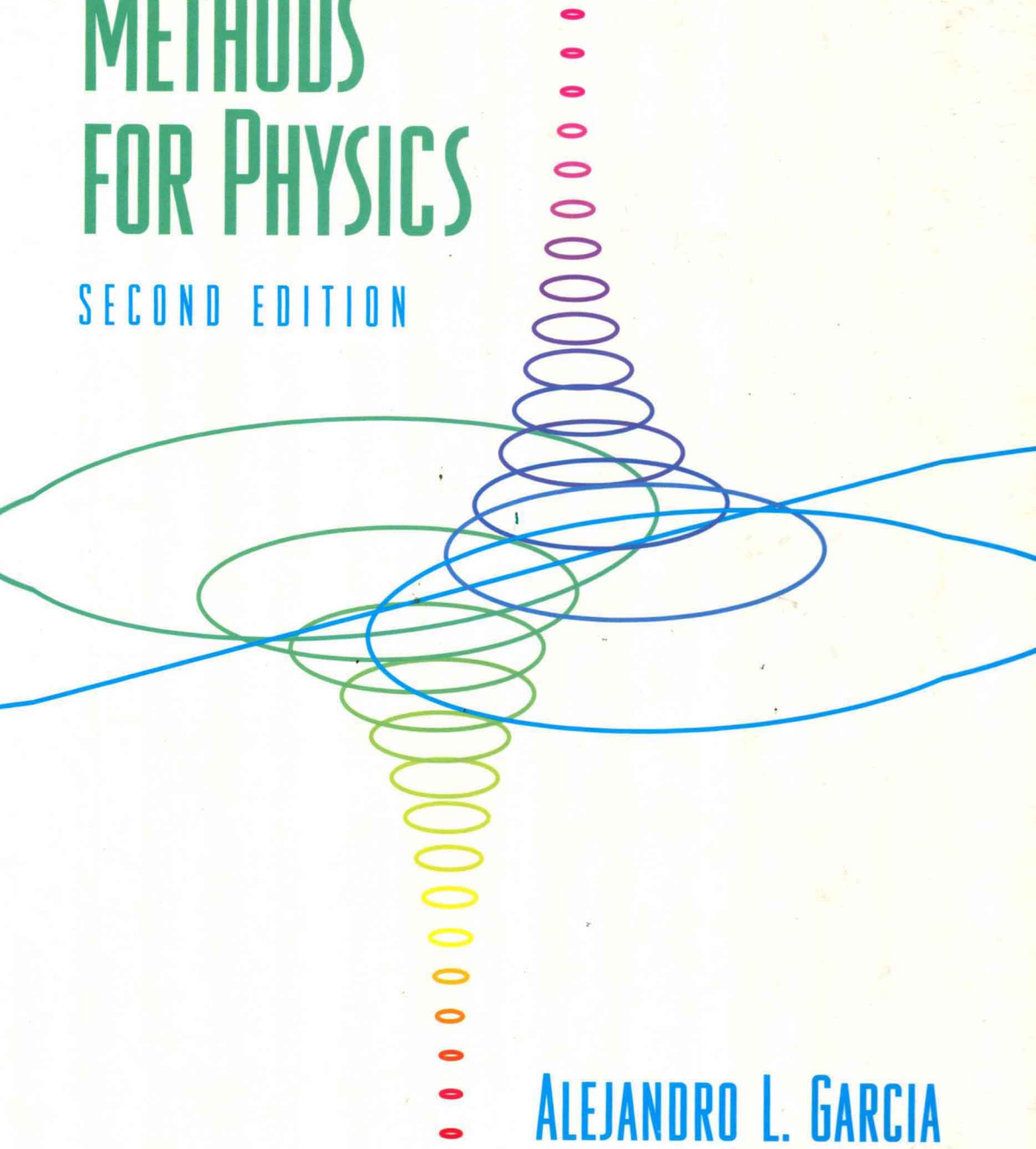
Numerical Methods for Physics

SECOND EDITION



Alejandro L. Garcia

SECOND EDITION

Numerical Methods for Physics

Alejandro L. Garcia
San Jose State University

Prentice Hall, Upper Saddle River, New Jersey 07458

Library of Congress Cataloging-in-Publication Data

Garcia, Alejandro L.,
Numerical methods for physics / Alejandro L. Garcia. – 2nd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-13-906744-2
1. Mathematical physics. 2. Differential equations,
Partial–Numerical solutions. 3. Physics–Data processing. I.
Title.
QC20 .G37 2000
530.15–dc21

99-28552
CIP

Executive Editor: Alison Reeves
Editor-in-Chief: Paul F. Corey
Assistant Vice President of Production and Manufacturing: David W. Riccardi
Executive Managing Editor: Kathleen Schiaparelli
Assistant Managing Editor: Lisa Kinne
Production Editor: Linda DeLorenzo
Marketing Manager: Steven Sartori
Editorial Assistant: Gillian Buonanno
Manufacturing Manager: Trudy Pisciotti
Art Director: Jayne Conte
Cover Designer: Bruce Kenselaar
Cover Art: John Christiana

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-906744-2

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall (*Singapore*) Pte Ltd
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Preface

When I was an undergraduate, computers were just beginning to be introduced into the university curriculum. Physics majors were expected to take a single semester of Pascal taught by the computer science department. We wrote programs to sort lists, process a payroll, and so forth, but were expected to acquire the specialized tools of scientific computing on our own. Most of us wasted many human and computer hours learning them by trial and error.

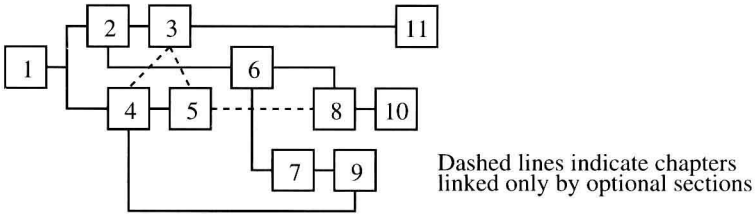
In recent years, many departments have added a computational physics course, taught by physicists, to their curricula. However, there is still considerable debate as to how this course should be organized. My philosophy is to use the upper division/graduate mathematical physics course as a model. Consider the following parallels between this text and a typical math physics book: A variety of numerical and analytical techniques used in physics are covered. Topics include ordinary and partial differential equations, linear algebra, Fourier transforms, integration, and probability. Because the text is written for physicists, these techniques are applied to solving realistic problems, many of which the students have encountered in other courses.

Numerical Methods for Physics is organized to cover what I believe are the most important, basic computational methods for physicists. The structure of the book differs considerably from the generic numerical analysis text. For example, about a third of the book is devoted to partial differential equations. This emphasis is natural considering the fundamental importance of Maxwell's equations, the Schrödinger equation, the Boltzmann equation, and so forth. Chapters 6 and 7 introduce some methods in computational fluid dynamics, an increasingly important topic in the fields of nonlinear physics, environmental physics, and astrophysics.

Numerical techniques may be classified as basic, advanced, and cutting edge. On the whole, this text covers only fundamental techniques; to work effectively with advanced numerical methods requires that the user first understand the basic algorithms. The discussion in the "Beyond This Chapter" section at the end of each chapter guides the reader to advanced algorithms and indicates when it is appropriate to use them. Unfortunately, the cutting edge moves so quickly that any attempt to summarize the latest algorithms would quickly be out of date.

The material in this text may be arranged in various ways to suit anything from a 10-week, upper-division class to a full-semester, graduate course. Most

chapters include optional sections that may be omitted without loss of continuity. Furthermore, entire chapters may be skipped; chapter interdependencies are indicated in the flow chart.



I have tried to present the algorithms in a clear, universal form that would allow the reader to easily implement them in *any* language. The programs are given in outline form in the main text with MATLAB and C++ listings in the appendices. In my classes, the students are allowed to use any language, yet I find that most end up using MATLAB. Its plotting utilities are particularly good—all the graphical results in the book were generated directly from the MATLAB programs. Advanced programmers (and students wishing to improve this skill) prefer using C++. FORTRAN versions of the programs, along with the MATLAB and C++ source code, are available online from Prentice Hall.

The over 250 exercises should be regarded as an essential part of the text. The time needed to do a problem ranges from 30 minutes to 2 days; in my classes, I assign about five exercises per week. Each exercise is labeled as:

[Pencil]	can be solved with pencil and paper.
[Computer]	requires using the computer.
[MATLAB]	best solved using MATLAB.
[C++]	best solved using C++.

While some texts emphasize month-long projects, I find that shorter exercises allow the class to move at a brisker pace, covering a wider variety of topics. Some instructors may wish to give one or two longer assignments, and many of the exercises may be expanded into such projects.

Readers familiar with the first edition will notice the following changes: C++ versions of the programs have been added, along with a new section (1.3) summarizing the language. The MATLAB programs have been updated to version 5. The discussion of derivatives has been moved from Chapter 1 to Chapter 2. A new section (6.3) has been added to Chapter 6. The discussion of hyperbolic partial differential equations has been collected into a new Chapter 7.

I wish to thank the people in my department, especially D. Strandburg, P. Hamill, A. Tucker, and J. Becker, for their strong support; my students and teaching assistants, J. Stroh, S. Moon, and D. Olson, who braved the rough waters of the early drafts; the National Science Foundation for its support of the computational physics program at San Jose State University; my editors at Prentice Hall and the technical staff of The MathWorks Inc. for their assistance; the National Oceanic and Atmospheric Administration Climate Monitoring and Diagnostics Laboratory for the CO_2 data used in Chapter 5. In addition, I

appreciate the comments of the following reviewers: David A. Boness, Seattle University; Wolfgang Christian, Davidson College; David M. Cook, Lawrence University; Harvey Gould, Clark University; Cleve Moler, The MathWorks, Inc; Cecile Penland, University of Colorado, Boulder; and Ross L. Spencer, Brigham Young University. Finally, I owe a special debt of gratitude to my entire family for their moral support as I wrote this book.

Alejandro L. Garcia

Dedicated to
Josefina Ovies García
and
Miriam González López

Contents

Chapter 1

Preliminaries

This chapter has no physics; to use the computer to do physics, one must first know how to use it to do math. The book presents algorithms in their general form, but when we sit down at the computer we have to give it instructions that it understands. Sections 1.2 and 1.3 present a synopsis of the MATLAB and C++ programming languages, and some simple programs are developed in Section 1.4. The chapter concludes with a discussion of the effect of hardware limitations (e.g., round-off errors) on mathematical calculations.

1.1 PROGRAMMING

General Thoughts

Before we get started, let me warn you that this book does not teach programming. Presumably, you have already learned a programming language (it doesn't really matter which one) and have had some practice in writing programs. This book covers numerical algorithms, specifically those that are most useful in physics. The style of presentation is informal. Instead of rigorously deriving all the details of all possible algorithms, I'll cover only the essential points and stress the practical methods.

If you've had a math course in numerical analysis, you may see some old friends (such as Romberg integration). This book emphasizes the application of such methods to physics problems. You will also learn some specialized techniques generally not presented in a mathematics course. If you have not had numerical analysis, don't worry. The book is organized assuming no prior knowledge of numerical methods.

In your earlier programming course I hope you learned about good programming style. I try to use what I consider good style in the programs, but everyone has personal preferences. The point of good style is to make your life easier by organizing and structuring your program development. Many programs in this book sacrifice efficiency for the sake of clarity. After you understand how a

program works, you should make it a regular exercise to improve it. However, always be sure to check your improved version with the original.

Most of the exercises in this book involve programming projects. In the first few chapters, many exercises require only that you modify an existing program. In the later chapters you are asked to write more and more of your own code. The exercises are purposely organized in this fashion to allow you to come up to speed on whatever computer system you choose to use. Unfortunately, computational physics is often like experimental work in the following regard: Debugging and testing is a slow, tedious, but necessary task similar to aligning optics or fixing vacuum leaks.

Programming Languages

In writing this book, one of the most difficult decisions I had to make was choosing a language. The obvious choices were Basic, FORTRAN, MATLAB, C++, and Java. I also considered symbolic manipulators such as Maple and Mathematica. When the first edition of this book appeared, there were significant differences among these choices. Some were more powerful, but difficult to use; others had better graphics, but were not portable across computing platforms, etc. Since that time, advances in software engineering have diminished the deficiencies (and in many ways the distinctiveness) of these languages, making the choice of language for the second edition even more difficult. With my editor's assistance, we put the question to students and instructors using the first edition, and their choices were MATLAB and C++.*

MATLAB is the language that I encourage my students to use in their course work. Although you may not be familiar with MATLAB, it is widely used in both academia and industry. It is especially popular in the engineering community and with applied mathematicians. MATLAB is very portable; it runs on Windows PCs, Macintoshes, and Unix workstations.

MATLAB is an interpreted language with excellent scientific libraries. Because it is an interpreted language, it is easy to use interactively, while the compiled libraries improve its performance. Being an interpreted language also makes MATLAB very clean. Many details (such as dimensioning matrices) are handled automatically. MATLAB has very good graphics facilities, including high-level routines (e.g., contour and surface plots). If you are comfortable using Basic, FORTRAN, or symbolic manipulators, you should have no trouble programming in MATLAB.

C++ is a rich, elegant, and powerful programming language. Most of the applications on your computer were probably written in C++. Arguably, FORTRAN remains the dominant language in the physics community, but C++ is the *lingua franca* of engineering. For these reasons many students are eager to learn this object-oriented language. C++ is difficult to master, but knowing the basics will suffice for programming the algorithms in this book. If you have programming experience with C or Java, you will probably enjoy using C++.

*FORTRAN versions of the programs in this book are also available online.

1.2 BASIC ELEMENTS OF MATLAB

This section summarizes the basic elements of MATLAB. Advanced features are introduced in later chapters as we need them. The MATLAB manuals include several tutorial chapters, along with a complete reference to the language. If you don't have a copy of the manual at hand, most of it is available from the built-in help system. For other MATLAB tutorials, see the texts by Etter [44] and by Hanselman and Littlefield [70].

Variables

The fundamental data type in MATLAB is the matrix (MATLAB is an acronym for MATrix LABoratory). A scalar is a 1×1 matrix, a row vector is a $1 \times N$ matrix, and a column vector is an $N \times 1$ matrix. Variables are not declared explicitly; MATLAB just dimensions them as they are used. For example, take the scalars x and y, the vectors $\mathbf{a}$ and $\mathbf{b}$, and the matrices $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{E}$, and give them the values

$$x = 3; \quad y = -2; \quad \mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}; \quad \mathbf{b} = \begin{bmatrix} 0 & 3 & -4 \end{bmatrix};$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \\ 1 & 2 & 0 \end{bmatrix}; \quad \mathbf{D} = \begin{bmatrix} 0 & 1 & 1 \\ 2 & 3 & -1 \\ 0 & 0 & 1 \end{bmatrix}; \quad \mathbf{E} = \begin{bmatrix} 1 & \pi \\ 0 & -1 \\ x & \sqrt{-1} \end{bmatrix} \tag{1.1}$$

In MATLAB these variables would be set by the assignment statements

```
x = 3;       % These are some simple assignments
y = -2;
a = [1; 2; 3];   % Column vector a; Row vector b
b = [0 3 -4];
C = [1 0 1; 0 1 -1; 1 2 0];   % Matrices
D = [0 1 1; 2 3 -1; 0 0 1];
E = [1 pi; 0 -1; x sqrt(-1)]; % In MATLAB, pi = 3.14...
```

Variable names in MATLAB, as in most languages, must start with a letter but can be composed of any combination of letters, numbers, and underscores (e.g., `mass`, `mass_of_particle`, `MassOfParticle2`). Anything following a percent sign is considered a comment in MATLAB.

The semicolon at the end of each assignment marks the end of the statement; you can put multiple statements on a line by separating them with semicolons. If this semicolon is omitted, the statement is assumed to end at the end of the line, and the value assigned is displayed on the screen (sometimes useful but more often annoying). Notice that rows in a matrix are separated by semicolons.

Two handy MATLAB functions for creating matrices are `zeros` and `ones`. The statement `A=zeros(M,N)` sets `A` to be an $M \times N$ matrix, with all elements equal to zero. The `ones` function works in the same fashion, but creates matrices filled with ones.

Mathematics

The basic arithmetic operations are defined in the natural manner. For example, the MATLAB statements

```
z = x - y;     % Some basic operations
t = b * a;
F = C + D;
G = C * E;
```

assign the values

$$z = 5; \quad t = -6; \quad \mathbf{F} = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 4 & -2 \\ 1 & 2 & 1 \end{bmatrix}; \quad \mathbf{G} = \begin{bmatrix} 4 & \pi + i \\ -3 & -1 - i \\ 1 & \pi - 2 \end{bmatrix} \tag{1.2}$$

The power operator is `^`, thus `2^3` is $2^3 = 8$. Other mathematical functions are available from MATLAB's large collection of built-in functions (Table 1.1).

MATLAB performs matrix multiplication; thus in the example above, $G_{ij} = \sum_k C_{ik} E_{kj}$. Notice that `b*a` is just the dot product of these vectors. MATLAB will balk if you try to do a matrix operation when the dimensions don't match (e.g., it will not compute `C+E`). For matrices, division is implemented by using Gaussian elimination (discussed in Chapter 4).

Sometimes we want to perform operations element by element, so MATLAB defines the operators `.* ./` and `.^`. Here are some examples of these array operations:

```
H = C .* D;    % These operations are performed
J = E .^ x;    % element-by-element
```

In this case $H_{i,j} = C_{i,j} D_{i,j}$ and $J_{i,j} = (E_{i,j})^x$, thus

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & 0 \end{bmatrix}; \quad \mathbf{J} = \begin{bmatrix} 1 & \pi^3 \\ 0 & -1 \\ 27 & -i \end{bmatrix} \tag{1.3}$$

Individual elements of a matrix may be addressed by using their indices. For example, `J(1,2)` equals π^3 and `J(3,1)` equals 27. Similarly, for vectors, `b(3)` (or `b(1,3)`) equals -4. Notice that matrix indices start at 1 and not 0.

Matrix **B** is the transpose of matrix **A** if $A_{ij} = B_{ji}$, that is, the rows and columns are exchanged. The Hermitian conjugate of a matrix is the transpose of its complex conjugate. In MATLAB

```
K = J';    % Hermitian conjugate
L = J.';   % Transpose
```

give the values

$$\mathbf{K} = \begin{bmatrix} 1 & 0 & 27 \\ \pi^3 & -1 & i \end{bmatrix}; \quad \mathbf{L} = \begin{bmatrix} 1 & 0 & 27 \\ \pi^3 & -1 & -i \end{bmatrix} \tag{1.4}$$

The Hermitian conjugate of **J** is **K**, while **L** is the transpose of **J**.

Table 1.1: Selected MATLAB mathematical functions.

`abs(x)`	Absolute value or complex magnitude
`norm(x)`	Magnitude of a vector
`sqrt(x)`	Square root
`sin(x)`, `cos(x)`	Sine and cosine
`tan(x)`	Tangent
`atan2(y,x)`	Arc tangent of `y/x` in $[0, 2\pi]$
`exp(x)`	Exponential
`log(x)`, `log10(x)`	Natural logarithm and base-10 logarithm
`rem(x,y)`	Remainder (modulo) function (e.g., `rem(10.3,4)=2.3`)
`floor(x)`	Round down to nearest integer (e.g., `floor(3.2)=3`)
`ceil(x)`	Round up to nearest integer (e.g., `ceil(3.2)=4`)
`rand(N)`	Uniformly distributed random numbers from the interval [0, 1). Returns $N \times N$ matrix.
`randn(N)`	Normal (Gaussian) distributed random numbers (zero mean, unit variance). Returns $N \times N$ matrix.

Loops and Conditionals

Repeated operations are performed by using loops. Here is an example of a `for` loop in MATLAB:

```
for i=1:5        % Your basic loop; i goes from 1 to 5
  p(i) = i^2;
end              % This is the end of the loop
```

This loop assigns the value `p = [1 4 9 16 25]`. The body of the `for` loop (i.e., the set of statements executed in the loop) is terminated by the `end` statement. Notice that `p` is created as a row vector. If we wanted it to be a column vector, we could build it as

```
for i=1:5        % Your basic loop; i goes from 1 to 5
  p(i,1) = i^2;  % p is a column vector
end              % This is the end of the loop
```

or

```
for i=1:5        % Your basic loop; i goes from 1 to 5
  p(i) = i^2;
end              % This is the end of the loop
p = p.';         % Transpose p into a column vector
```

using the transpose operator.

In a `for` loop, the default step is +1, but it is possible to use a different increment. For example, the loop

```
for i=1:2:5     % Loop over odd values of i
  q(i) = i;
  q(i+1) = -i;
end
```

assigns the values `q = [1 -1 3 -3 5 -5]`.

MATLAB also has `while` loops; here is a simple example:

```
while( x > 1 )
  x = x/2;
end
```

A `while` command executes the statements in the body of the loop while the loop condition is true. If $x = 5$ before the loop, then x will equal $\frac{5}{8}$ when the loop completes. The `break` statement can be used to terminate `for` loops or `while` loops.

Here are some examples of how conditionals are implemented; you see that it is quite standard.

```
if( x > 5 )          % A simple conditional
  z = z-1;
  y = MaxHeight;     % Body of this conditional has two statements
end

if( x >= x_min & x <= x_max )   % A more complicated conditional
  status = 1;
else                            % This conditional uses else
  status = 0;
end

if( x == 0 | x == 1)        % Another conditional using elseif
  flag = 1;
elseif( x < 0 & x ~= -1)    % Notice that elseif is ONE WORD
  flag = -1;
else
  flag = 0;
end
```

Notice that equals and not equals are `==` and `~=`, respectively. Logical "and" is `&` (ampersand), and logical "or" is `|` (vertical bar). The `end` command terminates both loops and conditionals.

Colon Operator

The colon operator, `:` , is one of MATLAB's handiest tools.[†] Let's consider a few examples of its use. First, the `for` loop,

[†]FORTRAN 90 has a similar colon operator

```
tau = 0.1;
for i=1:100
  time(i) = tau * i;
end
```

could be replaced with

```
tau = 0.1;
i=1:100;
time = tau * i;
```

In the latter, a vector `i=[1 2 ... 100]` is created. We may further abbreviate this to

```
tau = 0.1;
time = tau * (1:100);
```

In all three cases, the vector `time=[0.1 0.2 ... 10.0]` is created.

The colon operator is also useful for looping over rows or columns of a matrix. For example, the loop

```
[M,N] = size(A);  % Find dimensions of A
for i=1:M
  first(i) = A(i,1);
  last(i) = A(i,N);
end
```

copies the first and last columns of matrix `A` into the vectors `first` and `last`. The above can be replaced with

```
[M,N] = size(A);  % Find dimensions of A
first = A(:,1);
last = A(:,N);
```

Not only does using the colon operator abbreviate our code, but it also makes it run faster. The program becomes more efficient because we are explicitly executing a vector operation instead of performing an element-by-element calculation.

Input, Output, and Graphics

MATLAB has various types of input and output facilities. The `input` command prints a prompt to the screen and accepts input from the keyboard. Here is a simple example of its use:

```
x = input('Enter the value of x: ');
```

In this example, you can enter a scalar, a matrix, or any valid MATLAB expression.

The `disp` command may be used to display the value of a variable or to print a string of text:

Table 1.2: Selected MATLAB graphics functions.

`plot(x,y)`	Plot vector `y` versus vector `x`
`loglog(x,y)`, `semilogx(x,y)`, `semilogy(x,y)`	Plot vector `y` versus vector `x` using log or semilog scales
`polar(theta,rho)`	Polar plot
`contour(z)`	Contour plot of matrix `z`
`mesh(z)`	3-D wire-mesh plot of matrix `z`
`title('text')`,	Write a title on a plot
`xlabel('text')`, `ylabel('text')`	Write axis labels on a plot
`print`	Print graphics

```
M = [1, 2, 3; 4, 5, 6; 7, 8, 9];
disp('The value of M is ');
disp(M);
```

produces the output

```
The value of M is
     1     2     3
     4     5     6
     7     8     9
```

Formatted output is also available with the `fprintf` command,

```
fprintf('The values of x and y are %g and %g meters \n',x,y)
```

The values of the variables `x` and `y` are displayed in place of the `%g`'s. The `\n` at the end of the text string indicates a carriage return (new line). The MATLAB commands `load` and `save` can be used to read and write data files.

MATLAB has various graphics commands for creating xy plots, contour plots, and three-dimensional wire-mesh plots. Table 1.2 gives a list of a few of the basic graphics commands.

MATLAB Session

When you first enter the MATLAB environment, you are at the command level as indicated by the `>>` prompt (in the regular edition) or the `EDU>>` prompt (in the student edition). From the command line you can enter individual MATLAB commands. To end your MATLAB session, type `quit` or `exit`.

For programming, it is more convenient to enter a set of commands to be executed in a file. In the MATLAB terminology such a script of commands is called an *M-file*. Our programs and functions will all be M-files. You run an M-file by invoking its name (the name of the file) on the command line. Before running a program you need to tell MATLAB where to look for your M-files,

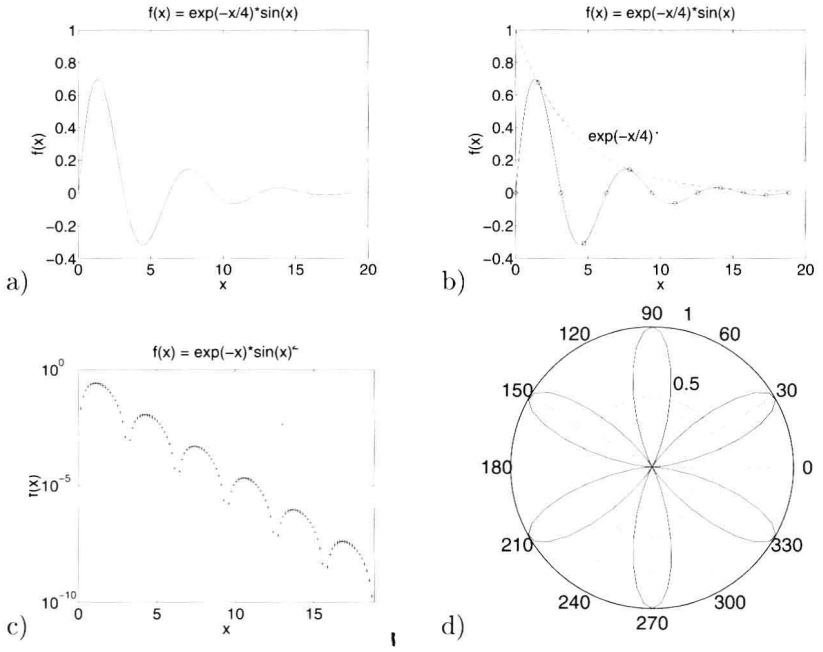


Figure 1.1: Samples of MATLAB plotting.

that is, the directory where your files are located. MATLAB searches for M-files in all locations specified in a list called the "path." Use the `path` command or the "Set path..." menu item to add your directories to MATLAB's path.

After MATLAB executes the commands in the M-file, it returns control to the command line. You can then enter individual commands (for example, to display the values of your variables). The interactive help may be used from the command line. For example,

```
EDU>>  help bessel
```

tells you about MATLAB's Bessel function routines. You can also get help on special characters (e.g., try `help &`).

EXERCISES

(Recommended exercises indicated by boldface numbers)

1. For the matrix `A=[1 2; 3 4]`, use compute MATLAB to find: (a) `A*A`; (b) `A.*A`; (c) `A^2`; (d) `A.^2`; (e) `A/A`; (f) `A./A`. [MATLAB]

2. Given the vectors `x` = [1 2 3... 10] and `y` = [1 4 9...100], plot them in MATLAB using: (a) `plot(x,y)`; (b) `plot(x,y,'+')`; (c) `plot(x,y,'-',x,y,'+')`; (d) `plot(x,y,'-', x(1:2:10), y(1:2:10), '+')`; (e) `semilogy(x,y)`; (f) `loglog(x,y,'+')`. [MATLAB]

3. Reproduce the plots shown in Figure 1.1. Try to be as accurate as possible in your reconstruction. [MATLAB]