

# 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  $\rightarrow$  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.

<b>Laboratory experiment</b>	<b>Computer simulation</b>
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.

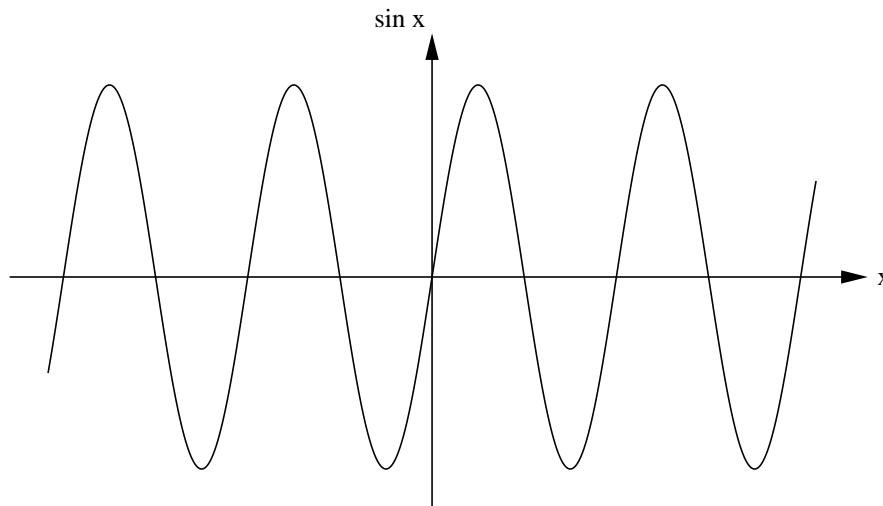


Figure 1.1: Plot of the sine function.

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 a 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 a 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/](http://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).