that any change in the energy of the smaller system does not have a significant effect on the temperature of the environment. We say that the environment acts as a *heat reservoir* or *heat bath* at a fixed absolute temperature T . If a small but macroscopic system is placed in thermal contact with a heat bath, the system reaches thermal equilibrium by exchanging energy with the heat bath until the system attains the temperature of the bath.

Imagine an infinitely large number of copies of a system at fixed volume V and number of particles N in equilibrium at temperature T . In Chapter 16 we verified that P_s , the probability that the system is in microstate s with energy E_s , is given by

$$P_s = \frac{1}{Z} e^{-\beta E_s}, \text{(canonical distribution)} \quad (17.1)$$

where $\beta = 1/kT$, and Z is a normalization constant. The ensemble defined by (17.1) is known as the *canonical* ensemble. Because $\sum P_s = 1$, Z is given by

$$Z = \sum_{s=1}^M e^{-E_s/kT}. \quad (17.2)$$

The summation in (17.2) is over all M accessible microstates of the system. The quantity Z is known as the *partition function* of the system.

We can use (17.1) to obtain the ensemble average of the physical quantities of interest. For example, the mean energy is given by

$$\langle E \rangle = \sum_{s=1}^M E_s P_s = \frac{1}{Z} \sum_{s=1}^M E_s e^{-\beta E_s}. \quad (17.3)$$

Note that the energy fluctuates in the canonical ensemble.

17.2 The Metropolis Algorithm

How can we simulate a system of N particles confined in a volume V at a fixed temperature T ? Because we can generate only a finite number m of the total number of M microstates, we might hope to obtain an estimate for the mean value of the physical quantity A by writing

$$\langle A \rangle \approx A_m = \frac{\sum_{s=1}^m A_s e^{-\beta E_s}}{\sum_{s=1}^m e^{-\beta E_s}}. \quad (17.4)$$

A_s is the value of the physical quantity A in microstate s . A crude Monte Carlo procedure is to generate a microstate s at random, calculate E_s , A_s , and $e^{-\beta E_s}$, and evaluate the corresponding contribution of the microstate to the sums in (17.4). However, a microstate generated in this way would likely be very improbable and hence contribute little to the sums. Instead, we use an *importance sampling* method and generate microstates according to a probability distribution function π_s .

We follow the same procedure as in Section 11.7 and rewrite (17.4) by multiplying and dividing by π_s :

$$A_m = \frac{\sum_{s=1}^m (A_s/\pi_s) e^{-\beta E_s} \pi_s}{\sum_{s=1}^m (1/\pi_s) e^{-\beta E_s} \pi_s}. \quad (\text{no importance sampling}) \quad (17.5)$$

If we generate microstates with probability π_s , then (17.5) becomes

$$A_m = \frac{\sum_{s=1}^m (A_s/\pi_s) e^{-\beta E_s}}{\sum_{s=1}^m (1/\pi_s) e^{-\beta E_s}}. \quad (\text{importance sampling}) \quad (17.6)$$

That is, if we average over a biased sample, we need to weight each microstate by $1/\pi_s$ to eliminate the bias. Although any form of π_s could be used, the form of (17.6) suggests that a reasonable choice of π_s is the Boltzmann probability itself, that is,

$$\pi_s = \frac{e^{-\beta E_s}}{\sum_{s=1}^m e^{-\beta E_s}}. \quad (17.7)$$

This choice of π_s implies that the estimate A_m of the mean value of A can be written as

$$A_m = \frac{1}{m} \sum_{s=1}^m A_s. \quad (17.8)$$

The choice (17.7) for π_s is due to Metropolis et al.

Although we discussed the Metropolis sampling method in Section 11.8 in the context of the numerical evaluation of integrals, it is not necessary to read Section 11.8 to understand the Metropolis algorithm in the present context. The Metropolis algorithm can be summarized in the context of the simulation of a system of spins or particles as follows:

1. Establish an initial microstate.
2. Make a random trial change in the microstate. For example, choose a spin at random and flip it. Or choose a particle at random and displace it a random distance.
3. Compute $\Delta E \equiv E_{\text{trial}} - E_{\text{old}}$, the change in the energy of the system due to the trial change.
4. If ΔE is less than or equal to zero, accept the new microstate and go to step 8.
5. If ΔE is positive, compute the quantity $w = e^{-\beta \Delta E}$.
6. Generate a random number r in the unit interval.
7. If $r \leq w$, accept the new microstate; otherwise retain the previous microstate.
8. Determine the value of the desired physical quantities.
9. Repeat steps (2) through (8) to obtain a sufficient number of microstates.
10. Periodically compute averages over microstates.

Steps 2 through 7 give the conditional probability that the system is in microstate $\{s_j\}$ given that it was in microstate $\{s_i\}$. These steps are equivalent to the transition probability

$$W(i \rightarrow j) = \min(1, e^{-\beta \Delta E}), \quad (\text{Metropolis algorithm}) \quad (17.9)$$

where $\Delta E = E_j - E_i$. $W(i \rightarrow j)$ is the probability per unit time for the system to make a transition from microstate i to microstate j . Because it is necessary to evaluate only the ratio $P_j/P_i = e^{-\beta \Delta E}$, it is not necessary to normalize the probability. Note that because the microstates are generated with a probability proportional to the desired probability, all averages become arithmetic averages as in (17.8). However, because the constant of proportionality is not known, it is not possible to estimate the partition function Z in this way.

Although we choose π_s to be the Boltzmann distribution, other choices of π_s are possible and are useful in some contexts. In addition, the choice (17.9) of the transition probability is not the only one that leads to the Boltzmann distribution. It can be shown that if W satisfies the “detailed balance” condition

$$W(i \rightarrow j) e^{-\beta E_i} = W(j \rightarrow i) e^{-\beta E_j}, \quad (\text{detailed balance}) \quad (17.10)$$

then the corresponding Monte Carlo algorithm generates a sequence of states distributed according to the Boltzmann distribution. The derivation that the Metropolis algorithm generates states with a probability proportional to the Boltzmann probability distribution after a sufficient number of steps does not add much to our physical understanding of the algorithm. Instead, in Section 17.2 we apply the algorithm to the ideal classical gas and to a classical magnet in a magnetic field, and verify that the Metropolis algorithm yields the Boltzmann distribution after a sufficient number of trial changes have been made.

We have implicitly assumed in the above discussion that the system is ergodic. Ergodicity refers to the sampling of the important microstates of a system. In a Monte Carlo simulation, the existence of ergodicity depends on the way the trial moves are made, and on the nature of the energy barriers between microstates. For example, consider a one-dimensional lattice of Ising spins with all spins up. If the spins are updated sequentially from right to left, then if one spin is flipped, all remaining flips will be accepted regardless of the temperature because the change in energy is zero. Clearly, the system is not ergodic for this implementation of the algorithm, and we would not obtain the correct thermodynamic behavior.

17.3 Verification of the Boltzmann Distribution

We first consider the application of the Metropolis algorithm to an ideal classical gas in one dimension. The energy of an ideal gas depends only on the velocity of the particles, and hence a microstate is completely described by a specification of the velocity (or momentum) of each particle. Because the velocity is a continuous variable, it is necessary to describe the accessible microstates so that they are countable, and hence we place the velocity into bins. Suppose we have $N = 10$ particles and divide the possible values of the velocity into twenty bins. Then the total number of microstates would be 20^{10} . Not only would it be difficult to label these 20^{10} states, it would take a prohibitively long time to obtain an accurate estimate of their relative probabilities, and it would be difficult to verify directly that the Metropolis algorithm yields the Boltzmann distribution. For this reason we consider a single classical particle in one dimension in equilibrium with a heat bath and adopt the less ambitious goal of verifying that the Metropolis algorithm generates the Boltzmann distribution for this system. The quantity of interest is the probability $P(v)dv$ that the system has a velocity between v and $v + dv$. The algorithm is implemented in SUB `metropolis` in Program `boltzmann` listed below. The array `P` stores the desired probability. We choose units such that Boltzmann's constant and the mass are unity.

```
PROGRAM boltzmann
! Metropolis algorithm for a particle in one dimension
DIM P(-100 to 100),accum(3)
CALL initial(v,E,beta,mcs,nequil,delta,nbin,delv)
FOR imcs = 1 to nequil           ! equilibrate system
    CALL metropolis(v,E,beta,delta,accept)
NEXT imcs
CALL initialize_sums(P(),accum(),accept,nbin)
FOR imcs = 1 to mcs
    CALL metropolis(v,E,beta,delta,accept)
```

```

! accumulate data after each trial change
CALL data(P(),accum(),v,E,nbin,delv)
NEXT imcs
CALL output(P(),accum(),mcs,accept,nbin,delv)
END

SUB initial(v0,E0,beta,mcs,nequil,delta,nbin,delv)
  RANDOMIZE
  INPUT prompt "temperature = ": T
  LET beta = 1/T
  INPUT prompt "number of Monte Carlo steps = ": mcs
  LET nequil = int(0.1*mcs)
  INPUT prompt "initial speed = ": v0
  LET E0 = 0.5*v0*v0           ! initial kinetic energy
  INPUT prompt "maximum change in velocity = ": delta
  LET vmax = 10*sqr(T)
  LET nbin = 20                 ! number of bins
  LET delv = vmax/nbin         ! velocity interval
END SUB

SUB initialize_sums(P(),accum(),accept,nbin)
  FOR ibin = -nbin to nbin
    LET P(ibin) = 0
  NEXT ibin
  FOR i = 1 to 3
    LET accum(i) = 0
  NEXT i
  LET accept = 0
END SUB

SUB metropolis(v,E,beta,delta,accept)
  LET dv = (2*rnd - 1)*delta    ! trial velocity change
  LET vtrial = v + dv          ! trial velocity
  LET dE = 0.5*(vtrial*vtrial - v*v)    ! trial energy change
  IF dE > 0 then
    IF exp(-beta*dE) < rnd then
      EXIT SUB                  ! step not accepted
    END IF
  END IF
  LET v = vtrial
  LET accept = accept + 1
  LET E = E + dE
END SUB

SUB data(P(),accum(),v,E,nbin,delv)
  LET accum(1) = accum(1) + E

```

```

LET accum(2) = accum(2) + E*E
LET accum(3) = accum(3) + v
LET ibin = round(v/delv)
LET P(ibin) = P(ibin) + 1
END SUB

SUB output(P(),accum(),mcs,accept,nbin,delv)
  LET accept = accept/mcs
  PRINT "acceptance probability ="; accept
  LET vave = accum(3)/mcs
  PRINT "mean velocity ="; vave
  LET Eave = accum(1)/mcs
  PRINT "mean energy ="; Eave
  LET E2ave = accum(2)/mcs
  LET sigma2 = E2ave - Eave*Eave
  PRINT "sigma_E = "; sqr(sigma2)
  PRINT
  PRINT " v ", "P(v)"
  PRINT
  LET v = -nbin*delv
  FOR ibin = -nbin to nbin
    IF p(ibin) > 0 then
      LET prob = p(ibin)/mcs
      PRINT v,
      PRINT using "--.###": prob
    END IF
    LET v = v + delv
  NEXT ibin
END SUB

```

Problem 17.1. The Boltzmann distribution

- Use Program `boltzmann` to determine the form of the probability distribution that is generated by the Metropolis algorithm. Let the temperature $T = 4$, the initial velocity $v_0 = 0$, the maximum change in the particle's velocity $\delta = 4.0$, and the number of trial moves or Monte Carlo steps $mcs = 10000$. Compute the mean energy, the mean velocity, and the probability density $P(v)$.
- Is $P(v)$ an increasing or decreasing function of the energy $E = \frac{1}{2}v^2$? Increase the number of Monte Carlo steps until the Boltzmann form of $P(v)$ is approximately verified. Verify that a plot of $\ln P(v)$ versus E yields a straight line with a slope equal to $-1/T$.
- How do your results for the mean energy and the mean velocity compare with the corresponding exact values?
- To insure that your results do not depend on the initial conditions, let $v_0 = 2$ and compute the mean energy and velocity. How do your results compare with those found in part (a)? Explain

why the computed mean particle velocity is approximately zero even though the initial particle velocities are not zero.

- e. The *acceptance probability* is the fraction of trial moves that are accepted. What is the effect of changing the value of δ on the acceptance probability?

Problem 17.2. Planar spin in an external magnetic field

- a. Consider a classical planar magnet with magnetic moment μ_0 . The magnet can be oriented in any direction in the x - y plane, and the energy of interaction of the magnet with an external magnetic field \mathbf{B} is $-\mu_0 B \cos \phi$, where ϕ is the angle between the moment and \mathbf{B} . What are the possible microstates of this system? Write a Monte Carlo program to sample the microstates of this system in thermal equilibrium with a heat bath at temperature T . Compute the mean energy as a function of the ratio $\beta\mu_0 B$.
- b. Do an analytical calculation of the mean energy and compare the analytical and computed results for various values of $\beta\mu_0 B$.
- c. Compute the probability density $P(\phi)$ and analyze its dependence on the energy.

In Problem 17.3 we consider the Monte Carlo simulation of a classical ideal gas of N particles. It is convenient to say that one “time unit” or one “Monte Carlo step per particle” (MCS) has elapsed after N particles have had one chance each on the average to change their coordinates. If the particles are chosen at random, then during one Monte Carlo step per particle, some particles might not be chosen. Of course, all particles will be chosen equally on the average. The advantage of this definition is that the time is independent of the number of particles. However, this definition of time has no obvious relation to a physical time.

Problem 17.3. Simulation of an ideal gas in one dimension

- a. Modify **Program boltzmann** to simulate an ideal gas of N particles in one dimension. Assume all particles have the same initial velocity $v_0 = 10$. Let $N = 20$, $T = 10$, and $mcs = 200$. Choose the value of δ so that the acceptance probability is approximately 40%. What is the mean kinetic energy and mean velocity of the particles?
- b. We might expect the total energy of an ideal gas to remain constant because the particles do not interact with one another and hence cannot exchange energy directly. What is the value of the initial total energy of the system in part (a)? Does the total energy remain constant? If not, explain how the energy changes.
- c. What is the nature of the time dependence of the total energy starting from the initial condition in (a)? Estimate the number of Monte Carlo steps per particle necessary for the system to reach thermal equilibrium by computing a moving average of the total energy over a fixed time interval. Does this average change with time after a sufficient time has elapsed? What choice of the initial velocities allows the system to reach thermal equilibrium at temperature T as quickly as possible?
- d. Compute the probability $P(E) dE$ for the system of N particles to have a total energy between E and $E + dE$. Do you expect $\ln P(E)$ to depend linearly on E ? Plot $P(E)$ as a function of E

and describe the qualitative behavior of $P(E)$. If the plot of $\ln P(E)$ versus E does not yield a straight line, describe the qualitative features of the plot, and determine a functional form for $P(E)$.

- e. Compute the mean energy for $T = 10, 20, 40, 80$, and 120 and estimate the heat capacity from its definition $C = \partial E / \partial T$.
- f. Compute the mean square energy fluctuations $\langle (\Delta E)^2 \rangle = \langle E^2 \rangle - \langle E \rangle^2$ for $T = 10$ and $T = 40$. Compare the magnitude of the ratio $\langle (\Delta E)^2 \rangle / T^2$ with the heat capacity determined in part (e).

You might have been surprised to find in Problem 17.3d that the form of $P(E)$ is a Gaussian centered about the mean energy of the system. That is, the distribution function of a *macroscopic* quantity such as the total energy is sharply peaked about its mean value. If the microstates are distributed according to the Boltzmann probability, why is the total energy distributed according to the Gaussian distribution?

17.4 The Ising Model

One of the more interesting natural phenomena in nature is magnetism. You are probably familiar with ferromagnetic materials such as iron and nickel which exhibit a spontaneous magnetization in the absence of an applied magnetic field. This nonzero magnetization occurs only if the temperature is lower than a well defined temperature known as the Curie or critical temperature T_c . For temperatures $T > T_c$, the magnetization vanishes. Hence T_c separates the disordered phase for $T > T_c$ from the ferromagnetic phase for $T < T_c$.

The origin of magnetism is quantum mechanical in nature and an area of much experimental and theoretical interest. However, the study of simple classical models of magnetism has provided much insight. The two- and three-dimensional Ising model is the most commonly studied classical model and is particularly useful in the neighborhood of the magnetic phase transition. As discussed in Chapter 16, the energy of the Ising model is given by

$$E = -J \sum_{i,j=\text{nn}(i)}^N s_i s_j - \mu_0 B \sum_{i=1}^N s_i, \quad (17.11)$$

where $s = \pm 1$, J is a measure of the strength of the interaction between spins, and the first sum is over all pairs of spins that are nearest neighbors. The second term in (17.11) is the energy of interaction of the magnetic moment with an external magnetic field. Because of the neglect of the other spin components, the Ising model does not give a complete description of ferromagnetism, especially at temperatures well below T_c .

The thermal quantities of interest for the Ising model include the mean energy $\langle E \rangle$ and the heat capacity C . As we have discussed, one way to determine C at constant external magnetic field is from its definition $C = \partial \langle E \rangle / \partial T$. An alternative way of determining C is to relate it to the statistical fluctuations of the total energy in the canonical ensemble (see Appendix 17.31):

$$C = \frac{1}{kT^2} (\langle E^2 \rangle - \langle E \rangle^2). \quad (17.12)$$

Another quantity of interest is the mean magnetization $\langle M \rangle$ (see (16.7)) and the corresponding thermodynamic derivative χ :

$$\chi = \lim_{H \rightarrow 0} \frac{\partial \langle M \rangle}{\partial H}, \quad (17.13)$$

where H is proportional to the external magnetic field. In the following, we will refer to H as the magnetic field. The zero field magnetic susceptibility χ is an example of a linear response function, because it measures the ability of a spin to “respond” due to a change in the external magnetic field. In analogy to the heat capacity, χ is related to the fluctuations of the magnetization (see Appendix 17.31):

$$\chi = \frac{1}{kT} (\langle M^2 \rangle - \langle M \rangle^2), \quad (17.14)$$

where $\langle M \rangle$ and $\langle M^2 \rangle$ are evaluated in zero magnetic fields. Relations (17.12) and (17.14) are examples of the general relation between linear response functions and equilibrium fluctuations.

Now that we have specified several equilibrium quantities of interest, we implement the Metropolis algorithm for the Ising model. The possible trial change is the flip of a spin, $s_i \rightarrow -s_i$. The Metropolis algorithm was stated in Section 17.2 as a method for generating states with the desired Boltzmann probability, but the flipping of single spins also can be interpreted as a reasonable approximation to the real dynamics of an anisotropic magnet whose spins are coupled to the vibrations of the lattice. The coupling leads to random spin flips, and we expect that one Monte Carlo step per spin is proportional to the average time between single spin flips observed in the laboratory. We can regard single spin flip dynamics as a time dependent process and observe the relaxation to equilibrium after a sufficiently long time. In the following, we will frequently refer to the application of the Metropolis algorithm to the Ising model as “single spin flip dynamics.”

In Problem 17.4 we use the Metropolis algorithm to simulate the one-dimensional Ising model. Note that the parameters J and kT do not appear separately, but appear together in the dimensionless ratio J/kT . Unless otherwise stated, we measure temperature in units of J/k , and set $H = 0$.

Problem 17.4. One-dimensional Ising model

- Write a Monte Carlo program to simulate the one-dimensional Ising model in equilibrium with a heat bath. (Modify `SUB changes` in `Program demon` (see Chapter 16) or see `Program ising`, listed in the following, for an example of the implementation of the Metropolis algorithm to the two-dimensional Ising model.) Use periodic boundary conditions. As a test of your program, compute the mean energy and magnetization of the lattice for $N = 20$ and $T = 1$. Draw the microscopic state (configuration) of the system after each Monte Carlo step per spin.
- Choose $N = 20$, $T = 1$, $mcs = 100$, and all spins up, that is, $s_i = +1$ initially. What is the initial “temperature” of the system? Visually inspect the configuration of the system after each Monte Carlo step and estimate the time it takes for the system to reach equilibrium. Then change the initial condition so that the orientation of the spins is chosen at random. What is the initial “temperature” of the system in this case? Estimate the time it takes for the system to reach equilibrium in the same way as before.

- c. Choose $N = 20$ and equilibrate the system for $\text{mcs} \geq 100$. Let $\text{mcs} \geq 1000$ and determine $\langle E \rangle$, $\langle E^2 \rangle$, $\langle M \rangle$, and $\langle M^2 \rangle$ as a function of T in the range $0.1 \leq T \leq 5$. Plot $\langle E \rangle$ as a function of T and discuss its qualitative features. Compare your computed results for $\langle E(T) \rangle$ to the exact result (for $H = 0$)

$$\langle E \rangle = -N \tanh \beta J. \quad (17.15)$$

Use the relation (17.12) to determine the T dependence of C .

- d. What is the qualitative dependence of $\langle M \rangle$ on T ? Use the relation (17.14) to estimate the T dependence of χ . One of the best laboratory realizations of a one-dimensional Ising ferromagnet is a chain of bichloride-bridged Fe^{2+} ions known as FeTAC (Greene et al.). Measurements of χ yield a value of the exchange interaction J given by $J/k = 17.4\text{ K}$. (Experimental values of J are typically given in temperature units.) Use this value of J to plot your Monte Carlo results for χ versus T with T given in Kelvin. At what temperature is χ a maximum for FeTAC?
- e. Is the acceptance probability an increasing or decreasing function of T ? Does the Metropolis algorithm become more or less efficient as the temperature is lowered?
- f. Compute the probability density $P(E)$ for a system of 50 spins at $T = 1$. Choose $\text{mcs} \geq 1000$. Plot $\ln P(E)$ versus $(E - \langle E \rangle)^2$ and discuss its qualitative features.

We next apply the Metropolis algorithm to the two-dimensional Ising model on the square lattice. The main program is listed in the following.

```
PROGRAM ising
! Monte Carlo simulation of the Ising model on the square lattice
! using the Metropolis algorithm
DIM spin(32,32),w(-8 to 8),accum(10)
LIBRARY "csgraphics"
CALL initial(N,L,T,spin(),,mcs,nequil,w(),E,M)
FOR i = 1 to nequil      ! equilibrate system
    CALL Metropolis(N,L,spin(),,E,M,w(),accept)
NEXT i
CALL initialize(accum(),accept)
FOR pass = 1 to mcs      ! accumulate data while updating spins
    CALL Metropolis(N,L,spin(),,E,M,w(),accept)
    CALL data(E,M,accum())
NEXT pass
CALL output(T,N,mcs,accum(),accept)
END
```

In SUB `initial` we choose the initial directions of the spins, and compute the initial values of the energy and magnetization. To compute the total energy, we consider the interaction of a spin with its nearest neighbor spins to the north and the east. In this way we compute the energy of each interaction only once and avoid double counting. One of the most time consuming parts of the Metropolis algorithm is the calculation of the exponential function $e^{-\beta \Delta E}$. Because there are only a small number of possible values of $\beta \Delta E$ for the Ising model (see Fig. 16.3), we store the

small number of different probabilities for the spin flips in the array `w`. The values of this array are computed in `SUB initial`.

```

SUB initial(N,L,T,spin(,),mcs,nequil,w(),E,M)
  RANDOMIZE
  INPUT prompt "linear dimension of lattice = ": L
  LET N = L*L                      ! number of spins
  ! temperature measured in units of J/k
  INPUT prompt "temperature = ": T
  INPUT prompt "# MC steps per spin for equilibrium = ": nequil
  INPUT prompt "# MC steps per spin for data = ": mcs
  LET M = 0
  FOR y = 1 to L                  ! random initial configuration
    FOR x = 1 to L
      IF rnd < 0.5 then
        LET spin(x,y) = 1 ! spin up
      ELSE
        LET spin(x,y) = -1
      END IF
      LET M = M + spin(x,y)      ! total magnetization
    NEXT x
  NEXT y
  LET E = 0
  FOR y = 1 to L                  ! compute initial energy
    IF y = L then
      LET up = 1                ! periodic boundary conditions
    ELSE
      LET up = y + 1
    END IF
    FOR x = 1 to L
      IF x = L then
        LET right = 1
      ELSE
        LET right = x + 1
      END IF
      LET sum = spin(x,up) + spin(right,y)
      LET E = E - spin(x,y)*sum ! total energy
    NEXT x
  NEXT y
  ! compute Boltzmann probability ratios
  FOR dE = -8 to 8 step 4
    LET w(dE) = exp(-dE/T)
  NEXT dE
END SUB

```

One way to implement the Metropolis algorithm is to determine the change in the energy ΔE

and then accept the trial flip if $\Delta E \leq 0$. If this condition is not satisfied, the second step is to generate a random number in the unit interval and compare it to $e^{-\beta\Delta E}$. Instead of this two step process, we implement the Metropolis algorithm in one step. Which method do you think is faster?

```
SUB Metropolis(N,L,spin(,),E,M,w(),accept)
  DECLARE DEF DeltaE
  ! one Monte Carlo step per spin
  FOR ispin = 1 to N
    LET x = int(L*rnd + 1)      ! random x coordinate
    LET y = int(L*rnd + 1)      ! random y coordinate
    LET dE = DeltaE(x,y,L,spin(,))      ! compute change in energy
    IF rnd <= w(dE) then
      LET spin(x,y) = -spin(x,y)  ! flip spin
      LET accept = accept + 1
      LET M = M + 2*spin(x,y)
      LET E = E + dE
    END IF
  NEXT ispin
END SUB
```

A typical laboratory system has at least 10^{18} spins. In contrast, the number of spins that can be simulated typically ranges from 10^3 to 10^9 . As we have discussed in other contexts, the use of periodic boundary conditions minimizes finite size effects. However, periodic boundary conditions reduce the maximum separation between spins to one half the length of the system, and more sophisticated boundary conditions are sometimes convenient. For example, we can give the surface spins extra neighbors, whose direction is related to the mean magnetization of the microstate. We adopt the simpler periodic boundary conditions in FUNCTION DeltaE in which the change in energy dE of flipping a spin is computed.

```
FUNCTION DeltaE(x,y,L,spin(,))
  ! periodic boundary conditions
  IF x = 1 then
    LET left = spin(L,y)
  ELSE
    LET left = spin(x - 1,y)
  END IF
  IF x = L then
    LET right = spin(1,y)
  ELSE
    LET right = spin(x + 1,y)
  END IF
  IF y = 1 then
    LET down = spin(x,L)
  ELSE
    LET down = spin(x,y - 1)
  END IF
```

```

IF y = L then
    LET up = spin(x,1)
ELSE
    LET up = spin(x,y + 1)
END IF
LET DeltaE = 2*spin(x,y)*(left + right + up + down)
END DEF

```

SUB `data` is called from the main program and the values of the physical observables are recorded after each Monte Carlo step per spin. The optimum time for sampling various physical quantities is explored in Problem 17.6. Note that if a flip is rejected and the old configuration is retained, thermal equilibrium is not described properly unless the old configuration is included again in computing the averages. Various variables are initialized in SUB `initialize`.

```

SUB initialize(accum(),accept)
    ! use array to save accumulated values of magnetization and
    ! energy. Array used to make it easier to add other quantities
    FOR i = 1 to 5
        LET accum(i) = 0
    NEXT i
    LET accept = 0
END SUB

SUB data(E,M,accum())
    ! accumulate data after every Monte Carlo step per spin
    LET accum(1) = accum(1) + E
    LET accum(2) = accum(2) + E*E
    LET accum(3) = accum(3) + M
    LET accum(4) = accum(4) + M*M
    LET accum(5) = accum(5) + abs(M)
END SUB

```

At the end of a run various averages are normalized and printed in SUB `output`. All averages such as the mean energy and the mean magnetization are normalized by the number of spins.

```

SUB output(T,N,mcs,accum(),accept)
    LET norm = 1/(mcs*N)           ! averages per spin
    LET accept = accept*norm
    LET eave = accum(1)*norm
    LET e2ave = accum(2)*norm
    LET mave = accum(3)*norm
    LET m2ave = accum(4)*norm
    LET abs_mave = accum(5)*norm
    CLEAR
    SET BACKGROUND COLOR "black"
    SET COLOR "yellow"

```

```

PRINT "temperature = "; T
PRINT "acceptance probability = "; accept
PRINT "mean energy per spin = "; eave
PRINT "mean squared energy per spin = "; e2ave
PRINT "mean magnetization per spin = "; mave
PRINT "mean of absolute magnetization per spin = "; abs_mave
PRINT "mean squared magnetization per spin = "; m2ave
END SUB

```

Achieving thermal equilibrium can account for a substantial fraction of the total run time. The most practical choice of initial conditions is a configuration from a previous run that is at a temperature close to the desired temperature. The following subroutine saves the last configuration of a run and can be included at the end of the main loop in **Program ising**.

```

SUB save_config(N,L,T,spin())
  INPUT prompt "name of file for last configuration = ": file$
  OPEN #2: name file$, access output, create new
  PRINT #2: T
  FOR y = 1 to L
    FOR x = 1 to L
      PRINT #2: spin(x,y)
    NEXT x
  NEXT y
  CLOSE #2
END SUB

```

A previous configuration can be used in a later run by adding a few statements to **SUB initial** to allow the user to choose a previous configuration or a random configuration. A previous configuration can be read by calling the following subroutine:

```

SUB read_config(N,L,T,spin())
  INPUT prompt "filename?": file$
  OPEN #1: name file$, access input
  INPUT #1: T
  FOR y = 1 to L
    FOR x = 1 to L
      INPUT #1: spin(x,y)
    NEXT x
  NEXT y
  CLOSE #1
END SUB

```

Problem 17.5. Equilibration of the two-dimensional Ising model

- Run **Program ising** with the linear dimension of the lattice $L = 16$ and the heat bath temperature $T = 2$. Determine the time, **nequil**, needed to equilibrate the system, if the directions of the spins are initially chosen at random. Plot the values of E and M after each Monte Carlo

- step per spin. Estimate how many Monte Carlo steps per spin are necessary for the system to reach equilibrium.
- Write a subroutine that shows the spin configurations on the screen. One simple way to do so is to draw a solid square about each spin site and color code the orientation of the spins. Is the system “ordered” or “disordered” at $T = 2$ after equilibrium has been established?
 - Repeat part (a) with all spins initially up. Does the equilibration time increase or decrease?
 - Repeat parts (a)–(c) for $T = 2.5$.

Problem 17.6. Comparison with exact results

In general, a Monte Carlo simulation yields exact answers only after an infinite number of configurations have been sampled. How then can we be sure our program works correctly, and our results are statistically meaningful? One check is to ensure that our program can reproduce exact results in known limits. In the following, we test **Program ising** by considering a small system for which the mean energy and magnetization can be calculated analytically.

- Calculate analytically the T dependence of E , M , C and χ for the two-dimensional Ising model with $L = 2$ and periodic boundary conditions. (A summary of the calculation is given in Appendix 17.31.)
- Use **Program ising** with $L = 2$ and estimate E , M , C , and χ for $T = 0.5$ and 0.25 . Use the relations (17.12) and (17.14) to compute C and χ , respectively. Compare your estimated values to the exact results found in part (a). Approximately how many Monte Carlo steps per spin are necessary to obtain E and M to within 1%? How many Monte Carlo steps per spin are necessary to obtain C and χ to within 1%?

Now that we have checked our program and obtained typical equilibrium configurations, we consider the calculation of the mean values of the physical quantities of interest. Suppose we wish to compute the mean value of the physical quantity A . In general, the calculation of A is time consuming, and we do not want to compute its value more often than necessary. For example, we would not compute A after the flip of only one spin, because the values of A in the two configurations would almost be the same. Ideally, we wish to compute A for configurations that are statistically independent. Because we do not know *a priori* the mean number of spin flips needed to obtain configurations that are statistically independent, it is a good idea to estimate this time in our preliminary calculations.

One way to estimate the time interval over which configurations are correlated is to compute the time displaced *autocorrelation* function $C_A(t)$ defined as

$$C_A(t) = \frac{\langle A(t + t_0)A(t_0) \rangle - \langle A \rangle^2}{\langle A^2 \rangle - \langle A \rangle^2}. \quad (17.16)$$

$A(t)$ is the value of the quantity A at time t . The averages in (17.16) are over all possible time origins t_0 for an equilibrium system. Because the choice of the time origin is arbitrary for an equilibrium system, C_A depends only on the time difference t rather than t and t_0 separately. For sufficiently large t , $A(t)$ and $A(0)$ will become uncorrelated, and hence $\langle A(t + t_0)A(t_0) \rangle \rightarrow$

$\langle A(t+t_0) \rangle \langle A(t_0) \rangle = \langle A \rangle^2$. Hence $C_A(t) \rightarrow 0$ as $t \rightarrow \infty$. In general, $C_A(t)$ will decay exponentially with t with a decay or correlation time τ_A whose magnitude depends on the choice of the physical quantity A as well as the physical parameters of the system, for example, the temperature. Note that $C_A(t=0)$ is normalized to unity.

The time dependence of the two most common correlation functions, $C_M(t)$ and $C_E(t)$ is investigated in Problem 17.7. As an example of the calculation of $C_E(t)$, consider the equilibrium time series for E for the $L = 4$ Ising model at $T = 4$: $-4, -8, 0, -8, -20, -4, 0, 0, -24, -32, -24, -24, -8, -8, -16, -12$. The averages of E and E^2 over these sixteen values are $\langle E \rangle = -12$, $\langle E^2 \rangle = 240$, and $\langle E^2 \rangle - \langle E \rangle^2 = 96$. We wish to compute $E(t)E(0)$ for all possible choices of the time origin. For example, $E(t=4)E(0)$ is given by

$$\begin{aligned} \langle E(t=4)E(0) \rangle &= \frac{1}{12} [(-20 \times -4) + (-4 \times -8) + (0 \times 0) \\ &\quad + (0 \times -8) + (-24 \times -20) + (-32 \times -4) \\ &\quad + (-24 \times 0) + (-24 \times 0) + (-8 \times -24) \\ &\quad + (-8 \times -32) + (-16 \times -24) + (-12 \times -24)]. \end{aligned} \quad (17.17)$$

We averaged over the twelve possible choices of the origin for the time difference $t = 4$. Verify that $\langle E(t=4)E(0) \rangle = 460/3$ and $C_E(t=4) = 7/72$.

In the above calculation of $\langle E(t)E(0) \rangle$, we included all possible combinations of $E(t)E(0)$ for a given time series. To implement this procedure on a computer, we would need to store the time series in memory or in a data file. An alternative procedure is to save the last `nsave` values of the time series in memory and to average over fewer combinations. This procedure is implemented in `SUB correl`; the correlation functions are computed and printed in `SUB c_output`. `SUB correl` uses two arrays, `Esave` and `Msave`, to store the last `nsave` values of the energy and the magnetization at each Monte Carlo step per spin. These arrays and the arrays `Ce` and `Cm` may be initialized in a separate subroutine.

```
SUB correl(Ce(),Cm(),E,M,esave(),msave(),pass,nsave)
    ! accumulate data for time correlation functions
    ! save last nsave values of M and E
    ! index0 = array index for earliest saved time
    IF pass > nsave then
        ! compute Ce and Cm after nsave values are saved
        LET index0 = mod(pass-1,nsave) + 1
        LET index = index0
        FOR tdiff = nsave to 1 step -1
            LET Ce(tdiff) = Ce(tdiff) + E*esave(index)
            LET Cm(tdiff) = Cm(tdiff) + M*msave(index)
            LET index = index + 1
            IF index > nsave then LET index = 1
        NEXT tdiff
    END IF
    ! save latest value in array position of earliest value
    LET esave(index0) = E
    LET msave(index0) = M
```

```

END SUB

SUB c_output(N,Ce(),Cm(),accum(),mcs,nsave)
  ! compute time correlation functions
  LET ebar = accum(1)/mcs
  LET e2bar = accum(2)/mcs
  LET Ce(0) = e2bar - ebar*ebar
  LET mbar = accum(3)/mcs
  LET m2bar = accum(4)/mcs
  LET Cm(0) = m2bar - mbar*mbar
  LET norm = 1/(mcs - nsave)
  PRINT
  PRINT "t","Ce(t)","Cm(t)"
  PRINT
  FOR tdiff = 1 to nsave
    ! correlation functions defined so that C(t=0) = 1
    ! and C(infinity) = 0
    LET Ce(tdiff) = (Ce(tdiff)*norm - ebar*ebar)/Ce(0)
    LET Cm(tdiff) = (Cm(tdiff)*norm - mbar*mbar)/Cm(0)
    PRINT tdiff,Ce(tdiff),Cm(tdiff)
  NEXT tdiff
END SUB

```

Problem 17.7. Correlation times

- Choose $L = 4$ and $T = 3$ and equilibrate the system. Then look at the time series of M and E after every Monte Carlo step per spin and estimate how often M changes sign. Does E change sign when M changes sign? How often does M change sign for $L = 8$ (and $T = 3$)? In equilibrium, positive and negative values of M are equally likely in the absence of an external magnetic field. Is your time series consistent with this equilibrium property? Why is it more meaningful to compute the time displaced correlation function of the absolute value of the magnetization rather than the magnetization itself if L is relatively small?
- Choose $L = 16$ and $T = 1$ and equilibrate the system. Then look at the time series of M . Do you find that positive and negative values of M are equally likely? Explain your results.
- Modify **Program ising** so that the equilibrium averaged values of $C_M(t)$ and $C_E(t)$ are computed. As a check on your program, use the time series for E given in the text to do a hand calculation of $C_E(t)$ in the way that it is computed in **SUB correl** and **SUB c_output**. Choose $nsave = 10$.
- Estimate the correlation times from the energy and the magnetization correlation functions for $L = 8$, and $T = 3$, $T = 2.3$, and $T = 2$. Save the last $nsave = 100$ values of the magnetization and energy only after the system is equilibrated. Are the correlation times τ_M and τ_E comparable? One way to determine τ is to fit $C(t)$ to an assumed exponential form $C(t) \sim e^{-t/\tau}$. Another way is to define the integrated correlation time as

$$\tau = \sum_{t=1} C(t). \quad (17.18)$$

The sum is cut off at the first negative value of $C(t)$. Are the negative values of $C(t)$ physically meaningful? How does the behavior of $C(t)$ change if you average your results over longer runs? How do your estimates for the correlation times compare with your estimates of the relaxation time found in Problem 17.5? Why would the term “decorrelation time” be more appropriate than “correlation time?”

- e. To describe the relaxation towards equilibrium as realistically as possible, we have randomly selected the spins to be flipped. However, if we are interested only in equilibrium properties, it might be possible to save computer time by selecting the spins sequentially. Determine if the correlation time is greater, smaller, or approximately the same if the spins are chosen sequentially rather than randomly. If the correlation time is greater, does it still save CPU time to choose spins sequentially? Why is it not desirable to choose spins sequentially in the one-dimensional Ising model?

Problem 17.8. Estimate of errors

How can we quantify the accuracy of our measurements, for example, the accuracy of the mean energy $\langle E \rangle$? As discussed in Chapter 11, the usual measure of the accuracy is the standard deviation of the mean. If we make n independent measurements of E , then the most probable error is given by

$$\sigma_m = \frac{\sigma}{\sqrt{n-1}}, \quad (17.19)$$

where the standard deviation σ is defined as

$$\sigma^2 = \langle E^2 \rangle - \langle E \rangle^2. \quad (17.20)$$

The difficulty is that, in general, our measurements of the time series E_i are not independent, but are correlated. Hence, σ_m as given by (17.19) is an underestimate of the actual error.

How can we determine whether the measurements are independent without computing the correlation time? One way is based on the idea that the magnitude of the error should not depend on how we group the data. For example, suppose that we group every two data points to form $n/2$ new data points $E_i^{(2)}$ given by $E_i^{(g=2)} = (1/2)[E_{2i-1} + E_{2i}]$. If we replace n by $n/2$ and E by $E^{(2)}$ in (17.19) and (17.20), we would find the same value of σ_m as before provided that the original E_i are independent. If the computed σ_m is not the same, we continue this averaging process until σ_m calculated from

$$E_i^{(g)} = \frac{1}{2}[E_{2i-1}^{(g/2)} + E_{2i}^{(g/2)}] \quad (g = 2, 4, 8, \dots) \quad (17.21)$$

is approximately the same as that calculated from $E^{(g/2)}$.

- a. Use the above averaging method to estimate the errors in your measurements of $\langle E \rangle$ and $\langle M \rangle$ for the two-dimensional Ising model. Let $L = 8$, $T = 2.269$, and $\text{mcs} \geq 16384$, and calculate averages after every Monte Carlo step per spin after the system has equilibrated. If necessary, increase the number of Monte Carlo steps for averaging. A rough measure of the correlation time is the number of terms in the time series that need to be averaged for σ_m to be approximately unchanged. What is the qualitative dependence of the correlation time on $T - T_c$?

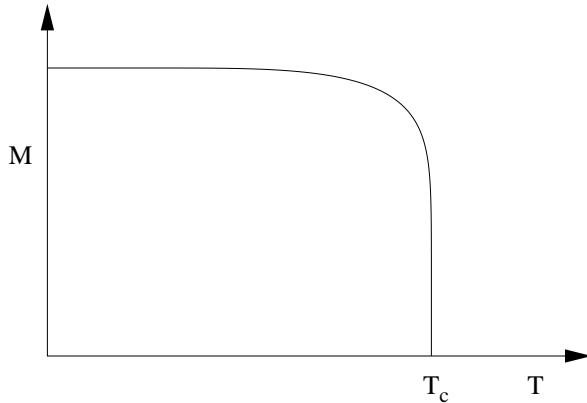


Figure 17.1: The temperature dependence of $m(T)$, the mean magnetization per spin, for the infinite lattice Ising model in two dimensions.

- b. Repeat for $L = 16$. Do you need more Monte Carlo steps than in part (a) to obtain statistically independent data? If so, why?
- c. The exact value of E/N for the two-dimensional Ising model on a square lattice with $L = 16$ and $T = T_c = 2/\ln(1 + \sqrt{2}) \approx 2.269$ is given by $E/N = -1.45306$ (to five decimal places). This value of T_c is exact for the infinite lattice. The exact result for E/N allows us to determine the actual error in this case. Compute $\langle E \rangle$ by averaging E after each Monte Carlo step per spin for $mcs \geq 10^6$. Compare your actual error to the estimated error given by (17.19) and (17.20) and discuss their relative values.

17.5 The Ising Phase Transition

Now that we have tested our program for the two-dimensional Ising model, we are ready to explore its properties. We first summarize some of the qualitative properties of infinite ferromagnetic systems in zero magnetic field. We know that at $T = 0$, the spins are perfectly aligned in either direction, that is, the mean magnetization per spin $m(T) = \langle M(T) \rangle / N$ is given by $m(T = 0) = \pm 1$. As T is increased, the magnitude of $m(T)$ decreases continuously until $T = T_c$ at which $m(T)$ vanishes (see Fig. 17.1). Because $m(T)$ vanishes continuously rather than abruptly, the transition is termed *continuous* rather than discontinuous. (The term *first-order* describes a discontinuous transition.)

How can we characterize a continuous magnetic phase transition? Because a nonzero m implies that a net number of spins are spontaneously aligned, we designate m as the *order parameter* of the system. Near T_c , we can characterize the behavior of many physical quantities by power law behavior just as we characterized the percolation threshold (see Table 13.1). For example, we can

write m near T_c as

$$m(T) \sim (T_c - T)^\beta, \quad (17.22)$$

where β is a critical exponent (not to be confused with the inverse temperature). Various thermodynamic derivatives such as the susceptibility and heat capacity diverge at T_c . We write

$$\chi \sim |T - T_c|^{-\gamma} \quad (17.23)$$

and

$$C \sim |T - T_c|^{-\alpha}. \quad (17.24)$$

We have assumed that χ and C are characterized by the same critical exponents above and below T_c .

Another measure of the magnetic fluctuations is the linear dimension $\xi(T)$ of a typical magnetic domain. We expect the *correlation length* $\xi(T)$ to be the order of a lattice spacing for $T \gg T_c$. Because the alignment of the spins becomes more correlated as T approaches T_c from above, $\xi(T)$ increases as T approaches T_c . We can characterize the divergent behavior of $\xi(T)$ near T_c by the critical exponent ν :

$$\xi(T) \sim |T - T_c|^{-\nu}. \quad (17.25)$$

The calculation of ξ is considered in Problem 17.9d.

As we found in our discussion of percolation in Chapter 13, a finite system cannot exhibit a true phase transition. Nevertheless, we expect that if $\xi(T)$ is less than the linear dimension L of the system, our simulations will yield results comparable to an infinite system. Of course, if T is close to T_c , our simulations will be limited by finite size effects. In the following problem, we obtain preliminary results for the T dependence of m , $\langle E \rangle$, C , and χ in the neighborhood of T_c . These results will help us understand the qualitative nature of the ferromagnetic phase transition in the two-dimensional Ising model.

Because we will consider the Ising model for different values of L , it will be convenient to compute intensive quantities such as the mean energy per spin, the specific heat (per spin) and the susceptibility per spin. We will retain the same notation for both the extensive and corresponding intensive quantities.

Problem 17.9. Qualitative behavior of the two-dimensional Ising model

- Use **Program ising** to compute the magnetization per spin m , the mean energy per spin $\langle E \rangle$, the specific heat C , and the susceptibility per spin χ . Choose $L = 4$ and consider T in the range $1.5 \leq T \leq 3.5$ in steps of $\Delta T = 0.2$. Choose the initial condition at $T = 3.5$ so that the orientation of the spins is chosen at random. Use an equilibrium configuration from a previous run at temperature T as the initial configuration for a run at temperature $T - \Delta T$. Because all the spins might overturn and the magnetization change sign during the course of your observation, estimate the mean value of $|m|$ in addition to that of m . Use at least 1000 Monte Carlo steps per spin and estimate the number of equilibrium configurations needed to obtain m and $\langle E \rangle$ to 5% accuracy. Plot $\langle E \rangle$, m , $|m|$, C , and χ as a function of T and describe their qualitative behavior. Do you see any evidence of a phase transition?

- b. Repeat the calculations of part (a) for $L = 8$ and $L = 16$. Plot $\langle E \rangle$, m , $|m|$, C , and χ as a function of T and describe their qualitative behavior. Do you see any evidence of a phase transition? For comparison, recent published Monte Carlo results for the two-dimensional Ising model are in the range $L = 10^2$ to $L = 10^3$ with order 10^6 Monte Carlo steps per spin.
- c. For a given value of L , for example, $L = 16$, choose a value of T that is well below T_c and choose the directions of the spins at random. Observe the spins evolve in time. Do you see several domains with positive and negative spontaneous magnetization? How does the magnetization evolve with time?
- d. The correlation length ξ can be obtained from the r -dependence of the spin correlation function $c(r)$. The latter is defined as:

$$c(r) = \langle s_i s_j \rangle - m^2, \quad (17.26)$$

where r is the distance between sites i and j . We have assumed the system is translationally invariant so that $\langle s_i \rangle = \langle s_j \rangle = m$. The average is over all sites for a given configuration and over many configurations. Because the spins are not correlated for large r , we see that $c(r) \rightarrow 0$ in this limit. It is reasonable to assume that $c(r) \sim e^{-r/\xi}$ for r sufficiently large. Use this behavior to estimate ξ as a function of T . How does your estimate of ξ compare with the size of the regions of spins with the same orientation?

One of the limitations of a computer simulation study of a phase transition is the relatively small size of the systems we can study. Nevertheless, we observed in Problem 17.9 that even systems as small as $L = 4$ exhibit behavior that is reminiscent of a phase transition. In Fig. 17.2 we show our Monte Carlo data for the T dependence of the specific heat of the two-dimensional Ising model for $L = 8$ and $L = 16$. We see that C exhibits a broad maximum which becomes sharper for larger L . Does your data for C exhibit similar behavior?

Because we can simulate only finite lattices, it is difficult to obtain estimates for the critical exponents α , β , and γ by using the definitions (17.22)–(17.24) directly. We learned in Section 13.4, we can do a *finite size scaling analysis* to extrapolate finite L results to $L \rightarrow \infty$. For example, from Fig. 17.2 we see that the temperature at which C exhibits a maximum becomes better defined for larger lattices. This behavior provides a simple definition of the transition temperature $T_c(L)$ for a finite system. According to finite size scaling theory, $T_c(L)$ scales as

$$T_c(L) - T_c(L = \infty) \sim aL^{-1/\nu}, \quad (17.27)$$

where a is a constant and ν is defined in (17.25). The finite size of the lattice is important when the correlation length

$$\xi(T) \sim L \sim |T - T_c|^{-\nu}. \quad (17.28)$$

As in Section 13.4, we can set $T = T_c$ and consider the L -dependence of M , C , and χ :

$$m(T) \sim (T_c - T)^\beta \rightarrow L^{-\beta/\nu} \quad (17.29)$$

$$C(T) \sim |T - T_c|^{-\alpha} \rightarrow L^{\alpha/\nu} \quad (17.30)$$

$$\chi(T) \sim |T - T_c|^{-\gamma} \rightarrow L^{\gamma/\nu}. \quad (17.31)$$

In Problem 17.10 we use the relations (17.29)–(17.31) to estimate the critical exponents β , γ , and α .

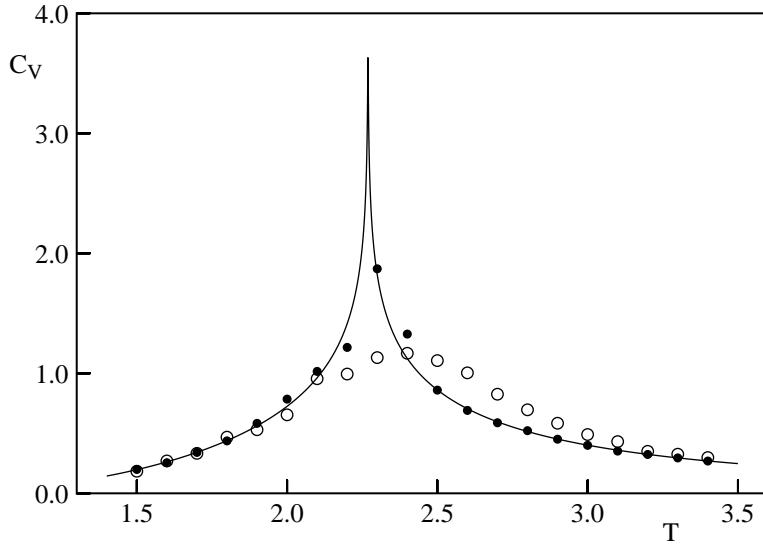


Figure 17.2: The temperature dependence of the specific heat C (per spin) of the Ising model on a $L = 8$ and $L = 16$ square lattice with periodic boundary conditions. One thousand Monte Carlo steps per spin were used for each value of the temperature. The continuous line represents the temperature dependence of C in the limit of an infinite lattice. (Note that C is infinite at $T = T_c$ for an infinite lattice.)

Problem 17.10. Finite size scaling and the critical properties of the two-dimensional Ising model

- Use the relation (17.27) together with the exact result $\nu = 1$ to estimate the value of T_c on an infinite square lattice. Because it is difficult to obtain a precise value for T_c with small lattices, we will use the exact result $kT_c/J = 2/\ln(1 + \sqrt{2}) \approx 2.269$ for the infinite lattice in the remaining parts of this problem.
- Determine the specific heat C , $|m|$, and the susceptibility χ at $T = T_c$ for $L = 2, 4, 8$, and 16 . Use as many Monte Carlo steps per spin as possible. Plot the logarithm of $|m|$ and χ versus L and use the scaling relations (17.29)–(17.31) to determine the critical exponents β and γ . Assume the exact result $\nu = 1$. Do your log-log plots of $|m|$ and χ yield reasonably straight lines? Compare your estimates for β and γ with the exact values given in Table 13.1.
- Make a log-log plot of C versus L . If your data for C is sufficiently accurate, you will find that the log-log plot of C versus L is not a straight line but shows curvature. The reason for this curvature is that α in (17.24) equals zero for the two-dimensional Ising model, and hence (17.30) needs to be interpreted as

$$C \sim C_0 \ln L. \quad (17.32)$$

Is your data for C consistent with (17.32)? The constant C_0 in (17.32) is approximately 0.4995.

So far we have performed our Ising model simulations on the square lattice. How do the critical temperature and the critical exponents depend on the symmetry and the dimension of the lattice? Based on your experience with the percolation transition in Chapter 13, you might have a good idea what the answer is.

Problem 17.11. The effects of symmetry and dimension on the critical properties of the Ising model

- The nature of the triangular lattice is discussed in Chapter 8 (see Fig. 8.5). The main difference between the triangular lattice and the square lattice is the number of nearest neighbors. Make the necessary modifications in your Ising program, for example, determine the possible transitions and the values of the transition probabilities. Compute C and χ for different values of T in the interval [1, 5]. Assume that $\nu = 1$ and use finite size scaling to estimate T_c in the limit of an infinite triangular lattice. Compare your estimate of T_c to the known value $kT_c/J = 3.641$ (to three decimal places). The simulation of Ising models on the triangular lattice is relevant to the understanding of the experimentally observed phases of materials that can be absorbed on the surface of graphite.
- No exact results are available for the Ising model in three dimensions. Write a Monte Carlo program to simulate the Ising model on the simple cubic lattice (six nearest neighbors). Compute C and χ for T in the range $3.2 \leq T \leq 5$ in steps of 0.2 for different values of L . Estimate $T_c(L)$ from the maximum of C and χ . How do these estimates of $T_c(L)$ compare? Use the values of $T_c(L)$ that exhibit a stronger L dependence and plot $T_c(L)$ versus $L^{-1/\nu}$ for different values of ν in the range 0.5 to 1 (see (17.27)). Show that the extrapolated value of $T_c(L = \infty)$ does not depend sensitively on the value of ν . Compare your estimate for $T_c(L = \infty)$ to the known value $kT_c/J = 4.5108$ (to four decimal places).
- Compute $|m|$, C , and χ at $T = T_c \approx 4.5108$ for different values of L on a simple cubic lattice. Do a finite size scaling analysis to estimate β/ν , α/ν , and γ/ν . The best known values of the critical exponents for the three-dimensional Ising model are given in Table 13.1. For comparison, published Monte Carlo results in 1976 for the finite size behavior of the Ising model on the simple cubic Ising lattice are for $L = 6$ to $L = 20$; 2000–5000 Monte Carlo steps per spin were used for calculating the averages after equilibrium had been reached.

Problem 17.12. Critical slowing down

- Consider the two-dimensional Ising model on a square lattice with $L = 16$. Compute $C_M(t)$ and $C_E(t)$ and determine the correlation times τ_M and τ_E for $T = 2.5, 2.4$, and 2.3 . Determine the correlation times as discussed in Problem 17.7d. How do these correlation times compare with one another? Show that τ increases as the critical temperature is approached, a physical effect known as *critical slowing down*.
- We can define the dynamical critical exponent z by the relation

$$\tau \sim \xi^z. \quad (17.33)$$

On a finite lattice we have the relation $\tau \sim L^z$ at $T = T_c$. Compute τ for different values of L at $T = T_c$ and make a very rough estimate of z . (The value of z for the two-dimensional Ising model with spin flip dynamics is still not definitely known, but appears to be slightly greater than 2.)

The magnitude of τ found in parts (a) and (b) depends in part on our choice of dynamics. Although we have generated a trial change by the attempted flip of one spin, it is possible that other types of trial changes, for example, the simultaneous flip of more than one spin, would be more efficient and lead to smaller correlation times and smaller values of z . A problem of much current interest is the development of more efficient algorithms near phase transitions (see Project 17.23).

17.6 Other Applications of the Ising Model

Because the applications of the Ising model are so wide ranging, we can mention only a few of the applications here. In the following, we briefly describe applications of the Ising model to first-order phase transitions, lattice gases, antiferromagnetism, and the order-disorder transition in binary alloys.

So far we have discussed the continuous phase transition in the Ising model and have found that the energy and magnetization vary continuously with the temperature, and thermodynamic derivatives such as the specific heat and the susceptibility diverge near T_c (in the limit of an infinite lattice). In Problem 17.13 we discuss a simple example of a *first-order* phase transition. Such transitions are accompanied by *discontinuous* (finite) changes in thermodynamic quantities such as the energy and the magnetization.

Problem 17.13. The two-dimensional Ising model in an external magnetic field

- a. Modify your two-dimensional Ising program so that the energy of interaction with an external magnetic field H is included. It is convenient to measure H in terms of the quantity $h = \beta H$. We wish to compute m , the mean magnetization per spin, as a function of h for $T < T_c$. Consider a square lattice with $L = 16$ and equilibrate the system at $T = 1.8$ and $h = 0$. Adopt the following procedure to obtain $m(h)$.
 - i. Use an equilibrium configuration at $h = 0$ as the initial configuration for $h_1 = \Delta h = 0.2$.
 - ii. Run the system for 100 Monte Carlo steps per spin before computing averages.
 - iii. Average m over 80 Monte Carlo steps per spin.
 - iv. Use the last configuration for h_n as the initial configuration for $h_{n+1} = h_n + \Delta h$.
 - v. Repeat steps (ii)–(iv) until $m \approx 0.95$. Plot m versus h . Do the measured values of m correspond to equilibrium averages?
- b. Decrease h by $\Delta h = -0.2$ in the same way as in part (a) until h passes through zero and until $m \approx -0.95$. Extend your plot of m versus h to negative h values. Does m remain positive for small negative h ? Do the measured values of m for negative h correspond to equilibrium averages? Draw the spin configurations for several values of h . Do you see evidence of domains?
- c. Increase h by $\Delta h = 0.2$ until the m versus h curve forms an approximately closed loop. What is the value of m at $h = 0$? This value of m is the spontaneous magnetization.

- d. A first-order phase transition is characterized by a discontinuity (for an infinite lattice) in the order parameter. In the present case the transition is characterized by the behavior of m as a function of h . What is your measured value of m for $h = 0.2$? If $m(h)$ is double valued, which value of m corresponds to the equilibrium state, an absolute minima in the free energy? Which value of m corresponds to a *metastable* state, a relative minima in the free energy? What are the equilibrium and metastable values of m for $h = -0.2$? The transition from positive m to negative m is first-order because there is a discontinuous jump in the magnetization. First-order transitions exhibit *hysteresis* and the properties of the system depend on the history of the system, for example, whether h is an increasing or decreasing function. Because of the long lifetime of metastable states near a phase transition, a system in such a state can mistakenly be interpreted as being in equilibrium. We also know that near a continuous phase transition, the relaxation to equilibrium becomes very long (see Problem 17.12), and hence a system with a continuous phase transition can behave as if it were effectively in a metastable state. For these reasons it is very difficult to distinguish the nature of a phase transition using computer simulations. This problem is discussed further in Section 17.8.
- e. Repeat the above simulation for $T = 3$, a temperature above T_c . Why do your results differ from the simulations in parts (a)–(c) done for $T < T_c$?

The Ising model also describes systems that might appear to have little in common with ferromagnetism. For example, we can interpret the Ising model as a “lattice gas,” where “down” represents a lattice site occupied by a molecule and “up” represents an empty site. Each lattice site can be occupied by at most one molecule, and the molecules interact with their nearest neighbors. The lattice gas is a crude model of the behavior of a real gas of molecules and is a simple lattice model of the gas-liquid transition and the critical point. What properties does the lattice gas have in common with a real gas? What properties of real gases does the lattice gas neglect?

If we wish to simulate a lattice gas, we have to decide whether to do the simulation at fixed density or at fixed chemical potential μ . The implementation of the latter is straightforward because the grand canonical ensemble for a lattice gas is equivalent to the canonical ensemble for Ising spins in an external magnetic field H , that is, the effect of the magnetic field is to fix the mean number of up spins. Hence, we can simulate a lattice gas in the grand canonical ensemble by doing spin flip dynamics. (The volume of the lattice is an irrelevant parameter.)

Another application of a lattice gas model is to the study of phase separation in a binary or A-B alloy. In this case spin up and spin down correspond to a site occupied by an *A* atom and *B* atom, respectively. As an example, the alloy β -brass has a low temperature ordered phase in which the two components (copper and zinc) have equal concentrations and form a cesium chloride structure. As the temperature is increased, some zinc atoms exchange positions with copper atoms, but the system is still ordered. However, above the critical temperature $T_c = 742$ K, the zinc and copper atoms become mixed and the system is disordered. This transition is an example of an *order-disorder* transition.

Because the number of *A* atoms and the number of *B* atoms is fixed, we cannot use spin flip dynamics to simulate a binary alloy. A dynamics that does conserve the number of down and up spins is known as *spin exchange dynamics*. In this dynamics a trial *interchange* of two nearest neighbor spins is made and the change in energy ΔE is calculated. The criterion for the acceptance or rejection of the trial change is the same as before.

Problem 17.14. Simulation of a lattice gas

- a. Modify your Ising program so that spin exchange dynamics rather than spin flip dynamics is implemented. For example, determine the possible values of ΔE on the square lattice, determine the possible values of the transition probability, and change the way a trial change is made. If we are interested only in the mean value of quantities such as the total energy, we can reduce the computation time by not interchanging like spins. For example, we can keep a list of bonds between occupied and empty sites and make trial moves by choosing bonds at random from this list. For small lattices such a list is unnecessary and a trial move can be generated by simply choosing a spin and one of its nearest neighbors at random.
- b. Consider a square lattice with $L = 8$ and with 32 sites initially occupied. (The number of occupied sites is a conserved variable and must be specified initially.) Determine the mean energy for T in the range $1 \leq T \leq 4$. Plot the mean energy as a function of T . Does the energy appear to vary continuously?
- c. Repeat the calculations of part (b) with 44 sites initially occupied, and plot the mean energy as a function of T . Does the energy vary continuously? Do you see any evidence of a first-order phase transition?
- d. Because the spins correspond to molecules, we can compute the single particle diffusion coefficient of the molecules. (See Problem 12.5 for a similar simulation.) Use an array to record the position of each molecule as a function of time. After equilibrium has been reached, compute $\langle R(t)^2 \rangle$, the mean square displacement per molecule. Is it necessary to “interchange” two like spins? If the atoms undergo a random walk, the self-diffusion constant D is defined as

$$D = \lim_{t \rightarrow \infty} \frac{1}{2dt} \langle R(t)^2 \rangle. \quad (17.34)$$

Estimate D for different temperatures and numbers of occupied sites.

Although you are probably familiar with ferromagnetism, for example, a magnet on a refrigerator door, nature provides more examples of antiferromagnetism. In the language of the Ising model, antiferromagnetism means that the interaction parameter J is negative and nearest neighbor spins prefer to be aligned in opposite directions. As we will see in Problem 17.15, the properties of the antiferromagnetic Ising model on a square lattice are similar to the ferromagnetic Ising model. For example, the energy and specific heat of the ferromagnetic and antiferromagnetic Ising models are identical at all temperatures in zero magnetic field, and the system exhibits a phase transition at the Néel temperature T_N . On the other hand, the total magnetization and susceptibility of the antiferromagnetic model do not exhibit any critical behavior near T_N . Instead, we can define two sublattices for the square lattice corresponding to the red and black squares of a checkerboard and introduce the “staggered magnetization” M_s equal to the difference of the magnetization on the two sublattices. We will find in Problem 17.15 that the temperature dependence of M_s and the staggered susceptibility χ_s are identical to the analogous quantities in the ferromagnetic Ising model.

Problem 17.15. Antiferromagnetic Ising model

- a. Modify `Program ising` to simulate the antiferromagnetic Ising model on the square lattice in zero magnetic field. Because J does not appear explicitly in `Program ising`, change the sign of the energy calculations in the appropriate places in the program. To compute the staggered magnetization on a square lattice, define one sublattice to be the sites (x, y) for which the product $\text{mod}(x, 2) \times \text{mod}(y, 2) = 1$; the other sublattice corresponds to the remaining sites.
- b. Choose $L = 16$ and the initial condition to be all spins up. What configuration of spins corresponds to the state of lowest energy? Compute the temperature dependence of the mean energy, specific heat, magnetization, and the susceptibility χ . Does the temperature dependence of any of these quantities show evidence of a phase transition?
- c. Compute the temperature dependence of M_s and the staggered susceptibility χ_s defined as (see (17.14))

$$\chi_s = \frac{1}{kT} [\langle M_s^2 \rangle - \langle M_s \rangle^2]. \quad (17.35)$$

Verify that the temperature dependence of M_s for the antiferromagnetic Ising model is the same as the temperature dependence of M for the Ising ferromagnet. Could you have predicted this similarity without doing the simulation?

- d. In part (b) you might have noticed that χ shows a cusp. Compute χ for different values of L at $T = T_N \approx 2.269$. Do a finite size scaling analysis and verify that χ does not diverge at $T = T_N$.
- e. Consider the behavior of the antiferromagnetic Ising model on a triangular lattice. Choose $L \geq 16$ and compute the same quantities as before. Do you see any evidence of a phase transition? Draw several configurations of the system at different temperatures. Do you see evidence of many small domains at low temperatures? Is there a unique ground state? If you cannot find a unique ground state, you share the same frustration as do the individual spins in the antiferromagnetic Ising model on the triangular lattice. We say that this model exhibits *frustration* because there is no spin configuration on the triangular lattice such that all spins are able to minimize their energy (see Fig. 17.3).

The Ising model is one of many models of magnetism. The Heisenberg, Potts, and x - y models are other examples of models of magnetic materials familiar to condensed matter scientists as well as to workers in other areas. Monte Carlo simulations of these models and others have been important in the development of our understanding of phase transitions in both magnetic and nonmagnetic materials. Some of these models are discussed in Section 17.11.

17.7 Simulation of Classical Fluids

The existence of matter as a solid, liquid and gas is well known (see Fig. 17.4). Our goal in this section is to use Monte Carlo methods to gain additional insight into the qualitative differences between these three phases.

Monte Carlo simulations of classical systems are simplified considerably by the fact that the velocity (momentum) variables are decoupled from the position variables. The total energy can be written as $E = K(\{\mathbf{v}_i\}) + U(\{\mathbf{r}_i\})$, where the kinetic energy K is a function of only the particle

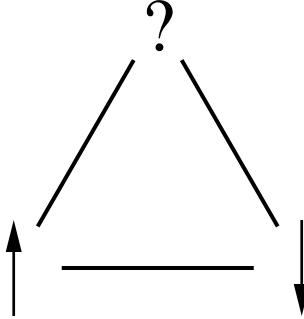


Figure 17.3: An example of frustration on a triangular lattice.

velocities $\{\mathbf{v}_i\}$, and the potential energy U is a function of only the particle positions $\{\mathbf{r}_i\}$. Because the velocity appears quadratically in the kinetic energy, the equipartition theorem implies that the contribution of the velocity coordinates to the mean energy is $\frac{1}{2}kT$ per degree of freedom. Hence, we need to sample only the positions of the molecules, that is, the “configurational” degrees of freedom. Is such a simplification possible for quantum systems?

The physically relevant quantities of a fluid include its mean energy, specific heat and equation of state. Another interesting quantity is the *radial distribution function* $g(r)$ which we introduced in Chapter 8. We will find in Problems 17.16–17.18 that $g(r)$ is a probe of the density fluctuations and hence a probe of the local order in the system. If only two-body forces are present, the mean potential energy per particle can be expressed as (see (??))

$$\frac{U}{N} = \frac{\rho}{2} \int g(r) V(r) d\mathbf{r}, \quad (17.36)$$

and the (virial) equation of state can be written as (see (8.15))

$$\frac{\beta P}{\rho} = 1 - \frac{\beta \rho}{2d} \int g(r) r \frac{dV(r)}{dr} d\mathbf{r}. \quad (17.37)$$

Hard core interactions. To separate the effects of the short range repulsive interaction from the longer range attractive interaction, we first investigate a model of *hard disks* with the interparticle interaction

$$V(r) = \begin{cases} +\infty & r < \sigma \\ 0 & r \geq \sigma. \end{cases} \quad (17.38)$$

Such an interaction has been extensively studied in one dimension (hard rods), two dimensions (hard disks), and in three dimensions (hard spheres). Hard sphere systems were the first systems studied by Metropolis and coworkers.

Because there is no attractive interaction present in (17.38), there is no transition from a gas to a liquid. Is there a phase transition between a fluid phase at low densities and a solid at high

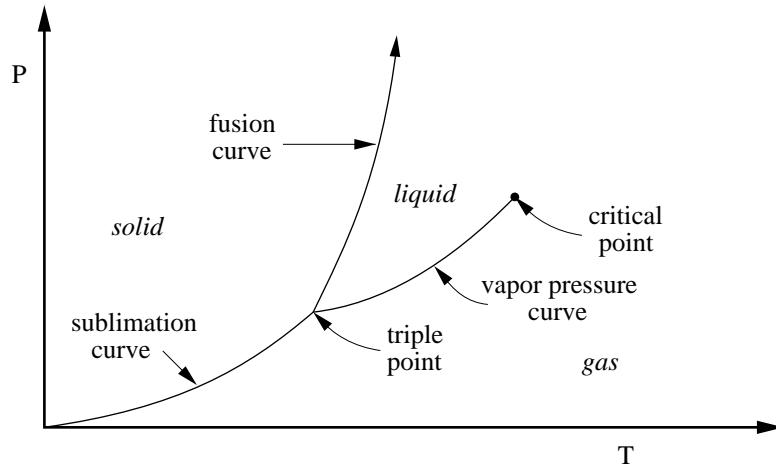


Figure 17.4: A sketch of the phase diagram for a simple material.

densities? Can a solid form in the absence of an attractive interaction? What are the physically relevant quantities for a system with an interaction of the form (17.38)? There are no thermal quantities such as the mean potential energy because this quantity is always zero. The major quantity of interest is $g(r)$ which yields information on the correlations of the particles and the equation of state. If the interaction is given by (17.38), it can be shown that (17.37) reduces to

$$\frac{\beta P}{\rho} = 1 + \frac{2\pi}{3}\rho\sigma^3 g(\sigma) \quad (d=3) \quad (17.39a)$$

$$= 1 + \frac{\pi}{2}\rho\sigma^2 g(\sigma) \quad (d=2) \quad (17.39b)$$

$$= 1 + \rho\sigma g(\sigma). \quad (d=1) \quad (17.39c)$$

We will calculate $g(r)$ for different values of r and then extrapolate our results to $r = \sigma$ (see Problem 17.16b).

Because the application of molecular dynamics and Monte Carlo methods to hard disks is similar, we discuss the latter method only briefly and do not include a program here. The idea is to choose a disk at random and move it to a trial position as implemented in the following:

```
LET itrial = int(N*rnd) + 1
LET xtrial = x(itrial) + (2*rnd - 1)*delta
LET ytrial = y(itrial) + (2*rnd - 1)*delta
```

If the new position overlaps another disk, the move is rejected and the old configuration is retained; otherwise the move is accepted. A reasonable, although not necessarily optimum, choice for the maximum displacement δ is to choose δ such that approximately one half of all trial states are accepted. We also need to fix the maximum amplitude of the move so that the moves are equally probable in all directions.

The major difficulty in implementing this algorithm is determining the overlap of two particles. If the number of particles is not too large, it is sufficient to compute the distances between the trial particle and all the other particles rather than the smaller number of particles that are in the immediate vicinity of the trial particle. For larger systems this procedure is too time consuming, and it is better to divide the system into cells and to only compute the distances between the trial particle and particles in the same and neighboring cells.

The choice of initial positions for the disks is more complicated than it might first appear. One strategy is to place each successive disk at random in the box. If a disk overlaps one that is already present, generate another pair of random numbers and attempt to place the disk again. If the desired density is low, an acceptable initial configuration can be computed fairly quickly in this way, but if the desired density is high, the probability of adding a disk will be very small (see Problem 17.17a). To reach higher densities, we might imagine beginning with the desired number of particles in a low density configuration and moving the boundaries of the central cell inward until a boundary just touches one of the disks. Then the disks are moved a number of Monte Carlo steps and the boundaries are moved inward again. This procedure also becomes more difficult as the density increases. The most efficient procedure is to start the disks on a lattice at the highest density of interest such that no overlap of disks occurs.

We first consider a one-dimensional system of hard rods for which the equation of state and $g(r)$ can be calculated exactly. The equation of state is given by

$$\frac{P}{NkT} = \frac{1}{L - N\sigma}. \quad (17.40)$$

Because hard rods cannot pass through one another, the excluded volume is $N\sigma$ and the available volume is $L - N\sigma$. Note that this form of the equation of state is the same as the van der Waals equation of state (cf. Reif) with the contribution from the attractive part of the interaction equal to zero.

Problem 17.16. Monte Carlo simulation of hard rods

- Write a program to do a Monte Carlo simulation of a system of hard rods. Adopt the periodic boundary condition and refer to [Program hd](#) in Chapter 8 for the basic structure of the program. The major difference is the nature of the “moves.” Measure all lengths in terms of the hard rod diameter σ . Choose $L = 12$ and $N = 10$. How does the number density $\rho = N/L$ compare to the maximum possible density? Choose the initial positions to be on a one-dimensional grid and let the maximum displacement $\delta = 0.1$. Approximately how many Monte Carlo steps per particle are necessary to reach equilibrium? What is the equilibrium acceptance probability? Compute the pair correlation function $g(x)$.
- Plot $g(x)$ as a function of the distance x . Why does $g(x) = 0$ for $x < 1$? Why are the values of $g(x)$ for $x > L/2$ not meaningful? What is the physical interpretation of the peaks in $g(x)$? Note that the results for $g(x)$ are for $x > 1$ because none of the hard disks are in contact in the configurations generated by the Monte Carlo algorithm. (Recall that x is measured in units of σ .) Because the mean pressure can be determined from $g(x)$ at $x = 1^+$ (see (17.39)), we need to determine $g(x)$ at contact by extrapolating your results for $g(x)$ to $x = 1$. An easy way to do so is to fit the three points of $g(x)$ closest to $x = 1$ to a parabola. Use your result for $g(x = 1^+)$ to determine the mean pressure.

- c. Compute $g(x)$ at several lower densities by using an equilibrium configuration from a previous run and increasing L . How do the size and the location of the peaks in $g(x)$ change?

Problem 17.17. Monte Carlo simulation of hard disks

- a. What is the maximum packing density of hard disks, that is, how many disks can be packed together in a cell of area A ? The maximum packing density can be found by placing the disks on a triangular lattice with the nearest neighbor distance equal to the disk diameter. Write a simple program that adds disks at random into a rectangular box of area $A = L_x \times L_y$ with the constraint that no two disks overlap. If a disk overlaps a disk already present, generate another pair of random numbers and try to place the disk again. If the density is low, the probability of adding a disk is high, but if the desired density is high most of the disks will be rejected. For simplicity, do not worry about periodic boundary conditions and accept a disk if its center lies within the box. Choose $L_x = 6$ and $L_y = \sqrt{3}L_x/2$ and determine the maximum density $\rho = N/A$ that you can attain in a reasonable amount of CPU time. How does this density compare to the maximum packing density? What is the qualitative nature of the density dependence of the acceptance probability?
- b. Adapt your Monte Carlo program for hard rods to a system of hard disks. Begin at a density ρ slightly lower than the maximum packing density ρ_0 . Choose $N = 16$ with $L_x = 4.41$ and $L_y = \sqrt{3}L_x/2$. Compare the density $\rho = N/(L_x L_y)$ to the maximum packing density. Choose the initial positions of the particles to be on a triangular lattice. A reasonable first choice for the maximum displacement δ is $\delta = 0.1$. Compute $g(r)$ for $\rho/\rho_0 = 0.95, 0.92, 0.88, 0.85, 0.80, 0.70, 0.60$, and 0.30 . Keep the ratio of L_x/L_y fixed and save a configuration from the previous run to be the initial configuration of the new run at lower ρ . Allow at least 400 Monte Carlo steps per particle for the system to equilibrate and average $g(r)$ for $mcs \geq 400$.
- c. What is the qualitative behavior of $g(r)$ at high and low densities? For example, describe the number and height of the peaks of $g(r)$. If the system is crystalline, then $g(\mathbf{r})$ is not spherically symmetric. How would you compute $g(\mathbf{r})$ in this case and what would you expect to see?
- d. Use your results for $g(r = 1^+)$ to compute the mean pressure P as a function of ρ (see (17.39b)). Plot the ratio PV/NkT as a function of ρ , where the “volume” V is the area of the system. How does the temperature T enter into the Monte Carlo simulation? Is the ratio PV/NkT an increasing or decreasing function of ρ ? At low densities we might expect the system to act like an ideal gas with the volume replaced by $(V - N\sigma)$. Compare your low density results with this prediction.
- e. Take “snapshots” of the disks at intervals of ten to twenty Monte Carlo steps per particle. Do you see any evidence of the solid becoming a fluid at lower densities?
- f. Compute an “effective diffusion coefficient” D by determining the mean square displacement $\langle R^2(t) \rangle$ of the particles after equilibrium is reached. Use the relation (17.34) and identify the time t with the number of Monte Carlo steps per particle. Estimate D for the densities considered in part (a), and plot the product ρD as a function of ρ . What is the dependence of D on ρ for a dilute gas? Try to identify a range of ρ where D drops abruptly. Do you observe any evidence of a phase transition?

- g. The magnitude of the maximum displacement parameter δ is arbitrary. If δ is large and the density is high, then a high proportion of the trial moves will be rejected. On the other hand, if δ is too small, the acceptance probability will be close to unity, but the successive configurations will be strongly correlated. Hence if δ is too large or is too small, our simulation is inefficient. In practice, δ is usually chosen so that approximately half of the moves are accepted. A better criterion might be to choose δ so that the mean square displacement is a maximum for a fixed time interval. The idea is that the mean square displacement is a measure of the exploration of phase space. Fix the density and determine the value of δ that maximizes $\langle R^2(t) \rangle$. What is the corresponding acceptance probability? If this probability is much less than 50%, how can you decide which criterion for δ is more appropriate?

Continuous potentials. Our simulations of hard disks have led us to conclude that there is a phase transition from a fluid at low densities to a solid at higher densities. This conclusion is consistent with molecular dynamics and Monte Carlo studies of larger systems. Although the existence of a fluid-solid transition for hard sphere and hard disk systems is well accepted, the relatively small numbers of particles used in any simulation should remind us that results of this type cannot be taken as evidence independently of any theoretical justification.

The existence of a fluid-solid transition for hard spheres implies that the transition is primarily determined by the repulsive part of the potential. We now consider a system with both a repulsive and an attractive contribution. Our primary goal will be to determine the influence of the attractive part of the potential on the structure of a liquid.

We adopt as our model interaction the Lennard-Jones potential:

$$U(r) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right]. \quad (17.41)$$

The nature of the Lennard-Jones potential and the appropriate choice of units for simulations was discussed in Chapter 8 (see Table 8.1). We consider in Problem 17.18 the application of the Metropolis algorithm to a system of N particles in a rectangular cell of fixed “volume” V interacting via the Lennard-Jones potential. Because the simulation is at fixed T, V , and N , the simulation samples configurations of the system according to the Boltzmann distribution (17.1).

Problem 17.18. Monte Carlo simulation of a Lennard-Jones system

- a. The properties of a two-dimensional Lennard-Jones system with the potential energy of interaction (17.41) have been studied by many workers under a variety of conditions. Write a program to compute the total energy of a system of N particles on a triangular lattice of area $L_x \times L_y$ with periodic boundary conditions. Choose $N = 16$, $L_x = 4.6$, and $L_y = \sqrt{3}L_x/2$. Why does this energy correspond to the energy at temperature $T = 0$? Does the energy per particle change if you consider bigger systems at the same density?
- b. Write a program to compute the mean energy, pressure, and radial distribution function using the Metropolis algorithm. One way of computing the change in the potential energy of the system due to a trial move of one of the particles is to use an array, `pe`, for the potential energy of interaction of each particle. For simplicity, compute the potential energy of particle `i` by considering its interaction with the other $N - 1$ particles. The total potential energy of the system is the sum of the array elements `pe(i)` over all N particles divided by two to account

- for double counting. For simplicity, accumulate data after each Monte Carlo step per particle. A reasonable choice for the bin width dr for the calculation of $g(r)$ is $\text{dr} = 0.1$.
- c. Choose the same values of N , L_x , and L_y as in part (a), but give each particle an initial random displacement from its triangular lattice site of magnitude 0.2. Do the Monte Carlo simulation at a very low temperature such as $T = 0.1$. Choose the maximum trial displacement $\delta = 0.15$ and consider $\text{mcs} \geq 400$. Does the system retain its hexagonal symmetry? Does the value of δ affect your results?
 - d. Use the same initial conditions as in part (a), but take $T = 0.5$. Choose $\delta = 0.15$ and run for a number of Monte Carlo steps per particle that is sufficient to yield a reasonable result for the mean energy. Do a similar simulation at $T = 1$ and $T = 2$. What is the best choice of the initial configuration in each case? The harmonic theory of solids predicts that the total energy of a system is due to a $T = 0$ contribution plus a term due to the harmonic oscillation of the atoms. The contribution of the latter part should be proportional to the temperature. Compare your results for $E(T) - E(0)$ with this prediction. Use the values of σ and ϵ given in Table 8.1 to determine the temperature and energy in SI units for your simulations of solid argon.
 - e. Describe the qualitative nature of $g(r)$ for a Lennard-Jones solid and compare it with your hard disk results for the same density.
 - f. Decrease the density by multiplying L_x , L_y , and all the particle coordinates by 1.07. What is the new value of ρ ? Estimate the number of Monte Carlo steps per particle needed to compute E and P for $T = 0.5$ to approximately 10% accuracy. Is the total energy positive or negative? How do E and P compare to their ideal gas values? Follow the method discussed in Problem 17.17 and compute an effective diffusion constant. Is the system a liquid or a solid? Plot $g(r)$ versus r and compare $g(r)$ to your results for hard disks at the same density. What is the qualitative behavior of $g(r)$? What is the interpretation of the peaks in $g(r)$ in terms of the structure of the liquid? If time permits, consider a larger system at the same density and temperature and compute $g(r)$ for larger r .
 - g. Consider the same density system as in part (f) at $T = 0.6$ and $T = 1$. Look at some typical configurations of the particles. Use your results for $E(T)$, $P(T)$, $g(r)$ and the other data you have collected, and discuss whether the system is a gas, liquid, or solid at these temperatures. What criteria can you use to distinguish a gas from a liquid? If time permits, repeat these calculations for $\rho = 0.7$.
 - h. Compute E , P , and $g(r)$ for $N = 16$, $L_x = L_y = 10$, and $T = 3$. These conditions correspond to a dilute gas. How do your results for P compare with the ideal gas result? How does $g(r)$ compare with the results you obtained for the liquid?

17.8 *Optimized Monte Carlo Data Analysis

As we have seen, the important physics near a phase transition occurs on long length scales. For this reason, we might expect that simulations, which for practical reasons are restricted to relatively small systems, might not be useful for simulations near a phase transition. Nevertheless, we have found that methods such as finite size scaling can yield information about how systems behave

in the thermodynamic limit. We next explore some additional Monte Carlo techniques that are useful near a phase transition.

The Metropolis algorithm yields mean values of various thermodynamic quantities, for example, the energy, at particular values of the temperature T . Near a phase transition many thermodynamic quantities change rapidly, and we need to determine these quantities at many closely spaced values of T . If we were to use standard Monte Carlo methods, we would have to do many simulations to cover the desired range of values of T . To overcome this problem, we introduce the use of *histograms* which allow us to extract more information from a single Monte Carlo simulation. The idea is to use our knowledge of the equilibrium probability distribution at one value of T (and other external parameters) to estimate the desired thermodynamic averages at neighboring values of the external parameters.

The first step of the single histogram method for the Ising model is to simulate the system at an inverse temperature β_0 which is near the values of β of interest and measure the energy of the system after every Monte Carlo step per spin (or other fixed interval). The measured probability that the system has energy E can be expressed as

$$P(E, \beta_0) = \frac{H_0(E)}{\sum_E H_0(E)}. \quad (17.42)$$

The histogram $H_0(E)$ is the number of configurations with energy E , and the denominator is the total number of measurements of E (for example, the number of Monte Carlo steps per spin). Because the probability of a given configuration is given by the Boltzmann distribution, we have

$$P(E, \beta) = \frac{W(E) e^{-\beta E}}{\sum_E W(E) e^{-\beta E}}, \quad (17.43)$$

where $W(E)$ is the number of microstates with energy E . (This quantity is frequently called the *density of states*, and is more generally defined as the number of states per unit energy interval.) If we compare (17.42) and (17.43) and note that $W(E)$ is independent of T , we can write

$$W(E) = a_0 H_0(E) e^{\beta_0 E}, \quad (17.44)$$

where a_0 is a proportionality constant that depends on β_0 . If we eliminate $W(E)$ from (17.43) by using (17.44), we obtain the desired relation

$$P(E, \beta) = \frac{H_0(E) e^{-(\beta - \beta_0)E}}{\sum_E H_0(E) e^{-(\beta - \beta_0)E}}. \quad (17.45)$$

Note that we have expressed the probability at inverse temperature β in terms of $H_0(E)$, the histogram at inverse temperature β_0 .

Because β is a continuous variable, we can estimate the β dependence of the mean value of any function A that depends on E , for example, the mean energy and the specific heat. We write the mean of $A(E)$ as

$$\langle A(\beta) \rangle = \sum_E A(E) P(E, \beta). \quad (17.46)$$

If the quantity A depends on another quantity M , for example, the magnetization, then we can generalize (17.46) to

$$\langle A(\beta) \rangle = \sum_{E,M} A(E,M) P(E,M,\beta) \quad (17.47)$$

$$= \frac{\sum_{E,M} A(E,M) H_0(E,M) e^{-(\beta-\beta_0)E}}{\sum_{E,M} H_0(E,M) e^{-(\beta-\beta_0)E}}. \quad (17.48)$$

The histogram method is useful only when the configurations relevant to the range of temperatures of interest occur with reasonable probability during the simulation at temperature T_0 . For example, if we simulate an Ising model at low temperatures at which only ordered configurations occur (most spins aligned in the same direction), we cannot use the histogram method to obtain meaningful thermodynamic averages at high temperatures at which most configurations are disordered.

Problem 17.19. Application of the histogram method

- a. Consider a 4×4 Ising lattice in zero magnetic field and compute the mean energy per spin, the mean magnetization per spin, the specific heat, and the susceptibility per spin for $T = 1$ to $T = 3$ in steps of $\Delta T = 0.05$. Average over at least 5000 Monte Carlo steps per spin after equilibrium has been reached for each value of T .
- b. What are the minimum and maximum values of the total energy E and magnetization M that might be observed in a simulation of a Ising model on a 4×4 lattice? Use these values to set the size of the two-dimensional array needed to accumulate data for the histogram $H(E, M)$. It is suggested that you modify **Program ising** and save $H(E, M)$ in a file. Accumulate data for $H(E, M)$ at $T = 2.27$, a value of T close to T_c , for at least 5000 Monte Carlo steps per spin after equilibration. Write a separate program to read the histogram file and compute the same thermodynamic quantities as in part (a) using (17.48). Compare your computed results with the data obtained by simulating the system directly, that is, without using the histogram method, at the same temperatures. At what temperatures does the histogram method break down?
- c. Repeat parts (b) and (c) for a simulation centered about $T = 1.5$ and $T = 2.5$.
- d. Repeat parts (b) and (c) for an 8×8 and a 16×16 lattice at $T = 2.27$.

The histogram method can be used to do a more sophisticated finite size scaling analysis to determine the nature of a transition. Suppose that we perform a Monte Carlo simulation and observe a peak in the specific heat as a function of the temperature. What can this observation tell us about a possible phase transition? In general, we can conclude very little without doing a careful analysis of the behavior of the system at different sizes. For example, a discontinuity in the energy in an infinite system might be manifested in small systems by a peak in the specific heat. However, a phase transition in the infinite system in which the energy is continuous, but its derivative diverges at the transition, might manifest itself in the same way in a small system. Another difficulty is that the peak in the specific heat of a small system occurs at a temperature that differs from the transition temperature in the infinite system (see Project 17.25). Finally, there might be no transition at all, and the peak might simply represent a broad crossover from high to low temperature behavior (see Project 17.26).

We now discuss a method due to Lee and Kosterlitz that uses the histogram data to determine the nature of a phase transition (if it exists). To understand this method, we need to introduce the (Helmholtz) free energy F of a system. The statistical mechanics definition of F is

$$F = -kT \ln Z. \quad (17.49)$$

At low T , the factor $e^{-\beta E}$ in the partition function Z determines the dominant contributions to Z , even though there are relatively few such configurations. At high T , the factor $e^{-\beta E}$ is not very large, but the number of disordered configurations with high E is large, and hence high energy configurations dominate the contribution to Z . These considerations suggest that it is useful to define a restricted free energy $F(E)$ that includes only configurations at a particular energy E . We define

$$F(E) = -kT \ln W(E) e^{-\beta E}. \quad (17.50)$$

For systems with a first-order phase transition, a plot of $F(E)$ versus E will show two local minima corresponding to configurations that are characteristic of the high and low temperature phases. At low T the minimum at the lower energy will be the absolute minimum, and at high T the higher energy minimum will be the absolute minimum of F . At the transition, the two minima will have the same value of $F(E)$. For systems with no transition in the thermodynamic limit, there will only be one minimum for all T .

How will $F(E)$ behave for the relatively small lattices considered in simulations? In systems with first-order transitions, the distinction between low and high temperature phases will become more pronounced as the system size is increased. If the transition is continuous, there are domains at all sizes, and we expect that the behavior of $F(E)$ will not change significantly as the system size increases. If there is no transition, there might be a spurious double minima for small systems, but this spurious behavior should disappear for larger systems. Lee and Kosterlitz proposed the following method for categorizing phase transitions.

1. Do a simulation at a temperature close to the suspected transition temperature, and compute $H(E)$. Usually, the temperature at which the peak in the specific heat occurs is chosen as the simulation temperature.
2. Use the histogram method to compute $-\ln H_0(E) + (\beta - \beta_0)E \propto F(E)$ at neighboring values of T . If there are two minima in $F(E)$, vary β until the values of $F(E)$ at the two minima are equal. This temperature is an estimate of the possible transition temperature T_c .
3. Measure the difference ΔF at T_c between $F(E)$ at the minima and $F(E)$ at the maximum between the two minima.
4. Repeat steps (1)–(3) for larger systems. If ΔF increases with size, the transition is first-order. If ΔF remains the same, the transition is continuous. If ΔF decreases with size, there is no thermodynamic transition.

The above procedure is applicable when the phase transition occurs by varying the temperature. Transitions also can occur by varying the pressure or the magnetic field. These *field-driven transitions* can be tested by a similar method. For example, consider the Ising model in a magnetic

field at low temperatures below T_c . As we vary the magnetic field from positive to negative, there is a transition from a phase with magnetization $M > 0$ to a phase with $M < 0$. Is this transition first-order or continuous? To answer this question, we can use the Lee-Kosterlitz method with a histogram $H(E, M)$ generated at zero magnetic field, and calculate $F(M)$ instead of $F(E)$. The quantity $F(M)$ is proportional to $-\ln \sum_E H(E, M) e^{-(\beta - \beta_0)E}$. Because the states with positive and negative magnetization are equally likely to occur for zero magnetic field, we should see a double minima structure for $F(M)$ with equal minima. As we increase the size of the system, ΔF should increase for a first-order transition and remain the same for a continuous transition.

Problem 17.20. Characterization of a phase transition

- Use your modified version of [Program ising](#) from Problem 17.19 to determine $H(E, M)$. Read the $H(E, M)$ data from a file, and compute and plot $F(E)$ for the range of temperatures of interest. First generate data at $T = 2.27$ and use the Lee-Kosterlitz method to verify that the Ising model in two dimensions has a continuous phase transition in zero magnetic field. Consider lattices of sizes $L = 4, 8$, and 16 .
- Perform a Lee-Kosterlitz analysis of the Ising model at $T = 2$ and zero magnetic field by plotting $F(M)$. Determine if the transition from $M > 0$ to $M < 0$ is first-order or continuous. This transition is called field-driven because the transition occurs if we change the magnetic field. Make sure your simulations sample configurations with both positive and negative magnetization by using small values of L such as $L = 4, 6$ and 8 .
- Repeat part (b) at $T = 2.5$ and determine if there is a field-driven transition at $T = 2.5$.

Problem 17.21. The Potts Model

- In the q -state Potts model, the total energy or Hamiltonian of the lattice is given by

$$H = -J \sum_{i,j=\text{nn}(i)} \delta_{s_i, s_j}, \quad (17.51)$$

where s_i at site i can have the values $1, 2, \dots, q$; the Kronecker delta function $\delta_{a,b}$ equals unity if $a = b$ and is zero otherwise. As before, we will measure the temperature in energy units. Convince yourself that the $q = 2$ Potts model is equivalent to the Ising model (except for a trivial difference in the energy minimum). One of the many applications of the Potts model is to helium absorbed on the surface of graphite. The graphite-helium interaction gives rise to preferred adsorption sites directly above the centers of the honeycomb graphite surface. As discussed by Plischke and Bergersen, the helium atoms can be described by a three-state Potts model.

- The transition in the Potts model is continuous for small q and first-order for larger q . Write a Monte Carlo program to simulate the Potts model for a given value of q and store the histogram $H(E)$. Test your program by comparing the output for $q = 2$ with your Ising model program.
- Use the Lee-Kosterlitz method to analyze the nature of the phase transition in the Potts model for $q = 3, 4, 5, 6$, and 10 . First find the location of the specific heat maximum, and then collect data for $H(E)$ at the specific heat maximum. Lattice sizes of order $L \geq 50$ are required to obtain convincing results for some values of q .

17.9 *Other Ensembles

So far, we have considered the microcanonical ensemble (fixed N , V , and E) and the canonical ensemble (fixed N , V , and T). Monte Carlo methods are very flexible and can be adapted to the calculation of averages in any ensemble. Two other ensembles of particular importance are the constant pressure and the grand canonical ensembles. The main difference in the Monte Carlo method is that there are additional “moves” corresponding to changing the volume or changing the number of particles. The constant pressure ensemble is particularly important for studying first-order phase transitions because the phase transition occurs at a fixed pressure, unlike a constant volume simulation where the system passes through a two phase coexistence region before changing phase completely as the volume is changed.

In the NPT ensemble, the probability of a microstate occurring is proportional to $e^{-\beta(E+PV)}$. For a classical system, the mean value of a physical quantity A that depends on the coordinates of the particles can be expressed as

$$\langle A \rangle_{NPT} = \frac{\int_0^\infty dV e^{-\beta PV} \int d\mathbf{r}_1 d\mathbf{r}_2 \dots d\mathbf{r}_N A(\{\mathbf{r}\}) e^{-\beta U(\{\mathbf{r}\})}}{\int_0^\infty dV e^{-\beta PV} \int d\mathbf{r}_1 d\mathbf{r}_2 \dots d\mathbf{r}_N e^{-\beta U(\{\mathbf{r}\})}}. \quad (17.52)$$

The potential energy $U(\{\mathbf{r}\})$ depends on the set of particle coordinates ($\{\mathbf{r}\}$). To simulate the NPT ensemble, we need to sample the coordinates $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_N$ of the particles and the volume V of the system. For simplicity, we assume that the central cell is a square or a cube so that $V = L^d$. It is convenient to use the set of scaled coordinates $\{\mathbf{s}\}$, where \mathbf{s}_i is defined as

$$\mathbf{s}_i = \frac{\mathbf{r}_i}{L}. \quad (17.53)$$

If we substitute (17.53) into (17.52), we can write $\langle A \rangle_{NPT}$ as

$$\langle A \rangle_{NPT} = \frac{\int_0^\infty dV e^{-\beta PV} V^N \int d\mathbf{s}_1 d\mathbf{s}_2 \dots d\mathbf{s}_N A(\{\mathbf{s}\}) e^{-\beta U(\{\mathbf{s}\})}}{\int_0^\infty dV e^{-\beta PV} V^N \int d\mathbf{s}_1 d\mathbf{s}_2 \dots d\mathbf{s}_N e^{-\beta U(\{\mathbf{s}\})}}, \quad (17.54)$$

where the integral over $\{\mathbf{s}\}$ is over the unit square (cube). The factor of V^N arises from the change of variables $\mathbf{r} \rightarrow \mathbf{s}$. If we let $V^N = e^{\ln V^N} = e^{N \ln V}$, we see that the quantity that is analogous to the Boltzmann factor can be written as

$$e^{-W} = e^{-\beta PV - \beta U(\{\mathbf{s}\}) + N \ln V}. \quad (17.55)$$

Because the pressure is fixed, a trial configuration is generated from the current configuration by either randomly displacing a particle and/or making a random change in the volume, for example, $V \rightarrow V + \delta V_{\max}(2r - 1)$, where r is a uniform random number in the unit interval. The trial configuration is accepted if the change $\Delta W \leq 0$ and with probability $e^{-\Delta W}$ if $\Delta W > 0$. It is not necessary or efficient to change the volume after every Monte Carlo step per particle.

In the grand canonical or μVT ensemble, the chemical potential μ is fixed and the number of particles fluctuates. The average of any function of the particle coordinates can be written as (in three dimensions)

$$\langle A \rangle_{\mu VT} = \frac{\sum_{N=0}^{\infty} (1/N!) \lambda^{-3N} e^{\beta \mu N} \int d\mathbf{r}_1 d\mathbf{r}_2 \dots d\mathbf{r}_N A(\{\mathbf{r}\}) e^{-\beta U_N(\{\mathbf{r}\})}}{\sum_{N=0}^{\infty} (1/N!) \lambda^{-3N} e^{\beta \mu N} \int d\mathbf{r}_1 d\mathbf{r}_2 \dots d\mathbf{r}_N e^{-\beta U_N(\{\mathbf{r}\})}}, \quad (17.56)$$

where $\lambda = (h^2/2\pi mkT)^{1/2}$. We have made the N -dependence of the potential energy U explicit. If we write $1/N! = e^{-\ln N!}$ and $\lambda^{-3N} = e^{-N \ln \lambda^3}$, we can write the quantity that is analogous to the Boltzmann factor as

$$e^{-W} = e^{\beta\mu N - N \ln \lambda^3 - \ln N! + N \ln V - \beta U_N}. \quad (17.57)$$

If we write the chemical potential as

$$\mu = \mu^* + kT \ln(\lambda^3/V), \quad (17.58)$$

then W can be expressed as

$$e^{-W} = e^{-\beta\mu^* N - \ln N! - \beta U_N}. \quad (17.59)$$

The parameters are μ^* , V , and T . There are two possible ways of obtaining a trial configuration. The first involves the displacement of a selected particle; such a move is accepted or rejected according to the usual criteria, that is, by the change in the potential energy U_N . In the second possible way, we choose with equal probability whether to attempt to add a particle at a randomly chosen position in the central cell or to remove a particle that is already present. In either case, the trial configuration is accepted if W in (17.59) is increased. If W is decreased, the change is accepted with a probability equal to

$$\frac{1}{N+1} e^{\beta(\mu^* - (U_{N+1} - U_N))} \text{(insertion)} \quad (17.60a)$$

or

$$N e^{-\beta(\mu^* + (U_{N+1} - U_N))} \text{(removal)} \quad (17.60b)$$

In this approach μ^* is an input parameter and μ is not determined until the end of the calculation when $\langle N \rangle_{\mu VT}$ is obtained.

17.10 More Applications

You probably do not need to be convinced that Monte Carlo methods are powerful, flexible, and applicable to a wide variety of systems. Extensions to the Monte Carlo methods that we have not discussed include multiparticle moves, biased moves where particles tend to move in the direction of the force on them, manipulation of bits for Ising-like models, the *n-fold way* algorithm for Ising-like models at low temperature, use of special processors for specific systems, and the use of parallel processing to update different parts of a large system simultaneously. We also have not described the simulation of systems with long-range potentials such as Coulombic systems and dipole-dipole interactions. For these potentials, it is necessary to include the interactions of the particles in the center cell with the infinite set of periodic images.

We conclude this chapter with a discussion of Monte Carlo methods in a context that might seem to have little in common with the types of problems we have discussed. This context is called *multivariate* or *combinatorial optimization*, a fancy way of saying, “How do you find the minimum of a function that depends on many parameters?” Problems of this type arise in many areas of scheduling and design. We explain the nature of this type of problem by an example known as the *traveling salesperson problem*, a.k.a., the *traveling salesman problem*.

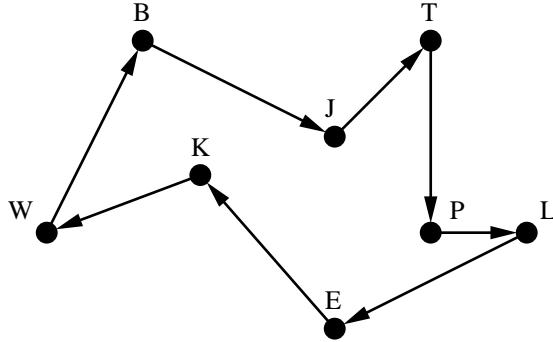


Figure 17.5: What is the optimum route for this random arrangement of $N = 8$ cities? The route begins and ends at city W. A possible route is shown.

Suppose that a salesperson wishes to visit N cities and follow a route such that no city is visited more than once and the end of the trip coincides with the beginning. Given these constraints, the traveling salesperson problem is to find the optimum route such that the total distance traveled is a minimum. An example of $N = 8$ cities and a possible route is shown in Fig. 17.5. All exact methods for determining the optimal route require a computing time that increases as e^N , and in practice, an exact solution can be found only for a small number of cities. The traveling salesperson problem belongs to a large class of problems known as NP-complete. (The NP refers to non-polynomial, that is, such problems cannot be done in a time proportional to some finite polynomial in N .) What is a reasonable estimate for the maximum number of cities that you can consider without the use of a computer?

To understand the nature of the different approaches to the traveling salesperson problem, consider the plot in Fig. 17.6 of the “energy” function $E(a)$. We can associate $E(a)$ with the length of the route and interpret a as a parameter that represents the order in which the cities are visited. If $E(a)$ has several local minima, what is a good strategy for finding the global (absolute) minimum of $E(a)$? One way is to vary a systematically and find the value of E everywhere. This exact enumeration method corresponds to determining the length of each possible route, clearly an impossible task if the number of cities is too large. Another way is to use a *heuristic method*, that is, an approximate method for finding a route that is close to the absolute minimum. One strategy is to choose a value of a , generate a small random change δa , and accept this change if $E(a + \delta a)$ is less than or equal to $E(a)$. This iterative improvement strategy corresponds to a search for steps that lead downhill. Because this search usually becomes stuck in a local and not a global minimum, it is useful to begin from several initial choices of a and to keep the best result. What would be the application of this type of strategy to the salesperson problem?

Let us consider a seemingly unrelated problem. Suppose we wish to make a perfect single crystal. You probably know that we should first melt the material, and then lower the temperature very slowly to the desired low temperature. If we lower the temperature too quickly (a rapid “quench”), the resulting crystal would have many defects or not become a crystal at all. The gradual lowering of the temperature is known as *annealing*.

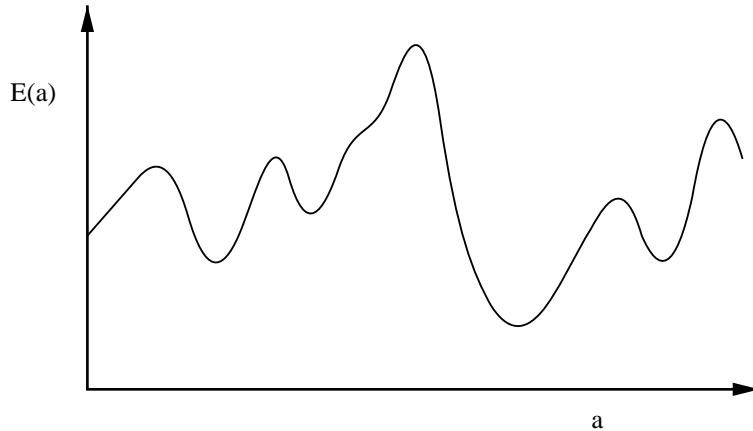


Figure 17.6: Plot of the function $E(a)$ as a function of the parameter a .

The method of annealing can be used to estimate the minimum of $E(a)$. We choose a value of a , generate a small random change δa , and calculate $E(a + \delta a)$. If $E(a + \delta a)$ is less than or equal to $E(a)$, we accept the change. However, if $\Delta E = E(a + \delta a) - E(a) > 0$, we accept the change with a probability $P = e^{-\Delta E/T}$, where T is an effective temperature. This procedure is the familiar Metropolis algorithm with the temperature playing the role of a control parameter. The *simulated annealing* process consists of first “melting” the system, that is, choosing T for which most moves are accepted, and then gradually lowering the temperature. At each temperature, the simulation should last long enough for the system to reach a steady state. The annealing schedule, that is, the rate of temperature decrease, determines the quality of the solution.

The moral of the simulated annealing method is that sometimes it is necessary to climb a hill to reach a valley. The first application of the method of simulated annealing was to the optimal design of computers. In Problem 17.22 we apply this method to the traveling salesperson problem. Perhaps you can think of other applications.

Problem 17.22. Simulated annealing and the traveling salesperson problem

- a. Generate a random arrangement of $N = 8$ cities in a square of linear dimension $L = \sqrt{N}$ and calculate the optimum route by hand. Then write a Monte Carlo program and apply the method of simulated annealing to this problem. For example, use two arrays to store the coordinates of each city and an array to store the distances between them. The state of the system, that is, the route represented by a sequence of cities, can be stored in another array. The length of this route is associated with the energy of an imaginary thermal system. A reasonable choice for the initial temperature is one that is the same order as the initial energy. One way to generate random rearrangements of the route is to choose two cities at random and to interchange the order of visit. Choose this method or one that you devise and find a reasonable annealing schedule. Compare your annealing results to exact results whenever possible. Extend your results to larger N , for example, $N = 12, 24$, and 48 . For a given annealing schedule, determine the probability of finding a route of a given length. More suggestions can be found in the

references.

- b. The microcanonical Monte Carlo algorithm (demon) discussed in Chapter 16 also can be used to do simulated annealing. The advantages of the demon algorithm are that it is deterministic and allows large temperature fluctuations. One way to implement the analog of simulated annealing is to impose a maximum value on the energy of the demon, $E_{d,\max}$ and then gradually lower the value of $E_{d,\max}$. Guo et al. choose the initial value of $E_{d,\max}$ to equal $\sqrt{N}/4$. Their results are comparable to the canonical simulated annealing method, but require approximately half the CPU time. Apply this method to the same routes that you considered in part (a) and compare your results.

17.11 Projects

Many of the original applications of Monte Carlo methods were done for systems of approximately one hundred particles and lattices of order 32^2 spins. Most of these applications can be done with much better statistics and with larger system sizes. In the following, we discuss some recent developments, but this discussion is not complete and we have omitted other important topics such as Brownian dynamics and umbrella sampling. More ideas for projects can be found in the references.

Project 17.23. Overcoming critical slowing down The usual limiting factor of most simulations either is the lack of computer memory or, more likely, the speed of the computer. One way to overcome the latter problem is to use a faster computer. However, near a phase transition, the most important limiting factor on even the fastest available computers is the existence of “critical slowing down” (see Problem 17.12). In this project we discuss the nature of critical slowing down and ways of overcoming it in the context of the Ising model.

The existence of critical slowing down is related to the fact that the size of the correlated regions of spins becomes very large near the Ising critical point. The large size of the correlated regions and the corresponding divergent behavior of the correlation length ξ near T_c implies that the time τ required for a region to lose its coherence becomes very long if a *local* dynamics is used. Near T_c , we have $\tau \sim \xi^z$, which defines the dynamical critical exponent z . At $T = T_c$, we have $\tau \sim L^z$ for L sufficiently large. For the single spin flip (Metropolis) algorithm, $z \approx 2$, and τ becomes very large for $L \gg 1$. On a serial computer, the CPU time needed to obtain n configurations increases as L^2 , the time needed to visit L^2 spins. This factor of L^2 is not a problem because a larger system contains proportionally more information. However, the time needed to obtain n approximately *independent* configurations is order $\tau L^2 \sim L^{2+z} \approx L^4$ for the Metropolis algorithm. We see that an increase of L by a factor of 10 requires 10^4 more computing time. Hence, the existence of critical slowing down limits the maximum value of L that can be considered.

If we are interested only in the static properties of the Ising model, the choice of dynamics is irrelevant as long as the transition probability satisfies the detailed balance condition (17.10). As we mentioned above, the Metropolis algorithm becomes inefficient near T_c because only single spins are flipped. Hence, it is reasonable to look for a *global* algorithm for which groups or *clusters* of spins are flipped simultaneously. We already are familiar with cluster properties in the context of percolation (see Chapter 13). A naive definition of a cluster of spins might be a domain or group of parallel nearest neighbor spins. We can make this definition more explicit by introducing a bond

between any two nearest neighbor spins that point in the same direction. The introduction of a bond between spins of the same sign defines a “site-bond” percolation problem. More generally, we can assume that such a bond exists with probability p and the randomness associated with the bond probability p depends on the temperature T . The dependence of p on T can be determined by requiring that the percolation transition of the clusters occurs at the Ising critical point, and by requiring that the critical exponents associated with the clusters be identical to the analogous thermal exponents. For example, we can define a critical exponent ν_p to characterize the divergence of the connectedness length of the clusters near p_c . The analogous thermal exponent ν quantifies the divergence of the thermal correlation length ξ near T_c . It is possible to show analytically that these (and other) critical exponents are identical if we define the bond probability as

$$p = 1 - e^{-2J/kT}. \quad (\text{bond probability}) \quad (17.61)$$

The relation (17.61) holds for any spatial dimension. What is the value of p at $T = T_c$ for the two-dimensional Ising model on the square lattice?

Now that we know how to generate clusters of spins, we use these clusters to construct a global dynamics. One way (known as the Swendsen-Wang algorithm) is to assign all bonds between parallel spins with probability p . No bonds are included between sites that have different spin values. From this configuration of bonds, we can form clusters of spins using the Hoshen-Kopelman algorithm (see Section 13.3). The smallest cluster contains a single spin. After the clusters have been identified, all the spins in each cluster are flipped with probability 1/2.

Because it is nontrivial to determine all the clusters for a given configuration, we instead explore an algorithm that flips single clusters. The idea is to “grow” a single (site-bond) percolation cluster in a way that is analogous to the single (site) percolation cluster algorithm discussed in Section 14.1. The algorithm can be implemented by the following steps:

1. Choose a seed spin at random. Its four nearest neighbor sites (on the square lattice) are the perimeter sites. Form an ordered array corresponding to the perimeter spins that are parallel to the seed spin and define a counter for the total number of perimeter spins.
2. Choose the first spin in the ordered perimeter array. Remove it from the array and replace it by the last spin in the array. Generate a random number r . If $r \leq p$, a bond exists and the spin is added to the cluster.
3. If the spin is added to the cluster, inspect its parallel perimeter spins. If such a spin is not already part of the cluster, add it to the end of the array of perimeter spins.
4. Repeat steps 2 and 3 until no perimeter spins remain.
5. Flip all the spins in the single cluster.

This algorithm is known as single cluster flip or Wolff dynamics. Note that bonds rather than sites are tested so that a spin might have more than one chance to join a cluster. In the following, we consider both the static and dynamical properties of the two-dimensional Ising model using the Wolff algorithm to generate the configurations.

- a. Modify your program for the two-dimensional Ising model on a square lattice so that single cluster flip dynamics (the Wolff algorithm) is used. Compute the mean energy and magnetization for $L = 16$ as a function of T for $T = 2.0$ to 2.7 in steps of 0.1 . Compare your results to those obtained using the Metropolis algorithm. How many cluster flips do you need to obtain comparable accuracy at each temperature? Is the Wolff algorithm more efficient at every temperature?
- b. Fix T at the critical temperature of the infinite lattice ($T_c = 2/\ln(1 + \sqrt{2})$) and use finite size scaling to estimate the values of the various static critical exponents, for example, γ and α . Compare your results to those obtained using the Metropolis algorithm.
- c. Because we are generating site-bond percolation clusters, we can study their geometrical properties as we did for site percolation. For example, measure the distribution sn_s of cluster sizes at $p = p_c$ (see Problem ??a). How does n_s depend on s for large s (see Project 14.18)? What is the fractal dimension of the clusters?
- d. The natural unit of time for single cluster flip dynamics is the number of cluster flips t_{cf} . Measure $C_M(t_{\text{cf}})$ and/or $C_E(t_{\text{cf}})$ and estimate the corresponding correlation time τ_{cf} for $T = 2.5, 2.4, 2.3$, and T_c for $L = 16$. As discussed in Problem 17.12, τ_{cf} can be found from the relation, $\tau_{\text{cf}} = \sum_{t_{\text{cf}}=1} C(t_{\text{cf}})$. The sum is cut off at the first negative value of $C(t_{\text{cf}})$. To compare our results for the Wolff algorithm to our results for the Metropolis algorithm, we should use the same unit of time. Because only a fraction of the spins are updated at each cluster flip, the time t_{cf} is not equal to the usual unit of time which corresponds to an update of the entire lattice or one Monte Carlo step per spin. We have that τ measured in Monte Carlo steps per spin is related to τ_{cf} by $\tau = \tau_{\text{cf}} \langle c \rangle / L^2$, where $\langle c \rangle$ is the mean number of spins in the single clusters, and L^2 is the number of spins in the entire lattice. Measure $\langle c \rangle$ and compare your values for τ for the Wolff algorithm to the values of τ that you obtained using the Metropolis algorithm. Which values of τ are smaller?
- e. Use the finite size scaling relation $\tau_{\text{cf}} \sim L^{z_{\text{cf}}}$ at $T = T_c$ to estimate z_{cf} . Because z_{cf} is small, you will find it very difficult to obtain a good estimate. Verify that the mean cluster size scales as $\langle c \rangle \sim L^{\gamma/\nu}$ with $\gamma = 7/4$ and $\nu = 1$. (Note that these exponents are identical to the analogous thermal exponents.) To obtain the value of z that is directly comparable to the value found for the Metropolis algorithm, we need to rescale the time as in part (d). We have that $\tau \sim L^z \propto L^{z_{\text{cf}}} L^{\gamma/\nu} L^{-d}$. Hence, z is related to the measured value of z_{cf} by $z = z_{\text{cf}} - (d - \gamma/\nu)$. It has been suggested that $\tau \propto \ln L$, which would imply that $z = 0$, but the value of z is not known with confidence.

Project 17.24. Physical test of random number generators

In Section 12.6 we discussed various statistical tests for the quality of random number generators. In this project we will find that the usual statistical tests might not be sufficient for determining the quality of a random number generator for a particular application. The difficulty is that the quality of a random number generator for a specific application depends in part on how the subtle correlations that are intrinsic to all deterministic random number generators couple to the way that the random number sequences are used. In this project we explore the quality of two random number generators when they are used to implement single spin flip dynamics (Metropolis algorithm) and single cluster flip dynamics (Wolff algorithm) for the two-dimensional Ising model.

- a. Write subroutines to generate sequences of random numbers based on the linear congruential algorithm

$$x_n = 16807 x_{n-1} \bmod (2^{31} - 1) \quad (17.62)$$

and the generalized feedback shift register (GFSR) algorithm

$$x_n = x_{n-103} \oplus x_{n-250}. \quad (17.63)$$

In both cases x_n is the n th random number. Both algorithms require that x_n be divided by the largest possible value of x_n to obtain numbers in the range $0 \leq x_n < 1$. The GFSR algorithm requires bit manipulation and should be written in C or Fortran (see Appendices ?? or ??). Which random number generator does a better job of passing the various statistical tests discussed in Problem 12.18?

- b. Use the Metropolis algorithm and the linear congruential random number generator to determine the mean energy per spin E/N and the specific heat (per spin) C for the $L = 16$ Ising model at $T = T_c = 2/\ln(1 + \sqrt{2})$. Make ten independent runs (that is, ten runs that use different random number seeds), and compute the standard deviation of the means σ_m from the ten values of E/N and C , respectively. Published results by Ferrenberg, Landau, and Wong are for 10^6 Monte Carlo steps per spin for each run. Calculate the differences δ_e and δ_c between the average of E/N and C over the ten runs and the exact values (to five decimal places) $E/N = -1.45306$ and $C = 1.49871$. If the ratio δ/σ_m for the two quantities is order unity, then the random number generator does not appear to be biased. Repeat your runs using the GFSR algorithm to generate the random number sequences. Do you find any evidence of statistical bias?
- c. Repeat part (b) using Wolff dynamics. Do you find any evidence of statistical bias?
- d. Repeat the computations in parts (b) and (c) using the random number generator supplied with your programming language.

Project 17.25. Kosterlitz-Thouless transition in the planar model

The planar model (also called the x - y model) consists of spins of unit magnitude that can point in any direction in the x - y plane. The energy or Hamiltonian function of the planar model in zero magnetic field can be written as

$$E = -J \sum_{i,j=nn(i)} [s_{i,x}s_{j,x} + s_{i,y}s_{j,y}], \quad (17.64)$$

where $s_{i,x}$ represents the x -component of the spin at the i th site, J measures the strength of the interaction, and the sum is over all nearest neighbors. We can rewrite (17.64) in a simpler form by substituting $s_{i,x} = \cos \theta_i$ and $s_{i,y} = \sin \theta_i$. The result is

$$E = -J \sum_{i,j=nn(i)} \cos(\theta_i - \theta_j), \quad (17.65)$$

where θ_i is the angle that the i th spin makes with the x axis. The most studied case is the two-dimensional model on a square lattice. In this case the mean magnetization $\langle \mathbf{M} \rangle = 0$ for all

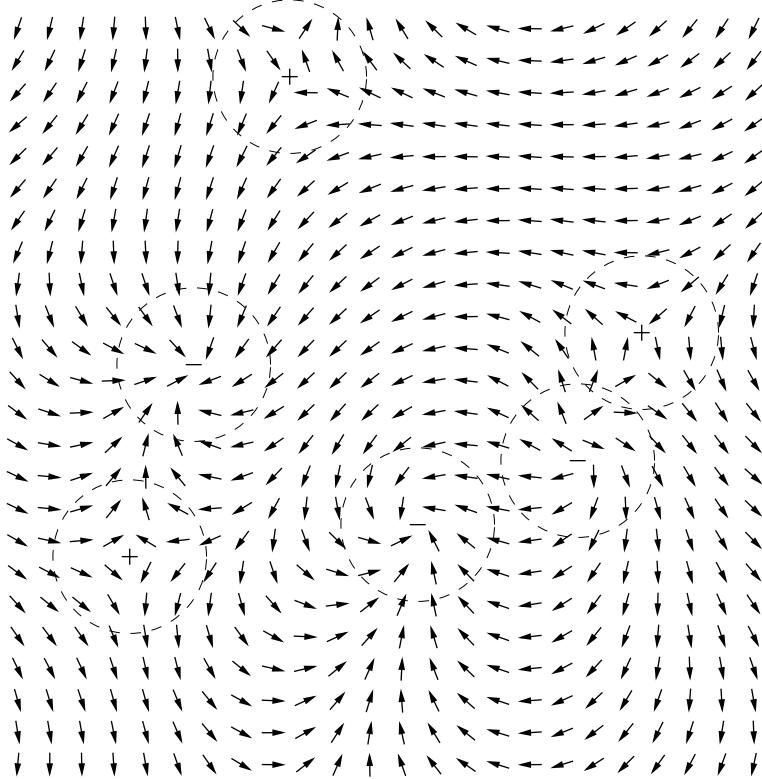


Figure 17.7: A typical configuration of the planar model on a 24×24 square lattice that has been quenched from $T = \infty$ to $T = 0$ and equilibrated for 200 Monte Carlo steps per spin after the quench. Note that there are six vortices. The circle around each vortex is a guide to the eye and is not meant to indicate the size of the vortex.

temperatures $T > 0$, but nevertheless, there is a phase transition at a nonzero temperature, T_{KT} , the Kosterlitz-Thouless (KT) transition. For $T \leq T_{KT}$, the spin-spin correlation function $C(r)$ decreases as a power law for increasing r ; for $T > T_{KT}$, $C(r)$ decreases exponentially. The power law decay of $C(r)$ for all $T \leq T_{KT}$ implies that every temperature below T_{KT} acts as if it were a critical point. We say that the planar model has a line of critical points. In the following, we explore some of the properties of the planar model and the mechanism that causes the transition.

- Write a Monte Carlo program to simulate the planar model on a square lattice using periodic boundary conditions. Because θ and hence the energy of the system is a continuous variable, it is not possible to store the previously computed values of the Boltzmann factor for each possible value of ΔE . Instead, of computing $e^{-\beta \Delta E}$ for each trial change, it is faster to set up an array w such that the array element $w(j) = e^{-\beta \Delta E}$, where j is the integer part of $1000\Delta E$. This procedure leads to an energy resolution of 0.001, which should be sufficient for most purposes.

- b. One way to show that the magnetization $\langle \mathbf{M} \rangle$ vanishes for all T is to compute $\langle \theta^2 \rangle$, where θ is the angle that a spin makes with the magnetization \mathbf{M} at any given instant. (Although the mean magnetization vanishes, $\mathbf{M} \neq 0$ at any given instant.) Compute $\langle \theta^2 \rangle$ as a function of the number of spins N at $T = 0.1$, and show that $\langle \theta^2 \rangle$ diverges as $\ln N$. Begin with a 4×4 lattice and choose the maximum change in θ_i to be $\Delta\theta_{\max} = 1.0$. If necessary, change θ_{\max} so that the acceptance probability is about 40%. If $\langle \theta^2 \rangle$ diverges, then the spins are not pointing along any preferred direction, and hence there is no mean magnetization.
- c. Modify your program so that an arrow is drawn at each site to show the orientation of each spin. We will look at a typical configuration and analyze it visually. Begin with a 32×32 lattice with spins pointing in random directions and do a temperature quench from $T = \infty$ to $T = 0.5$. (Simply change the value of β in the Boltzmann probability.) Such a quench should lock in some long lived, but metastable vortices. A vortex is a region of the lattice where the spins rotate by at least 2π as your eye moves around a closed path (see Fig. 17.7). To determine the center of a vortex, choose a group of four spins that are at the corners of a unit square, and determine whether the spins turn by $\pm 2\pi$ as your eye goes from one spin to the next in a counterclockwise direction around the square. Assume that the difference between the direction of two neighboring spins, $\delta\theta$, is in the range $-\pi < \delta\theta < \pi$. A total rotation of $+2\pi$ indicates the existence of a positive vortex, and a change of -2π indicates a negative vortex. Count the number of positive and negative vortices. Repeat these observations on several configurations. What can you say about the number of vortices of each sign?
- d. Write a subroutine to determine the existence of a vortex for each 1×1 square of the lattice. Represent the center of the vortices using a different symbol to distinguish between a positive and a negative vortex. Do a Monte Carlo simulation to compute the mean energy, specific heat, and number of vortices in the range from $T = 0.5$ to $T = 1.5$ in steps of 0.1. Use the last configuration at the previous temperature as the first configuration for the next temperature. Begin at $T = 0.5$ with all $\theta_i = 0$. Draw the vortex locations for the last configuration at each temperature. Use at least 1000 Monte Carlo steps per spin at each temperature to equilibrate and at least 5000 Monte Carlo steps per spin for computing the averages. Use an 8×8 or 16×16 lattice if your computer resources are limited, and larger lattices if you have sufficient resources. Describe the T dependence of the energy, specific heat, and vorticity (equal to the number of vortices per area). Plot the logarithm of the vorticity versus T for $T < 1.1$. What can you conclude about the T -dependence of the vorticity? Explain why this form is reasonable. Describe the vortex configurations. At what temperature can you find a vortex that appears to be free, that is, a vortex that is not obviously paired up with another vortex of opposite sign?
- e. The Kosterlitz-Thouless theory predicts that the susceptibility χ diverges above the transition as

$$\chi \sim A e^{b/\epsilon^\nu}, \quad (17.66)$$

where ϵ is the reduced temperature $\epsilon = (T - T_{KT})/T_{KT}$, $\nu = 0.5$, and A and b are nonuniversal constants. Compute χ from the relation (17.14) with $\mathbf{M} = 0$ because the mean magnetization vanishes. Assume the exponential form (17.66) for χ in the range $T = 1$ and $T = 1.2$ with $\nu = 0.7$, and find the best values of T_{KT} , A , and b . (Although the analytical theory predicts $\nu = 0.5$, simulations for small systems indicate that $\nu = 0.7$ gives a better fit.) One way to

determine T_{KT} , A , and b is to assume a value of T_{KT} and then do a least squares fit of $\ln \chi$ to determine A and b . Choose the set of parameters that minimizes the variance of $\ln \chi$. How does your estimated value of T_{KT} compare with the temperature at which free vortices first appear? At what temperature does the specific heat have a peak? The Kosterlitz-Thouless theory predicts that the specific heat peak does not occur at T_{KT} . This result has been confirmed by simulations (see Tobochnik and Chester). To obtain quantitative results, you will need lattices larger than 32×32 .

Project 17.26. Classical Heisenberg model in two dimensions

The energy or Hamiltonian of the classical Heisenberg model is similar to the Ising model and the planar model, except that the spins can point in any direction in three dimensions. The energy in zero external magnetic field is

$$E = -J \sum_{i,j=\text{nn}(i)}^N \mathbf{s}_i \cdot \mathbf{s}_j, \quad (17.67)$$

where \mathbf{s} is a classical vector of unit length. The spins have three components, in contrast to the spins in the Ising model which only have one component, and the spins in the planar model which have two components. We will consider the two-dimensional Heisenberg model for which the spins are located on a two-dimensional lattice.

Early simulations and approximate theories led researchers to believe that there was a continuous phase transition, similar to that found in the Ising model. The Heisenberg model received more interest after it was related to the confinement for quarks. Lattice models of the interaction between quarks, called lattice gauge theories, predict that the confinement of quarks can be explained if there are no phase transitions in these models. (The lack of a phase transition in these models implies that the attraction between quarks grows with distance.) The Heisenberg model is a two-dimensional analog of the four-dimensional models used to model quark-quark interactions. Shenker and Tobochnik used a combination of Monte Carlo and renormalization group methods to show that this model does not have a phase transition. Subsequent work on lattice gauge theories showed similar behavior.

- a. Modify your Ising model program to simulate the Heisenberg model in two dimensions. One way to do so is to define three arrays, one for each of the three components of the unit spin vectors. A trial Monte Carlo move consists of randomly changing the direction of a spin, \mathbf{s}_i . First compute a small vector $\Delta\mathbf{s} = \Delta s_{\max}(p_1, p_2, p_3)$, where $-1 \leq p_n \leq 1$ is a uniform random number, and Δs_{\max} is the maximum change of any spin component. If $|\Delta\mathbf{s}| > \Delta s_{\max}$, then compute another $\Delta\mathbf{s}$. This latter step is necessary to insure that the change in a spin direction is symmetrically distributed around the current spin direction. Next let the trial spin equal $\mathbf{s}_i + \Delta\mathbf{s}$ normalized to a unit vector. The standard Metropolis algorithm can now be used to determine if the trial spin is accepted. Compute the mean energy, specific heat, and susceptibility as a function of T . Choose lattice sizes of $L = 8, 16, 32$ and larger if possible and average over at least 2000 Monte Carlo steps per spin at each temperature. Is there any evidence of a phase transition? Does the susceptibility appear to diverge at a nonzero temperature? Plot the natural log of the susceptibility versus the inverse temperature, and determine the temperature dependence of the susceptibility in the limit of low temperatures.

- b. Use the Lee-Kosterlitz analysis at the specific heat peak to determine if there is a phase transition.

Project 17.27. Ground state energy of the Ising spin glass

A spin glass is a magnetic system with frozen-in disorder. An example of such a system is the Ising model with the exchange constant J_{ij} between nearest neighbor spins randomly chosen to be ± 1 . The disorder is said to be “frozen-in” because the set of interactions $\{J_{ij}\}$ does not change with time. Because the spins cannot arrange themselves so that every pair of spins is in its lowest energy state, the system exhibits frustration similar to the antiferromagnetic Ising model on a triangular lattice (see Problem 17.15). Is there a phase transition in the spin glass model, and if so, what is its nature? The answers to these questions are very difficult to obtain by doing simulations. One of the difficulties is that we need to do not only an average over the possible configurations of spins for a given set of $\{J_{ij}\}$, but we also need to average over different realizations of the interactions. Another difficulty is that there are many local minima in the energy (free energy at finite temperature) as a function of the configurations of spins, and it is very difficult to find the global minimum. As a result, Monte Carlo simulations typically become stuck in these local minima or metastable states. Detailed finite size scaling analyses of simulations indicate that there might be a transition in three dimensions. It is generally accepted that the transition in two dimensions is at zero temperature. In the following, we will look at some of the properties of an Ising spin glass on a square lattice at low temperatures.

- a. Write a program to apply simulated annealing to an Ising spin glass using the Metropolis algorithm with the temperature fixed at each stage of the annealing schedule (see Problem 17.22a). Search for the lowest energy configuration for a fixed set of $\{J_{ij}\}$. Use at least one other annealing schedule for the same $\{J_{ij}\}$ and compare your results. Then find the ground state energy for at least ten other sets of $\{J_{ij}\}$. Use lattice sizes of $L = 5$ and $L = 10$. Discuss the nature of the ground states you are able to find. Is there much variation in the ground state energy E_0 from one set of $\{J_{ij}\}$ to another? Theoretical calculations give an average over realizations of $\overline{E_0}/N \approx -1.4$. If you have sufficient computer resources, repeat your computations for the three-dimensional spin glass.
- b. Modify your program to do simulated annealing using the demon algorithm (see Problem 17.22b). How do your results compare to those that you found in part (a)?

Project 17.28. Zero temperature dynamics of the Ising model We have seen that various kinetic growth models (Section 14.3) and reaction-diffusion models (Section 12.4) lead to interesting and nontrivial behavior. Similar behavior can be seen in the zero temperature dynamics of the Ising model. Consider the one-dimensional Ising model with $J > 0$ and periodic boundary conditions. The initial orientation of the spins is chosen at random. We update the configurations by choosing a spin at random and computing the change in energy ΔE . If $\Delta E < 0$, then flip the spin; else if $\Delta E = 0$, flip the spin with 50% probability. The spin is not flipped if $\Delta E > 0$. This type of Monte Carlo update is known as Glauber dynamics. How does this algorithm differ from the Metropolis algorithm at $T = 0$?

The quantity of interest is $f(t)$, the fraction of spins that flip for the first time at time t . As usual, the time is measured in terms of Monte Carlo steps per spin. Published results (Derrida,

Bray, and Godrèche) for $N = 10^5$ indicate that $f(t)$

$$f(t) \sim t^{-\theta} \quad (17.68)$$

for $t \approx 3$ to $t \approx 10,000$ with $\theta \approx 0.37$. Verify this result and extend your results to the one-dimensional q -state Potts model. In the latter model each site is initially given a random integer between 1 and q . A site is chosen at random and set equal to either of its two neighbors with equal probability. The value of the exponent θ is not understood at present, but might be related to analogous behavior in reaction-diffusion models.

Project 17.29. The inverse power law potential

Consider the inverse power law potential

$$V(r) = V_0 \left(\frac{\sigma}{r}\right)^n \quad (17.69)$$

with $V_0 > 0$. One reason for interest in potentials of this form is that thermodynamic quantities such as the mean energy E do not depend on V_0 and σ separately, but depend on a single dimensionless parameter. This dimensionless parameter can be defined as

$$\Gamma = \frac{V_0}{kT} \frac{\sigma}{a}, \quad (17.70)$$

where a is defined in three and two dimensions by $4\pi a^3 \rho / 3 = 1$ and $\pi a^2 \rho = 1$, respectively. The length a is proportional to the mean distance between particles. A Coulomb interaction corresponds to $n = 1$, and a hard sphere system corresponds to $n \rightarrow \infty$. What phases do you expect to occur for arbitrary n ?

- a. Compare the qualitative features of $g(r)$ for a “soft” potential with $n = 4$ to a system of hard disks at the same density.
- b. Let $n = 12$ and compute the mean energy E as a function of T for fixed density for a three-dimensional system. Fix T and consider $N = 16, 32, 64$, and 128 . Does E depend on N ? Can you extrapolate your results for the N -dependence of E to $N \rightarrow \infty$? Fix N and determine E as a function of Γ . Do you see any evidence of a phase transition? If so, estimate the value of Γ at which it occurs. What is the nature of the transition if it exists?

Project 17.30. Rare gas clusters There has been much recent interest in structures that contain many particles, but that are not macroscopic. An example is the unusual structure of sixty carbon atoms known as a “buckeyball.” A less unusual structure is a cluster of argon atoms. Questions of interest include the structure of the clusters, the existence of “magic” numbers of particles for which the cluster is particularly stable, the temperature dependence of the thermodynamic quantities, and the possibility of different phases. This latter question has been subject to some controversy, because transitions between different kinds of behavior in finite systems are not nearly as sharp as they are for infinite systems.

- a. Write a Monte Carlo program to simulate a three-dimensional system of particles interacting via the Lennard-Jones potential. Use open boundary conditions, that is, do not enclose the system in a box. The number of particles N and the temperature T should be input parameters.

- b. Find the ground state energy E_0 as a function of N . For each value of N begin with a random initial configuration and accept any trial displacement that lowers the energy. Repeat for at least ten different initial configurations. Plot E_0/N versus N for $N = 2$ to 20 and describe the qualitative dependence of E_0/N on N . Is there any evidence of magic numbers, that is, value(s) of N for which E_0/N is a minimum? For each value of N save the final configuration. If you have access to a three-dimensional graphics program, plot the positions of the atoms. Does the cluster look like a part of a crystalline solid?
- c. Repeat part (b) using simulated annealing. The initial temperature should be sufficiently low so that the particles do not move far away from each other. Slowly lower the temperature according to some annealing schedule. Do your results for E_0/N differ from part (b)?
- d. To gain more insight into the structure of the clusters, compute the mean number of neighbors per particle for each value of N . What is a reasonable criteria for two particles to be neighbors? Also compute the mean distance between each pair of particles. Plot both quantities as a function of N , and compare their dependence on N with your plot of E_0/N .
- e. Is it possible to find any evidence for a “melting” transition? Begin with the configuration that has the minimum value of E_0/N and slowly increase the temperature T . Compute the energy per particle and the mean square displacement of the particles from their initial positions. Plot your results for these quantities versus T .

Project 17.31. Hard disks Although we have mentioned (see Section 17.7) that there is reasonable evidence for a transition in a hard disk system, the nature of the transition still is a problem of current research. In this project we follow the work of Lee and Strandburg and apply the constant pressure Monte Carlo method (see Section 17.9) and the Lee-Kosterlitz method (see Section 17.8) to investigate the nature of the transition. Consider $N = L^2$ hard disks of diameter $\sigma = 1$ in a two-dimensional box of volume $V = \sqrt{3}L^2v/2$ with periodic boundary conditions. The quantity $v \geq 1$ is the reduced volume and is related to the density ρ by $\rho = N/V = 2/(\sqrt{3}v)$; $v = 1$ corresponds to maximum packing. The aspect ratio of $2/\sqrt{3}$ is used to match the perfect triangular lattice. We can perform a constant pressure (actually constant $p^* = P/kT$) Monte Carlo simulation as follows. The trial displacement of each disk is implemented as discussed in Section 17.7. Lee and Strandburg find that a maximum displacement of 0.09 gives a 45% acceptance probability. The other type of move is a random isotropic change of the volume of the system. If the change of the volume leads to an overlap of the disks, the change is rejected. Otherwise, if the trial volume \tilde{V} is less than the current volume V , the change is accepted. A larger trial volume is accepted with probability

$$e^{-p^*(\tilde{V}-V)+N \ln \tilde{V}/V}. \quad (17.71)$$

Volume changes are attempted 40–200 times for each set of individual disk moves. The quantity of interest is $N(v)$, the distribution of reduced volume v . Because we need to store information about $N(v)$ in an array, it is convenient to discretize the volume in advance and choose the mesh size so that the acceptance probability for changing the volume by one unit is 40–50%. Do a Monte Carlo simulation of the hard disk system for $L = 10$ ($N = 100$) and $p^* = 7.30$. Published results are for 10^7 Monte Carlo steps. To apply the Lee-Kosterlitz method, smooth $\ln N(v)$ by fitting it to an eighth-order polynomial. Then extrapolate $\ln N(v)$ using the histogram method to determine

$p_c^*(L = 10)$, the pressure at which the two peaks of $N(v)$ are of equal height. What is the value of the free energy barrier ΔF ? If sufficient computer resources are available, compute ΔF for larger L (published results are for $L = 10, 12, 14, 16$, and 20) and determine if ΔF depends on L . Can you reach any conclusions about the nature of the transition?

Appendix 17A: Fluctuations in the Canonical Ensemble

We first obtain the relation of the constant volume heat capacity C_V to the energy fluctuations in the canonical ensemble. We adopt the notation $U = \langle E \rangle$ and write C_V as

$$C_V = \frac{\partial U}{\partial T} = -\frac{1}{kT^2} \frac{\partial U}{\partial \beta}. \quad (17.72)$$

From (17.3) we have

$$U = -\frac{\partial}{\partial \beta} \ln Z \quad (17.73)$$

and

$$\frac{\partial U}{\partial \beta} = -\frac{1}{Z^2} \frac{\partial Z}{\partial \beta} \sum_s E_s e^{-\beta E_s} - \frac{1}{Z} \sum_s E_s^2 e^{-\beta E_s} \quad (17.74)$$

$$= \langle E \rangle^2 - \langle E^2 \rangle. \quad (17.75)$$

The relation (17.12) follows from (17.72) and (17.75). Note that the heat capacity is at constant volume because the partial derivatives were performed with the energy levels E_s kept constant. The corresponding quantity for a magnetic system is the heat capacity at constant external magnetic field.

The relation of the magnetic susceptibility χ to the fluctuations of the magnetization M can be obtained in a similar way. We assume that the energy can be written as

$$E_s = E_{0,s} - HM_s, \quad (17.76)$$

where $E_{0,s}$ is the energy in the absence of a magnetic field, H is the external applied field, and M_s is the magnetization in the s state. The mean magnetization is given by

$$\langle M \rangle = \frac{1}{Z} \sum_s M_s e^{-\beta E_s}. \quad (17.77)$$

Because $\partial E_s / \partial H = -M_s$, we have

$$\frac{\partial Z}{\partial H} = \sum_s \beta M_s e^{-\beta E_s}. \quad (17.78)$$

Hence we obtain

$$\langle M \rangle = \frac{1}{\beta} \frac{\partial}{\partial H} \ln Z. \quad (17.79)$$

Number spins up	Degeneracy	Energy	Magnetization
4	1	-8	4
3	4	0	2
2	4	0	0
2	2	8	0
1	4	0	-2
0	1	-8	-4

Table 17.1: The energy and magnetization of the 2^4 states of the zero field Ising model on the 2×2 square lattice. The degeneracy is the number of microstates with the same energy.

If we use (17.77) and (17.79), we find

$$\frac{\partial \langle M \rangle}{\partial H} = -\frac{1}{Z^2} \frac{\partial Z}{\partial H} \sum_s M_s e^{-\beta E_s} + \frac{1}{Z} \sum_s \beta M_s^2 e^{-\beta E_s} \quad (17.80)$$

$$= -\beta \langle M \rangle^2 + \beta \langle M^2 \rangle. \quad (17.81)$$

The relation (17.14) for the zero field susceptibility follows from (17.81) and the definition (17.13).

Appendix 17B: Exact Enumeration of the 2×2 Ising Model

Because the number of possible states or configurations of the Ising model increases as 2^N , we can enumerate the possible configurations only for small N . As an example, we calculate the various quantities of interest for a 2×2 Ising model on the square lattice with periodic boundary conditions. In Table 17.1 we group the sixteen states according to their total energy and magnetization.

We can compute all the quantities of interest using Table 17.1. The partition function is given by

$$Z = 2 e^{8\beta J} + 12 + 2 e^{-8\beta J}. \quad (17.82)$$

If we use (17.73) and (17.82), we find

$$U = -\frac{\partial}{\partial \beta} \ln Z = -\frac{1}{Z} [2(8)e^{8\beta J} + 2(-8)e^{-8\beta J}]. \quad (17.83)$$

Because the other quantities of interest can be found in a similar manner, we only give the results:

$$\langle E^2 \rangle = \frac{1}{Z} [(2 \times 64) e^{8\beta J} + (2 \times 64) e^{-8\beta J}] \quad (17.84)$$

$$\langle M \rangle = \frac{1}{Z} (0) = 0 \quad (17.85)$$

$$\langle |M| \rangle = \frac{1}{Z} [(2 \times 4) e^{8\beta J} + 8 \times 2] \quad (17.86)$$

$$\langle M^2 \rangle = \frac{1}{Z} [(2 \times 16) e^{8\beta J} + 8 \times 4] \quad (17.87)$$

The dependence of C and χ on βJ can be found by using (17.83) and (17.84) and (17.85) and (17.87) respectively.

References and Suggestions for Further Reading

- M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Clarendon Press (1987). See Chapter 4 for a discussion of Monte Carlo methods.
- K. Binder, editor, *Monte Carlo Methods in Statistical Physics*, second edition, Springer-Verlag (1986). Also see K. Binder, editor, *Applications of the Monte Carlo Method in Statistical Physics*, Springer-Verlag (1984) and K. Binder, editor, *The Monte Carlo Method in Condensed Matter Physics*, Springer-Verlag (1992).
- Marvin Bishop and C. Bruin, “The pair correlation function: a probe of molecular order,” *Amer. J. Phys.* **52**, 1106 (1984). The authors compute the pair correlation function for a two-dimensional Lennard-Jones model.
- James B. Cole, “The statistical mechanics of image recovery and pattern recognition,” *Amer. J. Phys.* **59**, 839 (1991). A discussion of the application of simulated annealing to the recovery of images from noisy data.
- B. Derrida, A. J. Bray, and C. Godrèche, “Non-trivial exponents in the zero temperature dynamics of the 1D Ising and Potts models,” *J. Phys. A* **27**, L357 (1994).
- Jerome J. Erpenbeck and Marshall Luban, “Equation of state for the classical hard-disk fluid,” *Phys. Rev. A* **32**, 2920 (1985). These workers use a combined molecular dynamics/Monte Carlo method and consider 1512 and 5822 disks.
- Alan M. Ferrenberg, D. P. Landau, and Y. Joanna Wong, “Monte Carlo simulations: hidden errors from “good” random number generators,” *Phys. Rev. Lett.* **69**, 3382 (1992).
- Alan M. Ferrenberg and Robert H. Swendsen, “New Monte Carlo technique for studying phase transitions,” *Phys. Rev. Lett.* **61**, 2635 (1988); “Optimized Monte Carlo data analysis,” *Phys. Rev. Lett.* **63**, 1195 (1989); “Optimized Monte Carlo data analysis,” *Computers in Physics* **35**, 101 (1989). The second and third papers discuss using the multiple histogram method with data from simulations at more than one temperature.
- Harvey Gould and Jan Tobochnik, “Overcoming critical slowing down,” *Computers in Physics* **34**, 82 (1989).
- Hong Guo, Martin Zuckermann, R. Harris, and Martin Grant, “A fast algorithm for simulated annealing,” *Physica Scripta* **T38**, 40 (1991).
- J. Kertész, J. Cserti and J. Szép, “Monte Carlo simulation programs for microcomputer,” *Eur. J. Phys.* **6**, 232 (1985).
- S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science* **220**, 671 (1983). See also, S. Kirkpatrick and G. Toulouse, “Configuration space analysis of traveling salesman problems,” *J. Physique* **46**, 1277 (1985).
- J. M. Kosterlitz and D. J. Thouless, *J. Phys. C* **6**, 1181 (1973); J. M. Kosterlitz, *J. Phys. C* **7**, 1046 (1974).

- D. P. Landau, “Finite-size behavior of the Ising square lattice,” *Phys. Rev.* **B13**, 2997 (1976). A clearly written paper on a finite-size scaling analysis of Monte Carlo data. See also D. P. Landau, “Finite-size behavior of the simple-cubic Ising lattice,” *Phys. Rev.* **B14**, 255 (1976).
- D. P. Landau and R. Alben, “Monte Carlo calculations as an aid in teaching statistical mechanics,” *Am. J. Phys.* **41**, 394 (1973).
- Jooyoung Lee and J. M. Kosterlitz, “New numerical method to study phase transitions,” *Phys. Rev. Lett.* **65**, 137 (1990); *ibid.*, “Finite-size scaling and Monte Carlo simulations of first-order phase transitions,” *Phys. Rev. B* **43**, 3265 (1991).
- Jooyoung Lee and Katherine J. Strandburg, “First-order melting transition of the hard-disk system,” *Phys. Rev. B* **46**, 11190 (1992).
- N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Keller, “Equation of state calculations for fast computing machines,” *J. Chem. Phys.* **6**, 1087 (1953).
- J. Marro and R. Toral, “Microscopic observations on a kinetic Ising model,” *Am. J. Phys.* **54**, 1114 (1986).
- M. A. Novotny, “A new approach to an old algorithm for the simulation of Ising-like systems,” *Computers in Physics* **91**, 46 (1995). The n -fold way algorithm is discussed.
- Ole G. Mouritsen, *Computer Studies of Phase Transitions and Critical Phenomena*, Springer-Verlag (1984).
- E. P. Münger and M. A. Novotny, “Reweighting in Monte Carlo and Monte Carlo renormalization-group studies,” *Phys. Rev. B* **43**, 5773 (1991). The authors discuss the histogram method and combine it with renormalization group calculations.
- Michael Plischke and Birger Bergersen, *Equilibrium Statistical Physics*, second edition, Prentice Hall (1994). A graduate level text that discusses some of the more contemporary topics in statistical physics, many of which have been influenced by computer simulations.
- William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes*, second edition, Cambridge University Press (1992). A Fortran program for the traveling salesman problem is given in Section 10.9.
- Stephen H. Shenker and Jan Tobochnik, “Monte Carlo renormalization-group analysis of the classical Heisenberg model in two dimensions,” *Phys. Rev. B* **22**, 4462 (1980).
- Amihai Silverman and Joan Adler, “Animated Simulated Annealing,” *Computers in Physics* **6**, 277 (1992). The authors describe a simulation of the annealing process to obtain a defect free single crystal of a model material.
- Zoran Slanič, Harvey Gould, and Jan Tobochnik, “Dynamics of the classical Heisenberg chain,” *Computers in Physics* **5**, 630 (1991). Unlike the Ising model, the Heisenberg model has an intrinsic dynamics and can be studied by molecular dynamics methods.

- H. Eugene Stanley, *Introduction to Phase Transitions and Critical Phenomena*, Oxford University Press (1971). See Appendix B for the exact solution of the zero-field Ising model for a two-dimensional lattice.
- Jan Tobochnik and G. V. Chester, "Monte Carlo study of the planar model," *Phys. Rev. B* **20**, 3761 (1979).
- J. P. Valleau and S. G. Whittington, "A guide to Monte Carlo for statistical mechanics: 1. Highways," in *Statistical Mechanics, Part A*, Bruce J. Berne, editor, Plenum Press (1977). See also J. P. Valleau and G. M. Torrie, "A guide to Monte Carlo for statistical mechanics: 2. Byways," *ibid.*
- I. Vattulainen, T. Ala-Nissila, and K. Kankaala, "Physical tests for random numbers in simulations," *Phys. Rev. Lett.* **73**, 2513 (1994).