Contents

PREFACE

This book is designed to support a one-semester course in numerical methods. It has been written for students who want to learn and apply numerical methods in order to solve problems in engineering and science. As such, the methods are motivated by problems rather than by mathematics. That said, sufficient theory is provided so that students come away with insight into the techniques and their shortcomings.

MATLAB© provides a great environment for such a course. Although other environments (e.g., Excel/VBA, Mathcad) or languages (e.g., Fortran 90, C++) could have been chosen, MATLAB presently offers a nice combination of handy programming features with powerful built-in numerical capabilities. On the one hand, its M-file programming environment allows students to implement moderately complicated algorithms in a structured and coherent fashion. On the other hand, its built-in, numerical capabilities empower students to solve more difficult problems without trying to "reinvent the wheel."

The basic content, organization, and pedagogy of the second edition are essentially preserved in the third edition. In particular, the conversational writing style is intentionally maintained in order to make the book easier to read. This book tries to speak directly to the reader and is designed in part to be a tool for self-teaching.

That said, this edition differs from the past edition in three major ways: (1) two new chapters, (2) several new sections, and (3) revised homework problems.

1. **New Chapters.** As shown in 1, I have developed two new chapters for this edition. Their inclusion was primarily motivated by my classroom experience. That is, they are included because they work well in the undergraduate numerical methods course I teach at Tufts. The students in that class typically represent all areas of engineering and range from sophomores to seniors with the majority at the junior level. In addition, we typically draw a few math and science majors. The two new chapters are:

   - **Eigenvalues.** When I first developed this book, I considered that eigenvalues might be deemed an "advanced" topic. I therefore presented the material on this topic at the end of the semester and covered it in the book as an appendix. This sequencing had the ancillary advantage that the subject could be partly motivated by the role of eigenvalues in the solution of linear systems of ODEs. In recent years, I have begun

     to move this material up to what I consider to be its more natural mathematical position at the end of the section on linear algebraic equations. By stressing applications (in particular, the use of eigenvalues to study vibrations), I have found that students respond very positively to the subject in this position. In addition, it allows me to return to the topic in subsequent chapters which serves to enhance the studentsâĂŹ appreciation of the topic.

   - **Fourier Analysis.** In past years, if time permitted, I also usually presented a lecture at the end of the semester on Fourier analysis. Over the past two years, I have begun presenting this material at its more natural position just after the topic of linear least squares. I motivate the subject matter by using the linear least-squares approach to fit sinusoids to data. Then, by stressing applications (again vibrations), I have found that the students readily absorb the topic and appreciate its value in engineering and science.

     It should be noted that both chapters are written in a modular fashion and could be skipped without detriment to the course's pedagogical arc. Therefore, if you choose, you can either omit them from your course or perhaps move them to the end of the semester. In any event, I would not have included them in the current edition if they did not represent an enhancement within my current experience in the classroom. In particular, based on my teaching evaluations, I find that the stronger, more motivated students actually see these topics as highlights. This is particularly true because MATLAB greatly facilitates their application and interpretation.

2. **New Content.** Beyond the new chapters, I have included new and enhanced sections on a number of topics. The primary additions include sections on animation (Chap. 3), Brent's method for root location (Chap. 6), *LU* factorization with pivoting (Chap. 8), *random numbers and Monte Carlo simulation* (Chap. 14), *adaptive quadrature* (Chap. 20),and *event termination of ODEs* (Chap. 23).

3. **New Homework Problems.** Most of the end-of-chapter problems have been modified, and a variety of new problems have been added. In particular, an effort has been made to include several new problems for each chapter that are more challenging and difficult than the problems in the previous edition.

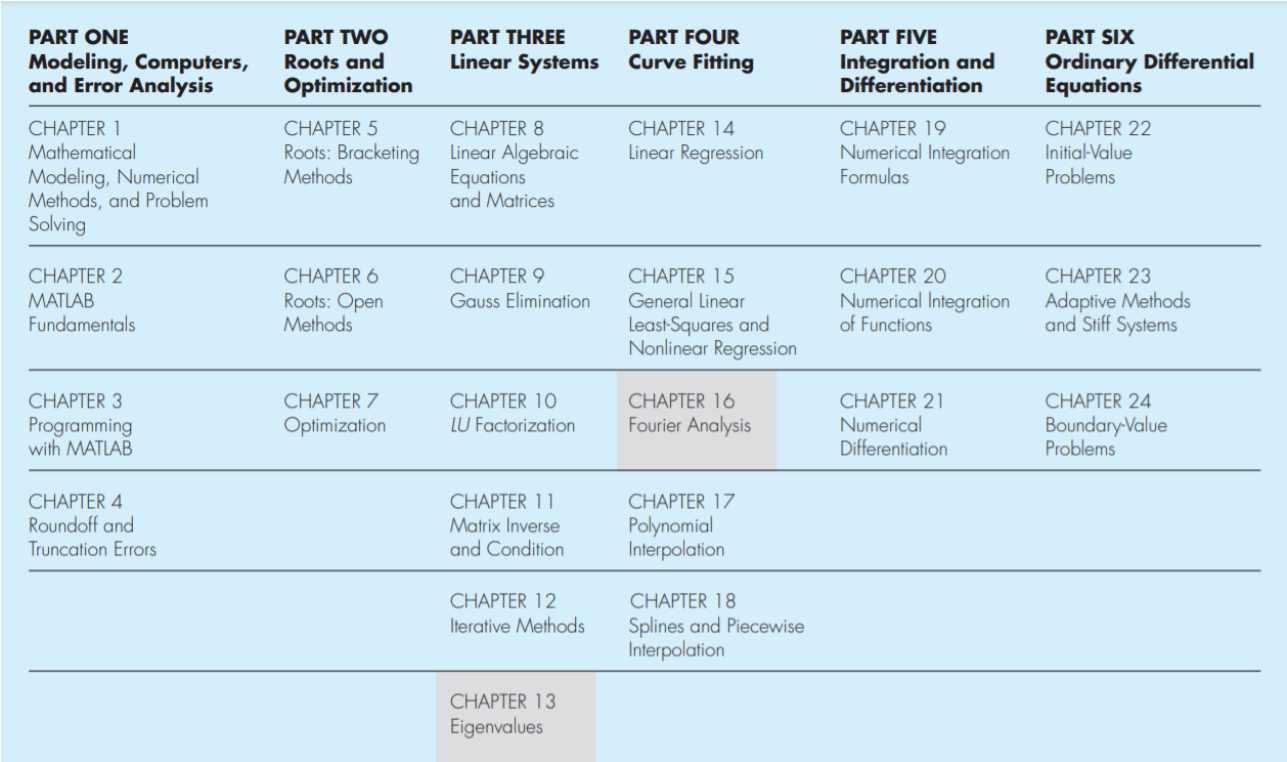


Figure 1: An outline of this edition. The shaded areas represent new material. In addition, several of the original chapters have been supplemented with new topics.

Aside from the new material and problems, the third edition is very similar to the second. In particular, I have endeavored to maintain most of the features contributing to its pedagogical effectiveness including extensive use of worked examples and engineering and scientific applications. As with the previous edition, I have made a concerted effort to make this book as "student-friendly" as possible. Thus, IâĂŹve tried to keep my explanations straightforward and practical. Although my primary intent is to empower students by providing them with a sound introduction to numerical problem solving, I have the ancillary objective of making this introduction exciting and pleasurable. I believe that motivated students who enjoy engineering and science, problem solving, mathemati cs and yes programming, will ultimately make better professionals. If my book fosters enthusiasm and appreciation for these subjects, I will consider the effort a success.

Acknowledgments. Several members of the McGraw-Hill team have contributed to this project. Special thanks are due to Lorraine Buczek, and Bill Stenquist, and Melissa Leick for their encouragement, support, and direction. Ruma Khurana of MPS Limited, a Macmillan Company also did an outstanding job in the bookâĂŹs final production phase. Last, but not least, Beatrice Sussman once again demonstrated why she is the best copyeditor in the business. During the course of this project, the folks at The MathWorks, Inc., have truly demonstrated their overall excellence as well as their strong commitment to engineering and science education. In particular, Courtney Esposito and Naomi Fernandes of The MathWorks, Inc., Book Program have been especially helpful. The generosity of the Berger family, and in particular Fred Berger, has provided me with the opportunity to work on creative projects such as this book dealing with computing and engineering. In addition, my colleagues in the School of Engineering at Tufts, notably Masoud Sanayei, Lew Edgers, Vince Manno, Luis Dorfmann, Rob White, Linda Abriola, and Laurie Baise, have been very supportive and helpful. Significant suggestions were also given by a number of colleagues. In particular, Dave Clough (University of ColoradoâĂŞBoulder), and Mike Gustafson (Duke University) provided valuable ideas and suggestions. In addition, a number of reviewers provided useful feedback and advice including Karen Dow Ambtman (University of Alberta), Jalal Behzadi (Shahid Chamran University), Eric Cochran (Iowa State University), Frederic Gibou (University of California at Santa Barbara), Jane Grande-Allen (Rice University), Raphael Haftka (University of Florida), Scott Hendricks (Virginia Tech University), Ming Huang (University of San Diego), Oleg Igoshin (Rice University), David Jack (Baylor University), Clare McCabe (Vanderbilt University), Eckart Meiburg (University of California at Santa Barbara), Luis Ricardez (University of Waterloo), James Rottman (University of California, San Diego), Bingjing Su (University of Cincinnati), Chin-An Tan (Wayne State University), Joseph Tipton (The University of Evansville), Marion W. Vance (Arizona State University), Jonathan Vande Geest (University of Arizona), and Leah J. Walker (Arkansas State University). It should be stressed that although I received useful advice from the aforementioned individuals, I am responsible for any inaccuracies or mistakes you may find in this book. Please contact me via e-mail if you should detect any errors. Finally, I want to thank my family, and in particular my wife, Cynthia, for the love, patience, and support they have provided through the time IâĂŹve spent on this project.

Steven C. Chapra
Tufts University
Medford, Massachusetts
steven.chapra@tufts.edu

PEDAGOGICAL TOOLS

Theory Presented as It Informs Key Concepts. The text is intended for Numerical Methods users, not developers. Therefore, theory is not included for âĂIJtheoryâĂŹs sake,âĂİ for example no proofs. Theory is included as it informs key concepts such as the Taylor series, convergence, condition, etc. Hence, the student is shown how the theory connects with practical issues in problem solving.

Introductory MATLAB Material. The text includes two introductory chapters on how to use MATLAB. Chapter 2 shows students how to perform computations and create graphs in MATLABâĂŹs standard command mode. Chapter 3 provides a primer on developing numerical programs via MATLAB M-file functions. Thus, the text provides students with the means to develop their own numerical algorithms as well as to tap into MATLABâĂŹs powerful built-in routines.

Algorithms Presented Using MATLAB M-files. Instead of using pseudocode, this book presents algorithms as well-structured MATLAB M-files. Aside from being useful computer programs, these provide students with models for their own M-files that they will develop as homework exercises.

Worked Examples and Case Studies. Extensive worked examples are laid out in detail so that students can clearly follow the steps in each numerical computation. The case studies consist of engineering and science applications which are more complex and richer than the worked examples. They are placed at the ends of selected chapters with the intention of (1) illustrating the nuances of the methods, and (2) showing more realistically how the methods along with MATLAB are applied for problem solving.

Problem Sets. The text includes a wide variety of problems. Many are drawn from engineering and scientific disciplines. Others are used to illustrate numerical techniques and theoretical concepts. Problems include those that can be solved with a pocket calculator as well as others that require computer solution with MATLAB.

Useful Appendices and Indexes. Appendix A contains MATLAB commands, and Appendix B contains M-file functions.

Textbook Website. A text-specific website is available at www.mhhe.com/chapra. Resources include the text images in PowerPoint, M-files, and additional MATLAB resources.

Part I

Modeling, Computers and Error Analysis

0.1. MOTIVATION

What are numerical methods and why should you study them?

Numerical methods are techniques by which mathematical problems are formulated so that they can be solved with arithmetic and logical operations. Because digital computers excel at performing such operations, numerical methods are sometimes referred to as computer mathematics.

In the preâĂŞcomputer era, the time and drudgery of implementing such calculations seriously limited their practical use. However, with the advent of fast, inexpensive digital computers, the role of numerical methods in engineering and scientific problem solving has exploded. Because they figure so prominently in much of our work, I believe that numerical methods should be a part of every engineerâĂŹs and scientistâĂŹs basic education. Just as we all must have solid foundations in the other areas of mathematics and science, we should also have a fundamental understanding of numerical methods. In particular, we should have a solid appreciation of both their capabilities and their limitations. Beyond contributing to your overall education, there are several additional reasons why you should study numerical methods:

1. Numerical methods greatly expand the types of problems you can address. They are capable of handling large systems of equations, nonlinearities, and complicated geometries that are not uncommon in engineering and science and that are often impossible to solve analytically with standard calculus. As such, they greatly enhance your problem-solving skills.

2. Numerical methods allow you to use âĂIJcannedâĂİ software with insight. During your career, you will invariably have occasion to use commercially available prepackaged computer programs that involve numerical methods. The intelligent use of these programs is greatly enhanced by an understanding of the basic theory underlying the methods. In the absence of such understanding, you will be left to treat such packages as âĂIJblack boxesâĂİ with little critical insight into their inner workings or the validity of the results they produce.

3. Many problems cannot be approached using canned programs. If you are conversant with numerical methods, and are adept at computer programming, you can design your own programs to solve problems without having to buy or commission expensive software.

4. Numerical methods are an efficient vehicle for learning to use computers. Because numerical methods are expressly designed for computer implementation, they are ideal for illustrating the computerâĂŹs powers and limitations. When you successfully implement numerical methods on a computer, and then apply them to solve otherwise intractable problems, you will be provided with a dramatic demonstration of how computers can serve your professional development. At the same time, you will also learn to acknowledge and control the errors of approximation that are part and parcel of large-scale numerical calculations.

5. Numerical methods provide a vehicle for you to reinforce your understanding of mathematics. Because one function of numerical methods is to reduce higher mathematics to basic arithmetic operations, they get at the âĂIJnuts and boltsâĂİ of some otherwise obscure topics. Enhanced understanding and insight can result from this alternative perspective.

With these reasons as motivation, we can now set out to understand how numerical methods and digital computers work in tandem to generate reliable solutions to mathematical problems. The remainder of this book is devoted to this task.

0.2. PART ORGANIZATION

This book is divided into six parts. The latter five parts focus on the major areas of numerical methods. Although it might be tempting to jump right into this material, *Part One* consists of four chapters dealing with essential background material.

Chapter 1 provides a concrete example of how a numerical method can be employed to solve a real problem. To do this, we develop a *mathematical model* of a free-falling bungee jumper. The model, which is based on NewtonâĂŹs second law, results in an ordinary differential equation. After first using calculus to develop a closed-form solution, we then show how a comparable solution can be generated with a simple numerical method. We end the chapter with an overview of the major areas of numerical methods that we cover in Parts Two through Six.

Chapters 2 and 3 provide an introduction to the MATLAB© software environment. *Chapter 2* deals with the standard way of operating MATLAB by entering commands one at a time in the so-called calculator, or command, mode. This interactive mode provides a straightforward means to orient you to the environment and illustrates how it is used for common operations such as performing calculations and creating plots.

Chapter 3 shows how MATLABâĂŹs programming mode provides a vehicle for assembling individual commands into algorithms. Thus, our intent is to illustrate how MATLAB serves as a convenient programming environment to develop your own software.

Chapter 4 deals with the important topic of error analysis, which must be understood for the effective use of numerical methods. The first part of the chapter focuses on the *roundoff errors* that result because digital computers cannot

represent some quantities exactly. The latter part addresses *truncation errors* that arise from using an approximation in place of an exact mathematical procedure.

Chapter 1

Modeling, Numerical Methods, and Problem Solving

CHAPTER OBJECTIVES

The primary objective of this chapter is to provide you with a concrete idea of what numerical methods are and how they relate to engineering and scientific problem solving. Specific objectives and topics covered are

- Learning how mathematical models can be formulated on the basis of scientific principles to simulate the behavior of a simple physical system.
- Understanding how numerical methods afford a means to generate solutions in a manner that can be implemented on a digital computer.
- Understanding the different types of conservation laws that lie beneath the models used in the various engineering disciplines and appreciating the difference between steady-state and dynamic solutions of these models.
- Learning about the different types of numerical methods we will cover in this book.

YOU'VE GOT A PROBLEM

Suppose that a bungee-jumping company hires you. YouâĂŹre given the task of predicting the velocity of a jumper (Fig. 1.1) as a function of time during the free-fall part of the jump. This information will be used as part of a larger analysis to determine the length and required strength of the bungee cord for jumpers of different mass. You know from your studies of physics that the acceleration should be equal to the ratio of the force to the mass (NewtonâĂŹs second law). Based on this insight and your knowledge of physics and fluid mechanics, you develop the following mathematical model for the rate of change of velocity with respect to time,

Figure 1.1: Forces acting on a free-falling bungee jumper

$\frac{dv}{dt} = g - \frac{c_d}{m}v^2$ where v = downward vertical velocity (m/s), t = time (s), g = the acceleration due to gravity ($\cong$ 9.81 m/s2), $c_d = a$ lumped drag coefficient (kg/m), and m = the jumperâĂŹs mass (kg). The drag coefficient is called âĂIJlumpedâĂİ because its magnitude depends on factors such as the jumperâĂŹs area and the fluid density (see Sec 1.4).

Because this is a differential equation, you know that calculus might be used to obtain an analytical or exact solution for v as a function of t. However, in the following pages, we will illustrate an alternative solution approach. This will involve developing a computeroriented numerical or approximate solution.

Aside from showing you how the computer can be used to solve this particular problem, our more general objective will be to illustrate (a) what numerical methods are and (b) how they figure in engineering and scientific problem solving. In so doing, we will also show how mathematical models figure prominently in the way engineers and scientists use numerical methods in their work.

1.1. A SIMPLE MATHEMATICAL MODEL

A *mathematical model* can be broadly defined as a formulation or equation that expresses the essential features of a physical system or process in mathematical terms. In a very general sense, it can be represented as a functional relationship of the form

$$\overset{Dependent}{variable} = f\left(\overset{independent}{variables}, parameters, \overset{forcing}{functions}\right) \tag{1.1}$$

where the *dependent variable* is a characteristic that typically reflects the behavior or state of the system; the independent variables are usually dimensions, such as time and space, along which the systemâĂŹs behavior is being determined; the parameters are reflective of the system's properties or composition; and the forcing functions are external influences acting upon it.

The actual mathematical expression of Eq. (1.1) can range from a simple algebraic relationship to large complicated sets of differential equations. For example, on the basis of his observations, Newton formulated his second law of motion, which states that the time rate of change of momentum of a body is equal to the resultant force acting on it. The mathematical expression, or model, of the second law is the well-known equation

$$F = ma \tag{1.2}$$

where F is the net force acting on the body $(N, orkgm/s^2)$, m is the mass of the object (kg), and a is its acceleration (m/s^2).

The second law can be recast in the format of Eq. (1.1) by merely dividing both sides by m to give

$$a = \frac{F}{m} \tag{1.3}$$

where a is the dependent variable reflecting the system's behavior, F is the forcing function, and m is a parameter. Note that for this simple case there is no independent variable because we are not yet predicting how acceleration varies in time or space.

- It describes a natural process or system in mathematical terms.
- It represents an idealization and simplification of reality. That is, the model ignores negligible details of the natural process and focuses on its essential manifestations. Thus, the second law does not include the effects of relativity that are of minimal importance when applied to objects and forces that interact on or about the earthâĂŹs surface at velocities and on scales visible to humans.
- Finally, it yields reproducible results and, consequently, can be used for predictive purposes. For example, if the force on an object and its mass are known, Eq. (1.3) can be used to compute acceleration.

Because of its simple algebraic form, the solution of Eq. (1.2) was obtained easily. However, other mathematical models of physical phenomena may be much more complex, and either cannot be solved exactly or require more sophisticated mathematical techniques than simple algebra for their solution. To illustrate a more complex model of this kind, NewtonâĂŹs second law can be used to determine the terminal velocity of a free-falling body near the earthâĂŹs surface. Our falling body will be a bungee jumper (Fig. 1.1). For this case, a model can be derived by expressing the acceleration as the time rate of change of the velocity (dv/dt) and substituting it into Eq. (1.3) to yield

$$\frac{dv}{dt} = \frac{F}{m} \tag{1.4}$$

where v is velocity (in meters per second). Thus, the rate of change of the velocity is equal to the net force acting on the body normalized to its mass. If the net force is positive, the object will accelerate. If it is negative, the object will decelerate. If the net force is zero, the object's velocity will remain at a constant level.

Next, we will express the net force in terms of measurable variables and parameters. For a body falling within the vicinity of the earth, the net force is composed of two opposing forces: the downward pull of gravity F_D and the upward force of air resistance F_U (Fig. 1.1):

$$F = F_D + F_U \tag{1.5}$$

If force in the downward direction is assigned a positive sign, the second law can be used to formulate the force due to gravity as

$$F_D = mg \tag{1.6}$$

where g is the acceleration due to gravity $(9.81 m/s^2)$.

Air resistance can be formulated in a variety of ways. Knowledge from the science of fluid mechanics suggests that a good first approximation would be to assume that it is proportional to the square of the velocity,

$$F_U = -c_d v^2 \tag{1.7}$$

where c_d is a proportionality constant called the *lumped drag coefficient* (kg/m). Thus, the greater the fall velocity, the greater the upward force due to air resistance. The parameter c_d accounts for properties of the falling object, such as shape or surface roughness, that affect air resistance. For the present case, c_d might be a function of the type of clothing or the orientation used by the jumper during free fall. The net force is the difference between the downward and upward force. Therefore, Eqs. (1.4) through (1.7) can be combined to yield

$$\frac{dv}{dt} = g - \frac{C_d}{m} v^2 \tag{1.8}$$

Equation (1.8) is a model that relates the acceleration of a falling object to the forces acting on it. It is a *differential equation* because it is written in terms of the differential rate of change (dv/dt) of the variable that we are interested in predicting. However, in contrast to the solution of Newton's second law in Eq. (1.3), the exact solution of Eq. (1.8) for the velocity of the jumper cannot be obtained using simple algebraic manipulation. Rather, more advanced techniques such as those of calculus must be applied to obtain an exact or analytical solution. For example, if the jumper is initially at rest $(v = 0 at t = 0)$, calculus can be used to solve Eq. (1.8) for

$$v(t) = \sqrt{\frac{gm}{c_d}} tanh\left(\sqrt{\frac{gc_d}{m}} t\right) \tag{1.9}$$

where tanh is the hyperbolic tangent that can be either computed directly[1] or via the more elementary exponential function as in

$$tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.10}$$

Note that Eq. (1.9) is cast in the general form of Eq. (1.1) where $v(t)$ is the dependent variable, t is the independent variable, c_d and m are parameters, and g is the forcing function.

EXAMPLE 1.1. ANALYTICAL SOLUTION TO THE BUNGEE JUMPER PROBLEM

Problem Statement. A bungee jumper with a mass of 68.1 kg leaps from a stationary hot air balloon. Use Eq. (1.9) to compute velocity for the first 12 s of free fall. Also determine the terminal velocity that will be attained for an infinitely long cord (or alternatively, the jumpmaster is having a particularly bad day!). Use a drag coefficient of 0.25 kg/m.

Solution. Inserting the parameters into Eq. (1.9) yields

$$v(t) = \sqrt{\frac{9.81(68.1)}{0.25}} tanh\left(\sqrt{\frac{9.81(0.25)}{68.1}} t\right) = 51.6938 tanh(0.18977 t)$$

[1] MATLAB allows direct calculation of the hyperbolic tangent via the built-in function $tanh(x)$.

which can be used to compute

t, s	v, m/s
0	0
2	18.7292
4	33.1118
6	42.0762
8	46.9575
10	49.4214
12	50.6175
∞	51.6938

According to the model, the jumper accelerates rapidly (Fig. 1.2). A velocity of 49.4214 m/s (about 110 mi/hr) is attained after 10 s. Note also that after a sufficiently long

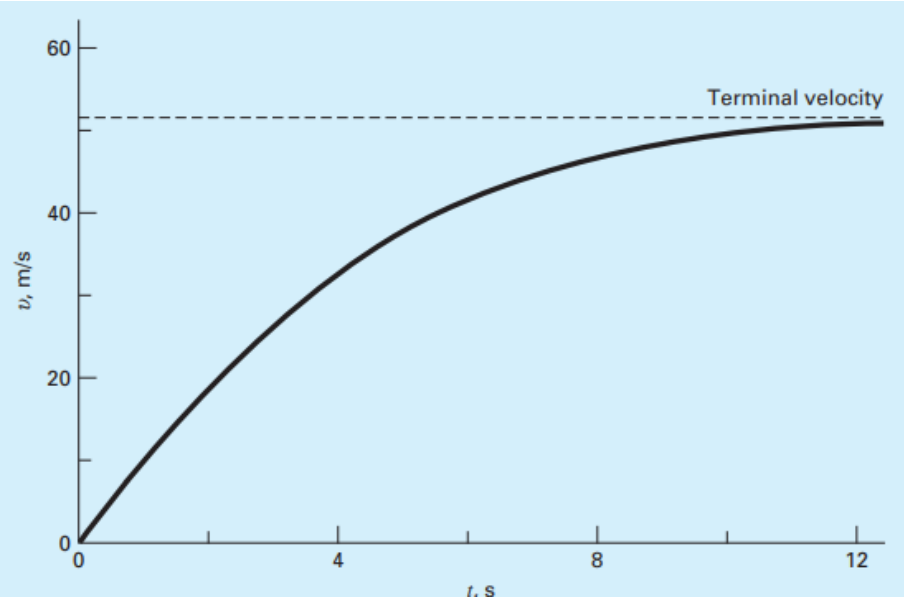


Figure 1.2: The analytical solution for the bungee jumper problem as computed in Example 1.1. Velocity increases with time and asymptotically approaches a terminal velocity.

time, a constant velocity, called the terminal velocity, of 51.6983 m/s (115.6 mi/hr) is reached. This velocity is constant because, eventually, the force of gravity will be in balance with the air resistance. Thus, the net force is zero and acceleration has ceased.

Equation (1.9) is called an analytical or closed-form solution because it exactly satisfies the original differential equation. Unfortunately, there are many mathematical models that cannot be solved exactly. In many of these cases, the only alternative is to develop a numerical solution that approximates the exact solution. *Numerical methods* are those in which the mathematical problem is reformulated so it can be solved by arithmetic operations. This can be illustrated for Eq. (1.8) by realizing that the time rate of change of velocity can be approximated by (Fig. 1.3):

$$\frac{dv}{dt} \cong \frac{\Delta v}{\Delta t} = \frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i} \tag{1.11}$$

where Δv and Δt are differences in velocity and time computed over finite intervals, $v(t_i)$ is velocity at an initial time t_i, and $v(t_{i+1})$ is velocity at some later time (t_{i+1}). Note that $dv/dt \cong \Delta v/\Delta t$ is approximate because Δt is finite. Remember from calculus that

$$\frac{dv}{dt} = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t}$$

Equation (1.11) represents the reverse process.

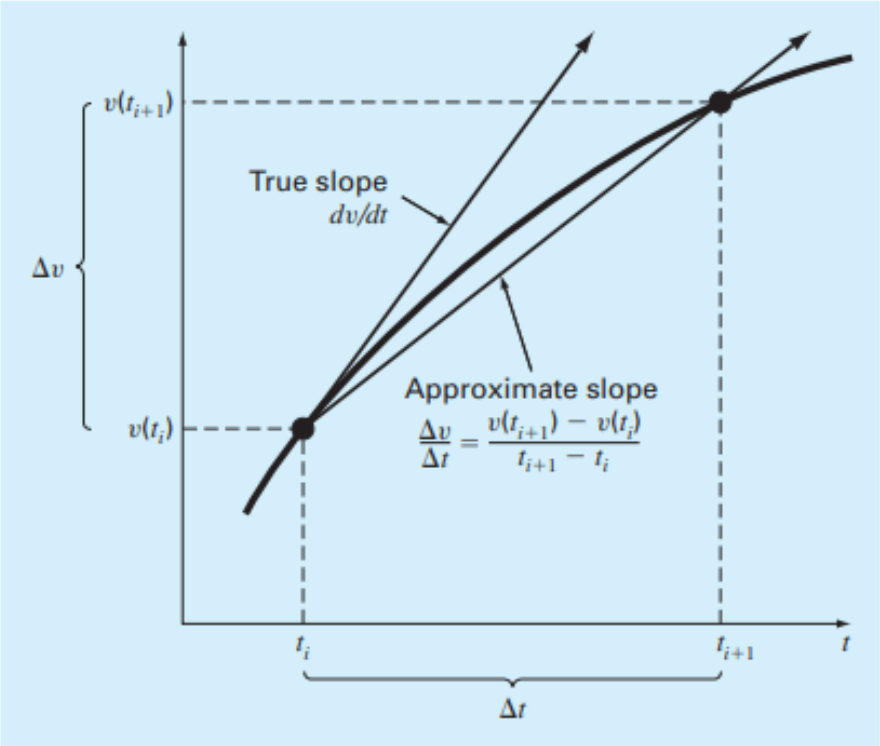


Figure 1.3: The use of a finite difference to approximate the first derivative of v with respect to t.

```
>> whos
  Name       Size          Bytes     Class
  A          3x3              72         double array
  a          1x5              40         double array
  ans          1x1             8           double array
  b          5x1              40         double array
  x          1x1              16         double array (complex)
  Grand total is 21 elements using 176 bytes
```

Chapter 2

Roundoff and Truncation Errors

CHAPTER OBJECTIVES

The primary objective of this chapter is to acquaint you with the major sources of errors involved in numerical methods. Specific objectives and topics covered are

- Understanding the distinction between accuracy and precision.
- Learning how to quantify error.
- Learning how error estimates can be used to decide when to terminate an iterative calculation.
- Understanding how roundoff errors occur because digital computers have a limited ability to represent numbers.
- Understanding why floating-point numbers have limits on their range and precision.
- Recognizing that truncation errors occur when exact mathematical formulations are represented by approximations.
- Knowing how to use the Taylor series to estimate truncation errors.
- Understanding how to write forward, backward, and centered finite-difference approximations of first and second derivatives.
- Recognizing that efforts to minimize truncation errors can sometimes increase roundoff errors.

YOU'VE GOT A PROBLEM

In Chap. 1 you developed a numerical model for the velocity of a bungee jumper. To solve the problem with a computer, you had to approximate the derivative of velocity with a finite difference.

$$\frac{dv}{dt} \cong \frac{|\,\Delta v}{\Delta t} = \frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i}$$

Thus, the resulting solution is not exact — that is, it has error.

In addition, the computer you use to obtain the solution is also an imperfect tool. Because it is a digital device, the computer is limited in its ability to represent the magnitudes and precision of numbers. Consequently, the machine itself yields results that contain error.

So both your mathematical approximation and your digital computer cause your resulting model prediction to be uncertain. Your problem is: How do you deal with such uncertainty? In particular, is it possible to understand, quantify and control such errors in order to obtain acceptable results? This chapter introduces you to some approaches and concepts that engineers and scientists use to deal with this dilemma.

2.1. ERRORS

Engineers and scientists constantly find themselves having to accomplish objectives based on uncertain information. Although perfection is a laudable goal, it is rarely if ever attained. For example, despite the fact that the model developed from Newton's second law is an excellent approximation, it would never in practice exactly predict the jumper's fall. A variety of factors such as winds and slight variations in air resistance would result in deviations from the prediction. If these deviations are systematically high or low, then we might need to develop a new model. However, if they are randomly distributed and tightly grouped around the prediction, then the deviations might be considered negligible and the model deemed adequate. Numerical approximations also introduce similar discrepancies into the analysis.

This chapter covers basic topics related to the identification, quantification, and minimization of these errors. General information concerned with the quantification of error is reviewed in this section. This is followed by Sections 4.2 and 4.3, dealing with the two major forms of numerical error: roundoff error (due to computer approximations) and truncation error (due to mathematical approximations). We also describe how strategies to reduce truncation error sometimes increase roundoff. Finally, we briefly discuss errors not directly connected with the numerical methods themselves. These include blunders, model errors, and data uncertainty.

2.1.1. Accuracy and Precision

The errors associated with both calculations and measurements can be characterized with regard to their accuracy and precision. Accuracy refers to how closely a computed or measured value agrees with the true value. Precision refers to how closely individual computed or measured values agree with each other.

These concepts can be illustrated graphically using an analogy from target practice. The bullet holes on each target in Fig. 4.1 can be thought of as the predictions of a numerical technique, whereas the bull's-eye represents the truth. Inaccuracy (also called bias) is defined as systematic deviation from the truth. Thus, although the shots in Fig. 4.1c are more tightly grouped than in Fig. 4.1a, the two cases are equally biased because they are both centered on the upper left quadrant of the target. Imprecision (also called uncertainty), on the other hand, refers to the magnitude of the scatter. Therefore, although Fig. 4.1b and d are equally accurate (i.e., centered on the bull's-eye), the latter is more precise because the shots are tightly grouped.

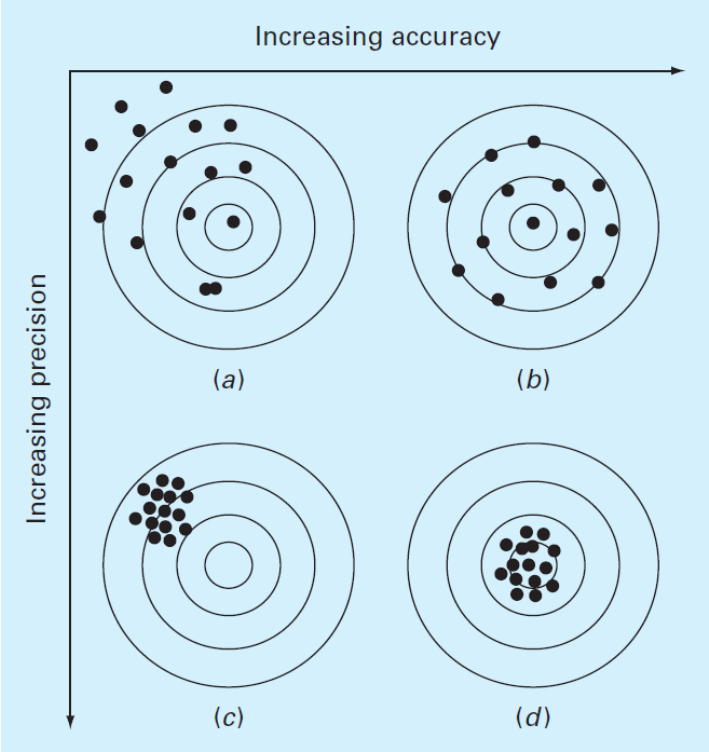


Figure 2.1: An example from marksmanship illustrating the concepts of accuracy and precision: (*a*) inaccurate and imprecise, (*b*) accurate and imprecise, (*c*) inaccurate and precise, and (*d*) accurate and precise.

Numerical methods should be sufficiently accurate or unbiased to meet the requirements of a particular problem. They also should be precise enough for adequate design. In this book, we will use the collective term *error* to represent both the inaccuracy and imprecision of our predictions.

2.1.2. Error Definitions

Numerical errors arise from the use of approximations to represent exact mathematical operations and quantities. For such errors, the relationship between the exact, or true, result and the approximation can be formulated as

$$\text{True value} = \text{approximation} + \text{error} \tag{4.1}$$

By rearranging Eq. (4.1), we find that the numerical error is equal to the discrepancy between the truth and the approximation, as in

$$E_t = \text{true value - approximation} \tag{4.2}$$

where E_t is used to designate the exact value of the error. The subscript t is included to designate that this is the "true" error. This is in contrast to other cases, as described shortly, where an "approximate" estimate of the error must be employed. Note that the true error is commonly expressed as an absolute value and referred to as the *absolute error*.

A shortcoming of this definition is that it takes no account of the order of magnitude of the value under examination. For example, an error of a centimeter is much more significant if we are measuring a rivet than a bridge. One way to account for the magnitudes of the quantities being evaluated is to normalize the error to the true value, as in

$$\text{True fractional relative error} = \frac{\text{true value} - \text{approximation}}{\text{true value}}$$

The relative error can also be multiplied by 100% to express it as

$$\varepsilon_t = \frac{\text{true value} - \text{approximation}}{\text{true value}} 100\% \tag{4.3}$$

where ε_t designates the true percent relative error.

For example, suppose that you have the task of measuring the lengths of a bridge and a rivet and come up with 9999 and 9 cm, respectively. If the true values are 10,000 and 10 cm, respectively, the error in both cases is 1 cm. However, their percent relative errors can be computed using Eq. (4.3) as 0.01% and 10%, respectively. Thus, although both measurements have an absolute error of 1 cm, the relative error for the rivet is much greater. We would probably conclude that we have done an adequate job of measuring the bridge, whereas our estimate for the rivet leaves something to be desired.

Notice that for Eqs. (4.2) and (4.3), E and ε are subscripted with a t to signify that the error is based on the true value. For the example of the rivet and the bridge, we were provided with this value. However, in actual situations such information is rarely available. For numerical methods, the true value will only be known when we deal with functions that can be solved analytically. Such will typically be the case when we investigate the theoretical behavior of a particular technique for simple systems. However, in real-world applications, we will obviously not know the true answer *a priori*. For these situations, an alternative is to normalize the error using the best available estimate of the true value — that is, to the approximation itself, as in

$$\varepsilon_a = \frac{\text{approximate error}}{\text{approximation}} 100\% \tag{4.4}$$

where the subscript a signifies that the error is normalized to an approximate value. Note also that for real-world applications, Eq. (4.2) cannot be used to calculate the error term in the numerator of Eq. (4.4). One of the challenges of numerical methods is to determine error estimates in the absence of knowledge regarding the true value. For example, certain numerical methods use *iteration* to compute answers. In such cases, a present approximation is made on the basis of a previous approximation. This process is performed repeatedly, or iteratively, to successively compute (hopefully) better and better approximations. For such cases, the error is often estimated as the difference between the previous and present approximations. Thus, percent relative error is determined according to

$$\varepsilon_a = \frac{\text{present approximation} - \text{previous approximation}}{\text{present approximation}} 100\% \tag{4.5}$$

This and other approaches for expressing errors is elaborated on in subsequent chapters.

The signs of Eqs. (4.2) through (4.5) may be either positive or negative. If the approximation is greater than the true value (or the previous approximation is greater than the current approximation), the error is negative; if the approximation is less than the true value, the error is positive. Also, for Eqs. (4.3) to (4.5), the denominator may be less than zero, which can also lead to a negative error. Often, when performing computations, we may not be concerned with the sign of the error but are interested in whether the absolute value of the percent relative error is lower than a prespecified tolerance ε_s. Therefore, it is often useful to employ the absolute value of Eq. (4.5). For such cases, the computation is repeated until

$$|\varepsilon_a| < \varepsilon_s \tag{4.6}$$

This relationship is referred to as a *stopping criterion*. If it is satisfied, our result is assumed to be within the prespecified acceptable level ε_s . Note that for the remainder of this text, we almost always employ absolute values when using relative errors.

It is also convenient to relate these errors to the number of significant figures in the approximation. It can be shown (Scarborough, 1966) that if the following criterion is met, we can be assured that the result is correct to *at least n* significant figures.

$$\varepsilon_s = (0.5 \times 10^{2-n})\% \tag{4.7}$$

Example 2.1. Error Estimates for Iterative Methods

Problem Statement. In mathematics, functions can often be represented by infinite series. For example, the exponential function can be computed using

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} \tag{E4.1.1}$$

Thus, as more terms are added in sequence, the approximation becomes a better and better estimate of the true value of e^x. Equation (E4.1.1) is called a *Maclaurin series expansion*.

Starting with the simplest version, $e^x = 1$, add terms one at a time in order to estimate $e^{0.5}$. After each new term is added, compute the true and approximate percent relative errors with Eqs. (4.3) and (4.5), respectively. Note that the true value is $e^{0.5} = 1.648721\ldots$ Add terms until the absolute value of the approximate error estimate ε_a falls below a prespecified error criterion ε_s conforming to three significant figures.

Solution. First, Eq. (4.7) can be employed to determine the error criterion that ensures a result that is correct to at least three significant figures:

$$\varepsilon_s = (0.5 \times 10^{2-3})\% = 0.05\%$$

Thus, we will add terms to the series until ε_a falls below this level.

The first estimate is simply equal to Eq. (E4.1.1) with a single term. Thus, the first estimate is equal to 1. The second estimate is then generated by adding the second term as in

$$e^x = 1 + x$$

or for $x = 0.5$

$$e^{0.5} = 1 + 0.5 = 1.5$$

This represents a true percent relative error of [Eq. (4.3)]

$$\varepsilon_t = \left| \frac{1.648721 - 1.5}{1.648721} \right| \times 100\% = 9.02\%$$

Equation (4.5) can be used to determine an approximate estimate of the error, as in

$$\varepsilon_a = \left| \frac{1.5 - 1}{1.5} \right| \times 100\% = 33.3\%$$

Because ε_a is not less than the required value of ε_s, we would continue the computation by adding another term, $x^2/2!$, and repeating the error calculations. The process is continued until $|\varepsilon_a| < \varepsilon_s$. The entire computation can be summarized as

Terms	Result	ε_t, %	ε_a, %
1	1	39.3	
2	1.5	9.02	33.3
3	1.625	1.44	7.69
4	1.645833333	0.175	1.27
5	1.648437500	0.0172	0.158
6	1.648697917	0.00142	0.0158

Thus, after six terms are included, the approximate error falls below $\varepsilon_s = 0.05\%$, and the computation is terminated. However, notice that, rather than three significant figures, the result is accurate to five! This is because, for this case, both Eqs. (4.5) and (4.7) are conservative. That is, they ensure that the result is at least as good as they specify. Although, this is not always the case for Eq. (4.5), it is true most of the time. ■

2.1.3. Computer Algorithm for Iterative Calculations

Many of the numerical methods described in the remainder of this text involve iterative calculations of the sort illustrated in Example 4.1. These all entail solving a mathematical problem by computing successive approximations to the solution starting from an initial guess.

The computer implementation of such iterative solutions involves loops. As we saw in Sec. 3.3.2, these come in two basic flavors: count-controlled and decision loops. Most iterative solutions use decision loops. Thus, rather than employing a pre-specified number of iterations, the process typically is repeated until an approximate error estimate falls below a stopping criterion as in Example 4.1.

To do this for the same problem as Example 4.1, the series expansion can be expressed as

$$e^x \cong \sum_{i=0}^{n} \frac{x^n}{n!}$$

An M-file to implement this formula is shown in Fig. 4.2. The function is passed the value to be evaluated (`x`) along with a stopping error criterion (`es`) and a maximum allowable number of iterations (`maxit`). If the user omits either of the latter two parameters, the function assigns default values.

```
function [fx,ea,iter] = IterMeth(x,es,maxit)
% Maclaurin series of exponential function
%   [fx,ea,iter] = IterMeth(x,es,maxit)
% input:
%   x = value at which series evaluated
%   es = stopping criterion (default = 0.0001)
%   maxit = maximum iterations (default = 50)
% output:
%   fx = estimated value
%   ea = approximate relative error (%)
%   iter = number of iterations

% defaults:
if nargin<2|isempty(es),es=0.0001;end
if nargin<3|isempty(maxit),maxit=50;end
% initialization
iter = 1; sol = 1; ea = 100;
% iterative calculation
while (1)
  solold = sol;
  sol = sol + x ^ iter / factorial(iter);
  iter = iter + 1;
  if sol~=0
    ea=abs((sol - solold)/sol)*100;
  end
  if ea<=es | iter>=maxit,break,end
end
fx = sol;
end
```

Figure 2.2: An M-file to solve an iterative calculation. This example is set up to evaluate the Maclaurin series expansion for ex as described in Example 4.1.

The function then initializes three variables: *(a)* `iter`, which keeps track of the number of iterations, *(b)* `sol`, which holds the current estimate of the solution, and *(c)* a variable, `ea`, which holds the approximate percent relative error. Note that *ea* is initially set to a value of 100 to ensure that the loop executes at least once.

These initializations are followed by a decision loop that actually implements the iterative calculation. Prior to generating a new solution, the previous value, `sol`, is first assigned to `solold`. Then a new value of `sol` is computed and the iteration counter is incremented. If the new value of `sol` is nonzero, the percent relative error, `ea`, is determined. The stopping criteria are then tested. If both are false, the loop repeats. If either is true, the loop terminates and the final solution is sent back to the function call.

When the M-file is implemented, it generates an estimate for the exponential function which is returned along with the approximate error and the number of iterations. For example, e^1 can be evaluated as

```
» format long
» [approxval, ea, iter] = IterMeth(1,1e-6,100)
approxval =
      2.718281826198493
ea =
      9.216155641522974e-007
iter =
      12
```

We can see that after 12 iterations, we obtain a result of 2.7182818 with an approximate error estimate of $= 9.2162 \times 10^{-7}\%$. The result can be verified by using the built-in `exp` function to directly calculate the exact value and the true percent relative error,

```
» trueval=exp(1)
trueval =
      2.718281828459046
» et=abs((trueval- approxval)/trueval)*100
et =
      8.316108397236229e-008
```

As was the case with Example 4.1, we obtain the desirable outcome that the true error is less than the approximate error.

2.2. ROUNDOFF ERRORS

Roundoff errors arise because digital computers cannot represent some quantities exactly. They are important to engineering and scientific problem solving because they can lead to erroneous results. In certain cases, they can actually lead to a calculation going unstable and yielding obviously erroneous results. Such calculations are said to be *ill-conditioned*. Worse still, they can lead to subtler discrepancies that are difficult to detect.

There are two major facets of roundoff errors involved in numerical calculations:

1. Digital computers have magnitude and precision limits on their ability to represent numbers.
2. Certain numerical manipulations are highly sensitive to roundoff errors. This can result from both mathematical considerations as well as from the way in which computers perform arithmetic operations.

2.2.1. Computer Number Representation

Numerical roundoff errors are directly related to the manner in which numbers are stored in a computer. The fundamental unit whereby information is represented is called a *word*. This is an entity that consists of a string of binary *digits*, or *bits*. Numbers are typically stored in one or more words. To understand how this is accomplished, we must first review some material related to number systems.

A *number system* is merely a convention for representing quantities. Because we have 10 fingers and 10 toes, the number system that we are most familiar with is the *decimal*, or *base-10*, number system. A base is the number used as the reference for constructing the system. The base-10 system uses the 10 digits—0, 1, 2, 3, 4, 5, 6, 7, 8, and 9—to represent numbers. By themselves, these digits are satisfactory for counting from 0 to 9.

For larger quantities, combinations of these basic digits are used, with the position or *place value* specifying the magnitude. The rightmost digit in a whole number represents a number from 0 to 9. The second digit from the right represents a multiple of 10. The third digit from the right represents a multiple of 100 and so on. For example, if we have the number 8642.9, then we have eight groups of 1000, six groups of 100, four groups of 10, two groups of 1, and nine groups of 0.1, or

$$(8 \times 10^3) + (6 \times 10^2) + (4 \times 10^1) + (2 \times 10^0) + (9 \times 10^{-1}) = 8642.9$$

This type of representation is called *positional notation*.

Now, because the decimal system is so familiar, it is not commonly realized that there are alternatives. For example, if human beings happened to have eight fingers and toes we would undoubtedly have developed an *octal*, or *base-8*, representation. In the same sense, our friend the computer is like a two-fingered animal who is limited to two states—either 0 or 1. This relates to the fact that the primary logic units of digital computers are on/off electronic components. Hence, numbers on the computer are represented with a *binary*, or *base-2*, system. Just as with the decimal

system, quantities can be represented using positional notation. For example, the binary number 101.1 is equivalent to $(1\times 2^2)+(0\times 2^1)+(1\times 2^0)+(1\times 2^{-1})=4+0+1+0.5=5.5$ in the decimal system.

Integer Representation. Now that we have reviewed how base-10 numbers can be represented in binary form, it is simple to conceive of how integers are represented on a computer. The most straightforward approach, called the *signed magnitude method*, employs the first bit of a word to indicate the sign, with a 0 for positive and a 1 for negative. The remaining bits are used to store the number. For example, the integer value of 173 is represented in binary as 10101101:

$$(10101101)_2 = 2^7+2^5+2^3+2^2+2^0 = 128+32+8+4+1 = (173)_{10}$$

Therefore, the binary equivalent of –173 would be stored on a 16-bit computer, as depicted in Fig. 4.3.

If such a scheme is employed, there clearly is a limited range of integers that can be represented. Again assuming a 16-bit word size, if one bit is used for the sign, the 15 remaining bits can represent binary integers from 0 to 111111111111111. The upper limit can be converted to a decimal integer, as in

$(1\times 2^{14})+(1\times 2^{13})+\ldots+(1\times 2^1)+(1\times 2^0)=32,767.$

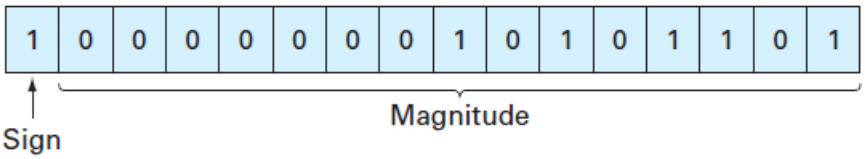


Figure 2.3: The binary representation of the decimal integer -173 on a 16-bit computer using the signed magnitude method.

Note that this value can be simply evaluated as $2^{15}-1$. Thus, a 16-bit computer word can store decimal integers ranging from –32,767 to 32,767.

In addition, because zero is already defined as 0000000000000000, it is redundant to use the number 1000000000000000 to define a "minus zero". Therefore, it is conventionally employed to represent an additional negative number: –32,768, and the range is from –32,768 to 32,767. For an n-bit word, the range would be from -2^{n-1} to $2^{n-1}-1$. Thus, 32-bit integers would range from –2,147,483,648 to +2,147,483,647.

Note that, although it provides a nice way to illustrate our point, the signed magnitude method is not actually used to represent integers for conventional computers. A preferred approach called the 2s *complement* technique directly incorporates the sign into the number's magnitude rather than providing a separate bit to represent plus or minus. Regardless, the range of numbers is still the same as for the signed magnitude method described above.

The foregoing serves to illustrate how all digital computers are limited in their capability to represent integers. That is, numbers above or below the range cannot be represented. A more serious limitation is encountered in the storage and manipulation of fractional quantities as described next.

Floating-Point Representation. Fractional quantities are typically represented in computers using *floating-point format*. In this approach, which is very much like scientific notation, the number is expressed as

$$\pm s\times b^e$$

where s = the *significand* (or *mantissa*), b = the base of the number system being used, and e = the exponent.

Prior to being expressed in this form, the number is *normalize*d by moving the decimal place over so that only one significant digit is to the left of the decimal point. This is done so computer memory is not wasted on storing useless nonsignificant zeros. For example, a value like 0.005678 could be represented in a wasteful manner as 0.005678×10^0. However, normalization would yield 5.678×10^{-3} which eliminates the useless zeroes.

Before describing the base-2 implementation used on computers, we will first explore the fundamental implications of such floating-point representation. In particular, what are the ramifications of the fact that in order to be stored in the computer, both the mantissa and the exponent must be limited to a finite number of bits? As in the next example, a nice way to do this is within the context of our more familiar base-10 decimal world.

Example 2.2. Implications of Floating-Point Representation

Problem Statement. Suppose that we had a hypothetical base-10 computer with a 5-digit word size. Assume that one digit is used for the sign, two for the exponent, and two for the mantissa. For simplicity, assume that one of the exponent digits is used for its sign, leaving a single digit for its magnitude.

Solution. A general representation of the number following normalization would be

$$s_1 d_1 d_2 \times 10^{s_0 d_0}$$

where s_0 and s_1 = the signs, d_0 = the magnitude of the exponent, and d_1 and d_2 = the magnitude of the significand digits.

Now, let's play with this system. First, what is the largest possible positive quantity that can be represented? Clearly, it would correspond to both signs being positive and all magnitude digits set to the largest possible value in base-10, that is, 9:

$$\text{Largest value} = +9.9 \times 10^{+9}$$

So the largest possible number would be a little less than 10 billion. Although this might seem like a big number, it's really not that big. For example, this computer would be incapable of representing a commonly used constant like Avogadro's number (6.022×1023). In the same sense, the smallest possible positive number would be

$$\text{Smallest value} = +1.0 \times 10^{-9}$$

Again, although this value might seem pretty small, you could not use it to represent a quantity like Planck's constant $(6.626 \times 10^{-34} J \cdot s)$.

Similar negative values could also be developed. The resulting ranges are displayed in Fig. 4.4. Large positive and negative numbers that fall outside the range would cause an overflow error. In a similar sense, for very small quantities there is a "hole" at zero, and very small quantities would usually be converted to zero.

Recognize that the exponent overwhelmingly determines these range limitations. For example, if we increase the mantissa by one digit, the maximum value increases slightly to 9.99×10^9. In contrast, a one-digit increase in the exponent raises the maximum by 90 orders of magnitude to 9.9×10^{99}!

When it comes to precision, however, the situation is reversed. Whereas the significand plays a minor role in defining the range, it has a profound effect on specifying the precision. This is dramatically illustrated for this example where we have limited the significand to only 2 digits. As in Fig. 4.5, just as there is a "hole" at zero, there are also "holes" between values.

For example, a simple rational number with a finite number of digits like $2^{-5} = 0.03125$ would have to be stored as 3.1×102 or 0.031. Thus, a *roundoff error* is introduced. For this case, it represents a relative error of

$$\frac{0.03125 - 0.031}{0.03125} = 0.008$$

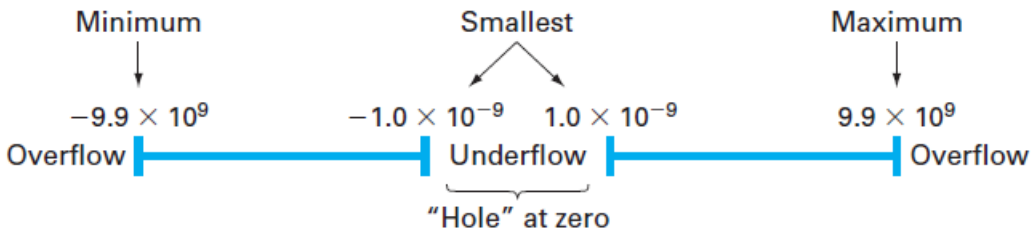


Figure 2.4: The number line showing the possible ranges corresponding to the hypothetical base-10 floating-point scheme described in Example 4.2.

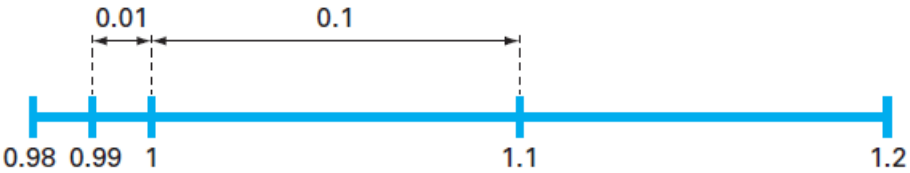


Figure 2.5: A small portion of the number line corresponding to the hypothetical base-10 floating-point scheme described in Example 4.2. The numbers indicate values that can be represented exactly. All other quantities falling in the "holes" between these values would exhibit some roundoff error.

While we could store a number like 0.03125 exactly by expanding the digits of the significand, quantities with infinite digits must always be approximated. For example, a commonly used constant such as $\pi (= 3.14159\ldots)$ would have to be represented as 3.1×10^0 or 3.1. For this case, the relative error is

$$\frac{3.14159 - 3.1}{3.14159} = 0.0132$$

Although adding significand digits can improve the approximation, such quantities will always have some roundoff error when stored in a computer.

Another more subtle effect of floating-point representation is illustrated by Fig. 4.5. Notice how the interval between numbers increases as we move between orders of magnitude. For numbers with an exponent of –1 (i.e., between 0.1 and 1), the spacing is 0.01. Once we cross over into the range from 1 to 10, the spacing increases to 0.1. This means that the roundoff error of a number will be proportional to its magnitude. In addition, it means that the relative error will have an upper bound. For this example, the maximum relative error would be 0.05. This value is called the *machine epsilon* (or machine precision).

■

As illustrated in Example 4.2, the fact that both the exponent and significand are finite means that there are both range and precision limits on floating-point representation. Now, let us examine how floating-point quantities are actually represented in a real computer using base-2 or binary numbers.

First, let's look at normalization. Since binary numbers consist exclusively of 0s and 1s, a bonus occurs when they are normalized. That is, the bit to the left of the binary point will always be one! This means that this leading bit does not have to be stored. Hence, nonzero binary floating-point numbers can be expressed as

$$\pm(1+f)\times 2^e$$

where f = the *mantissa* (i.e., the fractional part of the significand). For example, if we normalized the binary number 1101.1, the result would be $1.1011\times(2)^{-3}$ or $(1+0.1011)\times 2^{-3}$.

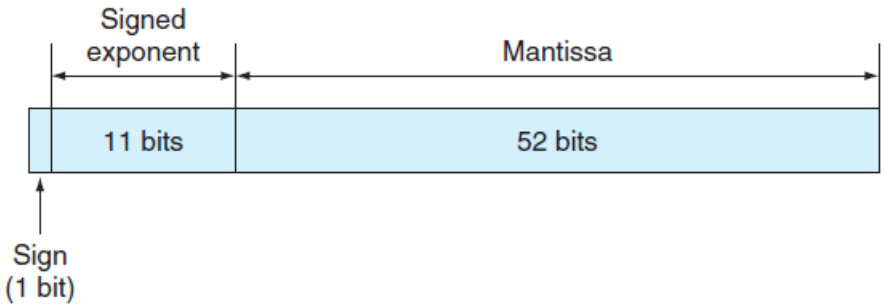


Figure 2.6: The manner in which a floating-point number is stored in an 8-byte word in IEEE doubleprecision format.

Thus, although the original number has five significant bits, we only have to store the four fractional bits: 0.1011.

By default, MATLAB has adopted the *IEEE double-precision format* in which eight bytes (64 bits) are used to represent floating-point numbers. As in Fig. 4.6, one bit is reserved for the number's sign. In a similar spirit to the way in which integers are stored, the exponent and its sign are stored in 11 bits. Finally, 52 bits are set aside for the mantissa. However, because of normalization, 53 bits can be stored.

Now, just as in Example 4.2, this means that the numbers will have a limited range and precision. However, because the IEEE format uses many more bits, the resulting number system can be used for practical purposes.

Range In a fashion similar to the way in which integers are stored, the 11 bits used for the exponent translates into a range from –1022 to 1023. The largest positive number can be represented in binary as

$$\text{Largest value} = +1.1111\ldots1111\times 2^{+1023}$$

where the 52 bits in the mantissa are all 1. Since the significant is approximately 2 (it is actually $2-2^{-52}$), the largest value is therefore $2^{1024}=1.7977\times 10^{308}$. In a similar fashion, the smallest positive number can be represented as

$$\text{Smallest value} = +1.0000\ldots0000\times 2^{-1022}$$

This value can be translated into a base-10 value of $2^{-1022}=2.2251\times 10^{-308}$

Precision. The 52 bits used for the mantissa correspond to about 15 to 16 base-10 digits. Thus, π would be expressed as

```
» format long
» pi
ans =
      3.14159265358979
```

Note that the machine epsilon is $2^{-52}=2.2204\times 10^{-16}$

MATLAB has a number of built-in functions related to its internal number representation. For example, the `realmax` function displays the largest positive real number:

```
» format long
» realmax
ans =
      1.797693134862316e+308
```

Numbers occurring in computations that exceed this value create an overflow. In MATLAB they are set to infinity, `inf`. The `realmin` function displays the smallest positive real number:

```
» realmin
ans =
      2.225073858507201e-308
```

Numbers that are smaller than this value create an *underflow* and, in MATLAB, are set to zero. Finally, the `eps` function displays the machine epsilon:

```
» eps
ans=
      2.220446049250313e-016
```

2.2.2. Arithmetic Manipulations of Computer Numbers

Aside from the limitations of a computer's number system, the actual arithmetic manipulations involving these numbers can also result in roundoff error. To understand how this occurs, let's look at how the computer performs simple addition and subtraction.

Because of their familiarity, normalized base-10 numbers will be employed to illustrate the effect of roundoff errors on simple addition and subtraction. Other number bases would behave in a similar fashion. To simplify the discussion, we will employ a hypothetical decimal computer with a 4-digit mantissa and a 1-digit exponent.

When two floating-point numbers are added, the numbers are first expressed so that they have the same exponents. For example, if we want to add $1.557 + 0.04341$, the computer would express the numbers as $0.1557 \times 10^1 + 0.004341 \times 10^1$. Then the mantissas are added to give 0.160041×10^1. Now, because this hypothetical computer only carries a 4-digit mantissa, the excess number of digits get chopped off and the result is 0.1600×10^1. Notice how the last two digits of the second number (41) that were shifted to the right have essentially been lost from the computation.

Subtraction is performed identically to addition except that the sign of the subtrahend is reversed. For example, suppose that we are subtracting 26.86 from 36.41. That is,

$$\begin{array}{rl} & 0.3641 \times 10^2 \\ - & 0.2686 \times 10^2 \\ \hline & 0.0955 \times 10^2 \end{array}$$

For this case the result must be normalized because the leading zero is unnecessary. So we must shift the decimal one place to the right to give $0.9550 \times 10^1 = 9.550$. Notice that the zero added to the end of the mantissa is not significant but is merely appended to fill the empty space created by the shift. Even more dramatic results would be obtained when the numbers are very close as in

$$\begin{array}{rl} & 0.7642 \times 10^3 \\ - & 0.7641 \times 10^3 \\ \hline & 0.0001 \times 10^3 \end{array}$$

which would be converted to $0.1000 \times 10^0 = 0.1000$. Thus, for this case, three nonsignificant zeros are appended. The subtracting of two nearly equal numbers is called *subtractive cancellation*. It is the classic example of how the manner in which computers handle mathematics can lead to numerical problems. Other calculations that can cause problems include:

Large Computations. Certain methods require extremely large numbers of arithmetic manipulations to arrive at their final results. In addition, these computations are often interdependent. That is, the later calculations are dependent on the results of earlier ones. Consequently, even though an individual roundoff error could be small, the cumulative effect over the course of a large computation can be significant. Avery simple case involves summing a round base-10 number that is not round in base-2. Suppose that the following M-file is constructed:

```
function sout = sumdemo()
s = 0;
for i = 1:10000
    s = s + 0.0001;
end
sout = s;
```

When this function is executed, the result is

```
» format long
» sumdemo
ans =
      0.99999999999991
```

The `format long` command lets us see the 15 significant-digit representation used by MATLAB. You would expect that sum would be equal to 1. However, although 0.0001 is a nice round number in base-10, it cannot be expressed exactly in base-2. Thus, the sum comes out to be slightly different than 1. We should note that MATLAB has features that are designed to minimize such errors. For example, suppose that you form a vector as in

```
» format long
s = [0:0.0001:1];
```

For this case, rather than being equal to 0.99999999999991, the last entry will be exactly one as verified by

```
» s(10001)
ans =
      1
```

Adding a Large and a Small Number. Suppose we add a small number, 0.0010, to a large number, 4000, using a hypothetical computer with the 4-digit mantissa and the 1-digit exponent. After modifying the smaller number so that its exponent matches the larger,

$$\begin{array}{ll} 0.4000 & \times 10^4 \\ 0.0000001 & \times 10^4 \\ \hline 0.4000001 & \times 10^4 \end{array}$$

which is chopped to 0.4000×10^4 . Thus, we might as well have not performed the addition! This type of error can occur in the computation of an infinite series. The initial terms in such series are often relatively large in comparison with the later terms. Thus, after a few terms have been added, we are in the situation of adding a small quantity to a large quantity. One way to mitigate this type of error is to sum the series in reverse order. In this way, each new term will be of comparable magnitude to the accumulated sum.

Smearing. Smearing occurs whenever the individual terms in a summation are larger than the summation itself. One case where this occurs is in a series of mixed signs.

Inner Products. As should be clear from the last sections, some infinite series are particularly prone to roundoff error. Fortunately, the calculation of series is not one of the more common operations in numerical methods. A far more ubiquitous manipulation is the calculation of inner products as in

$$\sum_{i=1}^{n} x_i y_i = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n$$

This operation is very common, particularly in the solution of simultaneous linear algebraic equations. Such summations are prone to roundoff error. Consequently, it is often desirable to compute such summations in double precision as is done automatically in MATLAB.

2.3. TRUNCATION ERRORS

Truncation errors are those that result from using an approximation in place of an exact mathematical procedure. For example, in Chap. 1 we approximated the derivative of velocity of a bungee jumper by a finite-difference equation of the form [Eq. (1.11)]

$$\frac{dv}{dt} \cong \frac{\Delta v}{\Delta t} = \frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i} \tag{4.8}$$

A truncation error was introduced into the numerical solution because the difference equation only approximates the true value of the derivative (recall Fig. 1.3). To gain insight into the properties of such errors, we now turn to a mathematical formulation that is used widely in numerical methods to express functions in an approximate fashion—the Taylor series.

2.3.1. The Taylor Series

Taylor's theorem and its associated formula, the Taylor series, is of great value in the study of numerical methods. In essence, the *Taylor theorem* states that any smooth function can be approximated as a polynomial. The *Taylor series* then provides a means to express this idea mathematically in a form that can be used to generate practical results.

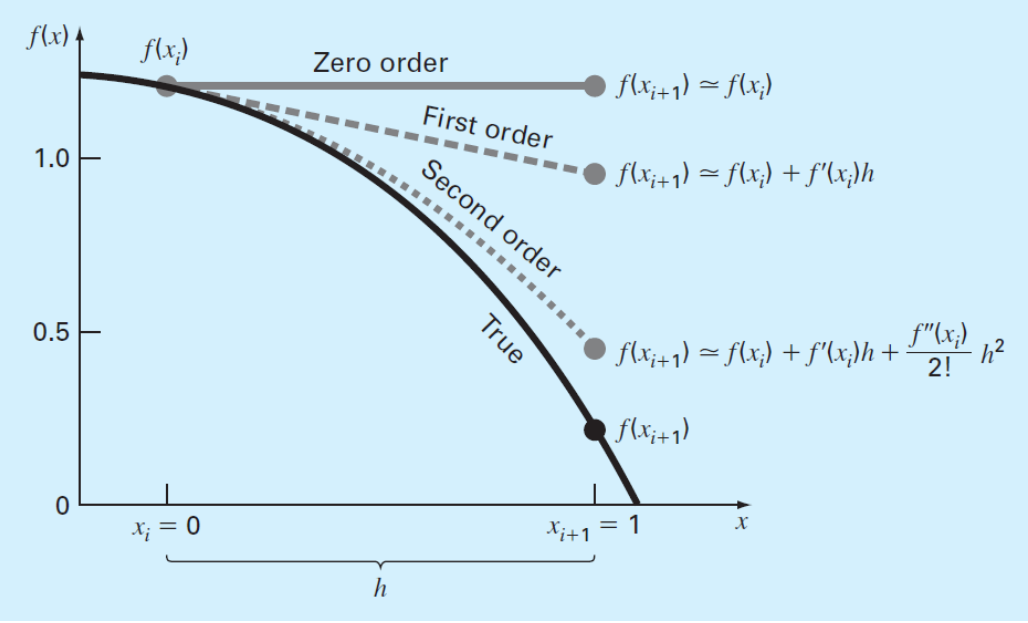


Figure 2.7: The approximation of $f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x = 1.2$ at $x = 1$ by zero-order, first-order, and second-order Taylor series expansions.

A useful way to gain insight into the Taylor series is to build it term by term. A good problem context for this exercise is to predict a function value at one point in terms of the function value and its derivatives at another point.

Suppose that you are blindfolded and taken to a location on the side of a hill facing downslope (Fig. 4.7). We'll call your horizontal location x_i and your vertical distance with respect to the base of the hill $f(x_i)$. You are given the task of predicting the height at a position x_{i+1}, which is a distance h away from you.

At first, you are placed on a platform that is completely horizontal so that you have no idea that the hill is sloping down away from you. At this point, what would be your best guess at the height at x_{i+1}? If you think about it (remember you have no idea whatsoever what's in front of you), the best guess would be the same height as where you're standing now! You could express this prediction mathematically as

$$f(x_{i+1}) \cong f(x_i) \tag{4.9}$$

This relationship, which is called the *zero-order approximation*, indicates that the value of f at the new point is the same as the value at the old point. This result makes intuitive sense because if x_i and x_{i+1} are close to each other, it is likely that the new value is probably similar to the old value.

Equation (4.9) provides a perfect estimate if the function being approximated is, in fact, a constant. For our problem, you would be right only if you happened to be standing on a perfectly flat plateau. However, if the function changes at all over the interval, additional terms of the Taylor series are required to provide a better estimate.

So now you are allowed to get off the platform and stand on the hill surface with one leg positioned in front of you and the other behind. You immediately sense that the front foot is lower than the back foot. In fact, you're allowed to obtain a quantitative estimate of the slope by measuring the difference in elevation and dividing it by the distance between your feet.

With this additional information, you're clearly in a better position to predict the height at $f(x_{i+1})$. In essence, you use the slope estimate to project a straight line out to x_{i+1}. You can express this prediction mathematically by

$$f(x_{i+1}) \cong f(x_i) + f'(x_i)h \tag{4.10}$$

This is called a *first-order approximation* because the additional first-order term consists of a slope $f'(x_i)$ multiplied by h, the distance between x_i and x_{i+1}. Thus, the expression is now in the form of a straight line that is capable of predicting an increase or decrease of the function between x_i and x_{i+1}.

Although Eq. (4.10) can predict a change, it is only exact for a straight-line, or *linear*, trend. To get a better prediction, we need to add more terms to our equation. So now you are allowed to stand on the hill surface and take two measurements. First, you measure the slope behind you by keeping one foot planted at x_i and moving the other one back a distance Δx. Let's call this slope $f'_b(x_i)$. Then you measure the slope in front of you by keeping one foot planted at x_i and moving the other one forward Δx. Let's call this slope $f'_f(x_i)$. You immediately recognize that the slope behind is milder than the one in front. Clearly the drop in height is "accelerating" downward in front of you. Thus, the odds are that $f(x_i)$ is even lower than your previous linear prediction.

As you might expect, you're now going to add a second-order term to your equation and make it into a parabola. The Taylor series provides the correct way to do this as in

$$f(x_{i+1}) \cong f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 \tag{4.11}$$

To make use of this formula, you need an estimate of the second derivative. You can use the last two slopes you determined to estimate it as

$$f''(x_{i+1}) \cong \frac{f'_f(x_i) - f'_b(x_i)}{\Delta x} \tag{4.12}$$

Thus, the second derivative is merely a derivative of a derivative; in this case, the rate of change of the slope.

Before proceeding, let's look carefully at Eq. (4.11). Recognize that all the values subscripted i represent values that you have estimated. That is, they are numbers. Consequently, the only unknowns are the values at the prediction position x_{i+1}. Thus, it is a quadratic equation of the form

$$f(h) \cong a_2h^2 + a_1h + a_0$$

Thus, we can see that the second-orderTaylor series approximates the function with a secondorder polynomial.

Clearly, we could keep adding more derivatives to capture more of the functionÂŠs curvature. Thus, we arrive at the complete Taylor series expansion

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 + \ldots + \frac{f^{(n)}(x_i)}{n!}h^n + R_n \tag{4.13}$$

Note that because Eq. (4.13) is an infinite series, an equal sign replaces the approximate sign that was used in Eqs. (4.9) through (4.11). A remainder term is also included to account for all terms from $n+1$ to infinity:

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}h^{n+1} \tag{4.14}$$

where the subscript n connotes that this is the remainder for the nth-order approximation and ξ is a value of x that lies somewhere between x_i and x_{i+1}.

We can now see why the Taylor theorem states that any smooth function can be approximated as a polynomial and that the Taylor series provides a means to express this idea mathematically.

In general, the nth-order Taylor series expansion will be exact for an nth-order polynomial. For other differentiable and continuous functions, such as exponentials and sinusoids, a finite number of terms will not yield an exact estimate. Each additional term will contribute some improvement, however slight, to the approximation. This behavior will be demonstrated in Example 4.3. Only if an infinite number of terms are added will the series yield an exact result.

Although the foregoing is true, the practical value of Taylor series expansions is that, in most cases, the inclusion of only a few terms will result in an approximation that is close enough to the true value for practical purposes. The assessment of how many terms are required to get "close enough" is based on the remainder term of the expansion (Eq. 4.14). This relationship has two major drawbacks. First, ξ is not known exactly but merely lies somewhere between x_i and x_{i+1}. Second, to evaluate Eq. (4.14), we need to determine the $(n+1)$th derivative of $f(x)$. To do this, we need to know $f(x)$. However, if we knew $f(x)$, there would be no need to perform the Taylor series expansion in the present context!

Despite this dilemma, Eq. (4.14) is still useful for gaining insight into truncation errors. This is because we *do* have control over the term h in the equation. In other words, we can choose how far away from x we want to evaluate $f(x)$, and we can control the number of terms we include in the expansion. Consequently, Eq. (4.14) is often expressed as

$$R_n = O(h^{n+1})$$

where the nomenclature $O(h^{n+1})$ means that the truncation error is of the order of h^{n+1}. That is, the error is proportional to the step size h raised to the $(n+1)$th power. Although this approximation implies nothing regarding the magnitude of the derivatives that multiply h^{n+1}, it is extremely useful in judging the comparative error of numerical methods based on Taylor series expansions. For example, if the error is $O(h)$, halving the step size will halve the error. On the other hand, if the error is $O(h^2)$, halving the step size will quarter the error.

In general, we can usually assume that the truncation error is decreased by the addition of terms to the Taylor series. In many cases, if h is sufficiently small, the first- and other lower-order terms usually account for a disproportionately high percent of the error. Thus, only a few terms are required to obtain an adequate approximation. This property is illustrated by the following example.

Example 2.3. Approximation of a Function with a Taylor Seres Expansion

Problem Statement. Use Taylor series expansions with $n = 0$ to 6 to approximate $f(x) = cosx$ at $x_{i+1} = \pi/3$ on the basis of the value of $f(x)$ and its derivatives at $x_i = \pi/4$. Note that this means that $h = \pi/3 - \pi/4 = \pi/12$.

Solution. Our knowledge of the true function allows us to determine the correct value $f(\pi/3) = 0.5$. The zero-order approximation is [Eq. (4.9)]

$$f\left(\frac{\pi}{3}\right) \cong cos\left(\frac{\pi}{4}\right) = 0.707106781$$

which represents a percent relative error of

$$\varepsilon_t = \left|\frac{0.5 - 0.707106781}{0.5}\right| 100\% = 41.1\%$$

For the first-order approximation, we add the first derivative term where $f'(x) = -sinx$:

$$f\left(\frac{\pi}{3}\right) \cong cos\left(\frac{\pi}{4}\right) - sin\left(\frac{\pi}{4}\right)\left(\frac{\pi}{12}\right) = 0.521986659$$

which has $|\varepsilon_t| = 4.40\%$. For the second-order approximation, we add the second derivative term where $f''(x) = -cosx$:

$$f\left(\frac{\pi}{3}\right) \cong cos\left(\frac{\pi}{4}\right) - sin\left(\frac{\pi}{4}\right)\left(\frac{\pi}{12}\right) - \frac{cos(\pi/4)}{2}\left(\frac{\pi}{12}\right)^2 = 0.497754491$$

with $|\varepsilon_t| = 0.449\%$. Thus, the inclusion of additional terms results in an improved estimate. The process can be continued and the results listed as in

Order n	$f^{(n)}(x)$	$f(\pi/3)$	$\lvert\varepsilon_t\rvert$
0	$\cos x$	0.707106781	41.4
1	$-\sin x$	0.521986659	4.40
2	$-\cos x$	0.497754491	0.449
3	$\sin x$	0.499869147	2.62×10^{-2}
4	$\cos x$	0.500007551	1.51×10^{-3}
5	$-\sin x$	0.500000304	6.08×10^{-5}
6	$-\cos x$	0.499999988	2.44×10^{-6}

Notice that the derivatives never go to zero as would be the case for a polynomial. Therefore, each additional term results in some improvement in the estimate. However, also notice how most of the improvement comes with the initial terms. For this case, by the time we have added the third-order term, the error is reduced to 0.026%, which means that we have attained 99.974% of the true value. Consequently, although the addition of more terms will reduce the error further, the improvement becomes negligible.

■

2.3.2. The Remainder for the Taylor Series Expansion

Before demonstrating how the Taylor series is actually used to estimate numerical errors, we must explain why we included the argument ξ in Eq. (4.14). To do this, we will use a simple, visually based explanation.

Suppose that we truncated the Taylor series expansion [Eq. (4.13)] after the zero-order term to yield

$$f(x_{i+1}) \cong f(x_i)$$

A visual depiction of this zero-order prediction is shown in Fig. 4.8. The remainder, or error, of this prediction, which is also shown in the illustration, consists of the infinite series of terms that were truncated

$$R_0 = f'(x_i)h + \frac{f''(x_1)}{2!}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 \ldots$$

It is obviously inconvenient to deal with the remainder in this infinite series format. One simplification might be to truncate the remainder itself, as in

$$R_0 \cong f'(x_i)h \tag{4.15}$$

Although, as stated in the previous section, lower-order derivatives usually account for a greater share of the remainder than the higher-order terms, this result is still inexact because of the neglected second- and higher-order terms. This "inexactness" is implied by the approximate equality symbol ($\cong$) employed in Eq. (4.15). An alternative simplification that transforms the approximation into an equivalence is based on a graphical insight. As in Fig. 4.9, the *derivative mean-value theorem* states that if a function $f(x)$ and its first derivative are continuous over an interval from x_i to x_{i+1}, then

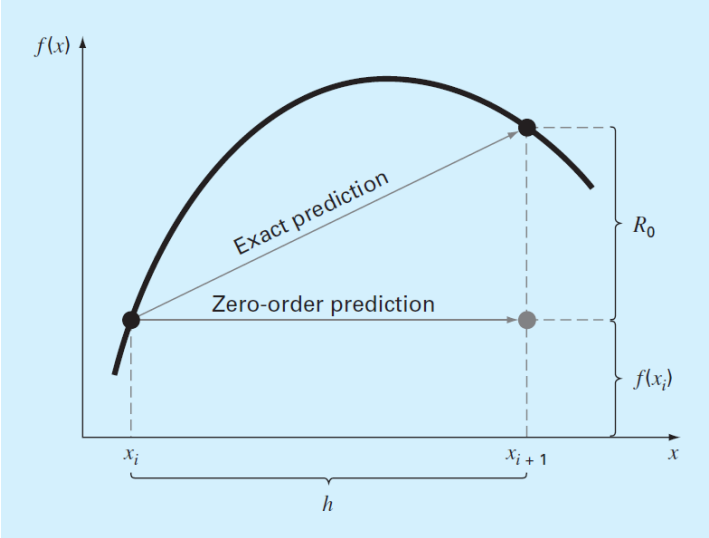


Figure 2.8: Graphical depiction of a zero-order Taylor series prediction and remainder.

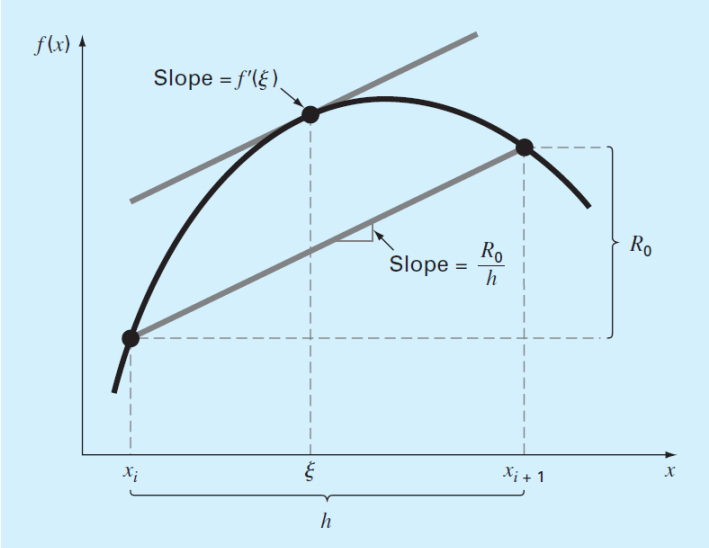


Figure 2.9: Graphical depiction of the derivative mean-value theorem.

there exists at least one point on the function that has a slope, designated by $f'(\xi)$, that is parallel to the line joining $f(x_i)$ and $f(x_{i+1})$. The parameter ξ marks the x value where this slope occurs (Fig. 4.9). A physical illustration of this theorem is that, if you travel between two points with an average velocity, there will be at least one moment during the course of the trip when you will be moving at that average velocity. By invoking this theorem, it is simple to realize that, as illustrated in Fig. 4.9, the slope $f'(\xi)$ is equal to the rise R_0 divided by the run h, or

$$f'(\xi) = \frac{R_0}{h}$$

which can be rearranged to give

$$R_0 = f'(\xi)h \tag{4.16}$$

Thus, we have derived the zero-order version of Eq. (4.14). The higher-order versions are merely a logical extension of the reasoning used to derive Eq. (4.16). The first-order version is

$$R_1 = \frac{f''(\xi)}{2!}h^2 \tag{4.17}$$

For this case, the value of ξ conforms to the x value corresponding to the second derivative that makes Eq. (4.17) exact. Similar higher-order versions can be developed from Eq. (4.14).

2.3.3. Using the Taylor Series to Estimate Truncation Errors

Although the Taylor series will be extremely useful in estimating truncation errors throughout this book, it may not be clear to you how the expansion can actually be applied to numerical methods. In fact, we have already done so in our example of the bungee jumper. Recall that the objective of both Examples 1.1 and 1.2 was to predict velocity as a function of time. That is, we were interested in determining $v(t)$. As specified by Eq. (4.13), $v(t)$ can be expanded in a Taylor series:

$$v(t_{i+1}) = v(t_i) + v'(t_i)(t_{i+1} - t_i) + \frac{v''(t_i)}{2!}(t_{i+1} - t_i)^2 + \ldots + R_n$$

Now let us truncate the series after the first derivative term:

$$v(t_{i+1}) = v(t_i) + v'(t_i)(t_{i+1} - t_i) + R_1 \tag{4.18}$$

Equation (4.18) can be solved for

$$v'(t_i) = \underbrace{\frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i}}_{\text{First-order approximation}} - \underbrace{\frac{R_1}{t_{i+1} - t_i}}_{\text{Truncation error}} \tag{4.19}$$

The first part of Eq. (4.19) is exactly the same relationship that was used to approximate the derivative in Example 1.2 [Eq. (1.11)]. However, because of the Taylor series approach, we have now obtained an estimate of the truncation error associated with this approximation of the derivative. Using Eqs. (4.14) and (4.19) yields

$$\frac{R_1}{t_{i+1} - t_i} = \frac{v''(\xi)}{2!}(t_{i+1} - t_i)$$

or

$$\frac{R_1}{t_{i+1} - t_i} = O(t_{i+1} - t_i)$$

Thus, the estimate of the derivative [Eq. (1.11) or the first part of Eq. (4.19)] has a truncation error of order $t_{i+1} - t_i$. In other words, the error of our derivative approximation should be proportional to the step size. Consequently, if we halve the step size, we would expect to halve the error of the derivative.

2.3.4. Numerical Differentiation

Equation (4.19) is given a formal label in numerical methods—it is called a *finite difference*. It can be represented generally as

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} + O(x_{i+1} - x_i) \tag{4.20}$$

or

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h) \tag{4.21}$$

where h is called the step size—that is, the length of the interval over which the approximation is made, $x_{i+1} - x_i$. It is termed a "forward" difference because it utilizes data at i and $i+1$ to estimate the derivative (Fig. 4.10a).

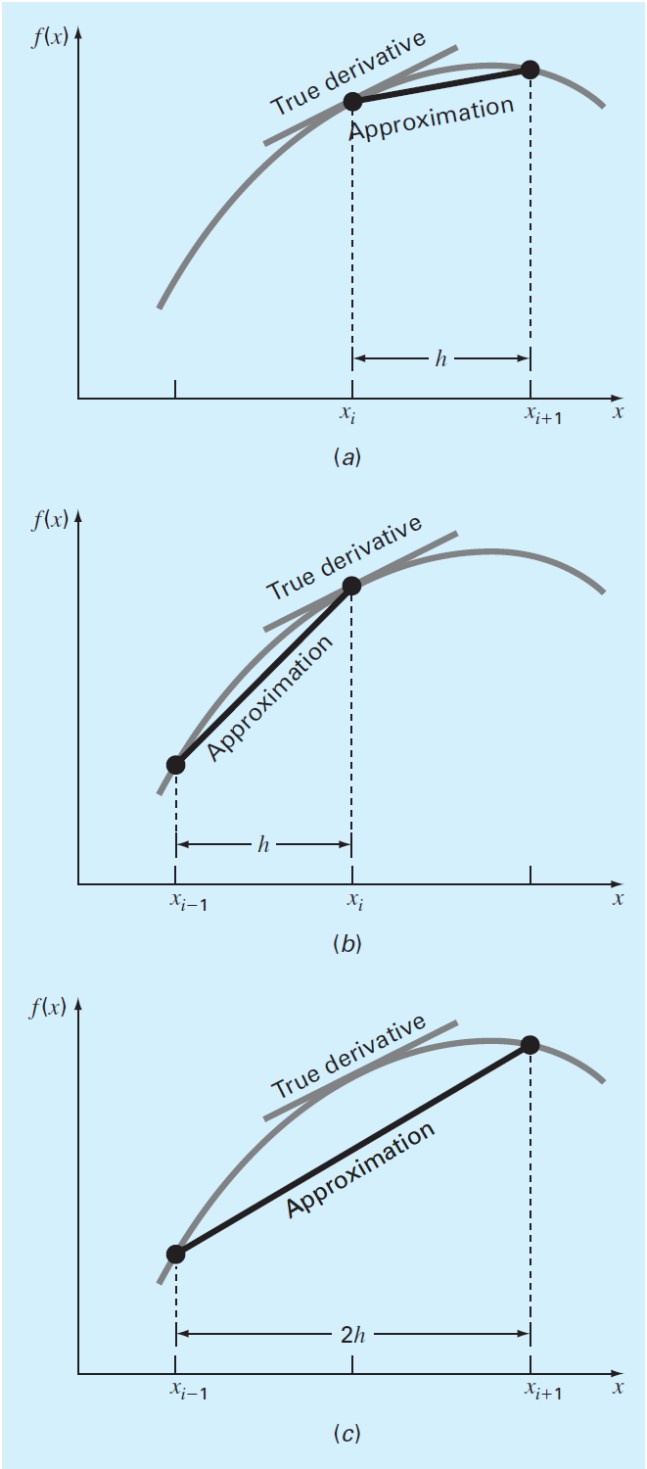


Figure 2.10: Graphical depiction of (a) forward, (b) backward, and (c) centered finite-difference approximations of the first derivative.

This forward difference is but one of many that can be developed from the Taylor series to approximate derivatives numerically. For example, backward and centered difference approximations of the first derivative can be developed in a fashion similar to the derivation of Eq. (4.19). The former utilizes values at x_{i-1} and x_i (Fig. 4.10b), whereas the latter uses values that are equally spaced around the point at which the derivative is estimated (Fig. 4.10c). More accurate approximations of the first derivative can be developed by including higher-order terms of the Taylor series. Finally, all the foregoing versions can also be developed for second, third, and higher derivatives. The following sections provide brief summaries illustrating how some of these cases are derived.

Backward Difference Approximation of the First Derivative. The Taylor series can be expanded backward to calculate a previous value on the basis of a present value, as in

$$f(x_{i-1}) = f(x_i) - f'(x_i)h + \frac{f''(x_i)}{2!}h^2 - \dots \tag{4.22}$$

Truncating this equation after the first derivative and rearranging yields

$$f'(x_i) \cong \frac{f(x_i) - f(x_{i-1})}{h} \tag{4.23}$$

where the error is $O(h)$.

Centered Difference Approximation of the First Derivative. A third way to approximate the first derivative is to subtract Eq. (4.22) from the forward Taylor series expansion:

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \dots \tag{4.24}$$

to yield

$$f(x_{i+1}) = f(x_{i-1}) + 2f'(x_i)h + 2\frac{f^{(3)}(x_i)}{3!}h^3 + \dots$$

which can be solved for

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} - \frac{f^{(3)}(x_i)}{6}h^2 + \dots$$

or

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} - O(h^2) \tag{4.25}$$

Equation (4.25) is a *centered finite difference* representation of the first derivative. Notice that the truncation error is of the order of h^2 in contrast to the forward and backward approximations that were of the order of h. Consequently, the Taylor series analysis yields the practical information that the centered difference is a more accurate representation of the derivative (Fig. 4.10c). For example, if we halve the step size using a forward or backward difference, we would approximately halve the truncation error, whereas for the central difference, the error would be quartered.

Example 2.4. Finite-Difference Approximations of Derivatives

Problem Statement. Use forward and backward difference approximations of $O(h)$ and a centered difference approximation of $O(h^2)$ to estimate the first derivative of

$$f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x + 1.2$$

at $x = 0.5$ using a step size $h = 0.5$. Repeat the computation using $h = 0.25$. Note that the derivative can be calculated directly as

$$f'(x) = -0.4x^3 - 0.45x^2 - 1.0x - 0.25$$

and can be used to compute the true value as $f'(0.5) = -0.9125$.

Solution. For $h = 0.5$, the function can be employed to determine

$$x_{i-1} = 0 \qquad f(X_{i-1}) = 1.2$$

$$x_i = 0.5 \qquad f(x_i) = 0.925$$

$$x_{i+1} = 1.0 \qquad f(x_{i+1}) = 0.2$$

These values can be used to compute the forward difference [Eq. (4.21)],

$$f'(0.5) \cong \frac{0.2 - 0.925}{0.5} = -1.45 \qquad |\varepsilon_t| = 58.9\%$$

the backward difference [Eq. (4.23)],

$$f'(0.5) \cong \frac{0.925 - 1.2}{0.5} = -0.55 \qquad |\varepsilon_t| = 39.7\%$$

and the centered difference [Eq. (4.25)],

$$f'(0.5) \cong \frac{0.2 - 1.2}{1.0} = -1.0 \qquad |\varepsilon_t| = 9.6\%$$

For $h = 0.25$,

$$x_{i-1} = 0.25 \qquad f(x_{i-1}) = 1.10351563$$

$$x_i = 0.5 \qquad f(x_i) = 0.925$$

$$x_{i+1} = 0.75 \qquad f(x_{i+1}) = 0.63632813$$

which can be used to compute the forward difference,

$$f'(0.5) \cong \frac{0.63632813 - 0.925}{0.25} = -1.155 \qquad |\varepsilon_t| = 26.5\%$$

the backward difference,

$$f'(0.5) \cong \frac{0.925 - 1.10351563}{0.25} = -0.714 \qquad |\varepsilon_t| = 21.7\%$$

and the centered difference,

$$f'(0.5) \cong \frac{0.63632813 - 1.10351563}{0.5} = -0.934 \qquad |\varepsilon_t| = 2.4\%$$

For both step sizes, the centered difference approximation is more accurate than forward or backward differences. Also, as predicted by the Taylor series analysis, halving the step size approximately halves the error of the backward and forward differences and quarters the error of the centered difference.

■

Finite-Difference Approximations of Higher Derivatives. Besides first derivatives, the Taylor series expansion can be used to derive numerical estimates of higher derivatives. To do this, we write a forward Taylor series expansion for $f(x_{i+2})$ in terms of $f(x_i)$:

$$f(x_{i+2}) = f(x_i) + f'(x_i)(2h) + \frac{f''(x_i)}{2!}(2h)^2 + \ldots \tag{4.26}$$

Equation (4.24) can be multiplied by 2 and subtracted from Eq. (4.26) to give

$$f(x_{i+2}) - 2f(x_{i+1}) = -f(x_i) + f''(x_i)h^2 + \ldots$$

which can be solved for

$$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2} + O(h) \tag{4.27}$$

This relationship is called the *second forward finite difference*. Similar manipulations can be employed to derive a backward version

$$f''(x_i) = \frac{f(x_i) - 2f(x_{i-1}) + f(x_{i-2})}{h^2} + O(h)$$

A centered difference approximation for the second derivative can be derived by adding Eqs. (4.22) and (4.24) and rearranging the result to give

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2} + O(h^2)$$

As was the case with the first-derivative approximations, the centered case is more accurate. Notice also that the centered version can be alternatively expressed as

$$f''(x_i) \cong \frac{\dfrac{f(x_{i+1}) - f(x_i)}{h} - \dfrac{f(x_i) - f(x_{i-1})}{h}}{h}$$

Thus, just as the second derivative is a derivative of a derivative, the second finite difference approximation is a difference of two first finite differences [recall Eq. (4.12)].

2.4. TOTAL NUMERICAL ERROR

The *total numerical error* is the summation of the truncation and roundoff errors. In general, the only way to minimize roundoff errors is to increase the number of significant figures of the computer. Further, we have noted that roundoff error may *increase* due to subtractive cancellation or due to an increase in the number of computations in an analysis. In contrast, Example 4.4 demonstrated that the truncation error can be reduced by decreasing the step size. Because a decrease in step size can lead to subtractive cancellation or to an increase in computations, the truncation errors are *decreased* as the roundoff errors are *increased*.

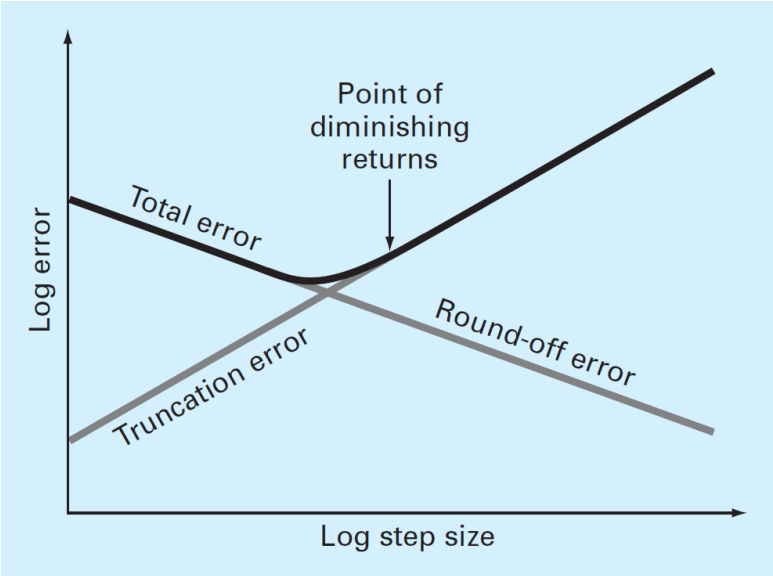


Figure 2.11: A graphical depiction of the trade-off between roundoff and truncation error that sometimes comes into play in the course of a numerical method. The point of diminishing returns is shown, where roundoff error begins to negate the benefits of step-size reduction.

Therefore, we are faced by the following dilemma: The strategy for decreasing one component of the total error leads to an increase of the other component. In a computation, we could conceivably decrease the step size to minimize truncation errors only to discover that in doing so, the roundoff error begins to dominate the solution and the total error grows! Thus, our remedy becomes our problem (Fig. 4.11). One challenge that we face is to determine an appropriate step size for a particular computation. We would like to choose a large step size to decrease the amount of calculations and roundoff errors without incurring the penalty of a large truncation error. If the total error is as shown in Fig. 4.11, the challenge is to identify the point of diminishing returns where roundoff error begins to negate the benefits of step-size reduction. When using MATLAB, such situations are relatively uncommon because of its 15- to 16- digit precision. Nevertheless, they sometimes do occur and suggest a sort of "numerical uncertainty principle" that places an absolute limit on the accuracy that may be obtained using certain computerized numerical methods. We explore such a case in the following section.

2.4.1. Error Analysis of Numerical Differentiation

As described in Sec. 4.3.4, a centered difference approximation of the first derivative can be written as (Eq. 4.25)

$$\underset{\text{True value}}{f'(x_i)} = \underbrace{\frac{f(x_{i+1})-f(x_{i-1})}{2h}}_{\text{Finite-difference approximation}} - \underbrace{\frac{f^{(3)}(\xi)}{6}h^2}_{\text{Truncation error}} \tag{4.28}$$

Thus, if the two function values in the numerator of the finite-difference approximation have no roundoff error, the only error is due to truncation.
However, because we are using digital computers, the function values do include roundoff error as in

$$f(x_{i-1}) = \tilde{f}(x_{i-1}) + e_{i-1}$$

$$f(x_{i+1}) = \tilde{f}(x_{i+1}) + e_{i+1}$$

where the $\tilde{f}$'s are the rounded function values and the e's are the associated roundoff errors. Substituting these values into Eq. (4.28) gives

$$\underset{\text{True value}}{f'(x_i)} = \underbrace{\frac{\tilde{f}(x_{i+1})-\tilde{f}(x_{i-1})}{2h}}_{\text{Finite-difference approximation}} + \underbrace{\frac{e_{i+1}-e_{i-1}}{2h}}_{\text{Roundoff error}} - \underbrace{\frac{f^{(3)}(\xi)}{6}h^2}_{\text{Truncation error}}$$

We can see that the total error of the finite-difference approximation consists of a roundoff error that decreases with step size and a truncation error that increases with step size. Assuming that the absolute value of each component of the roundoff error has an upper bound of ε, the maximum possible value of the difference $e_{i+1} - e_{i-1}$ will be 2ε. Further, assume that the third derivative has a maximum absolute value of M. An upper bound on the absolute value of the total error can therefore be represented as

$$\text{Total error} = \left| f'(x_i) - \frac{\tilde{f}(x_{i+1})-\tilde{f}(x_{i-1})}{2h} \right| \leq \frac{\varepsilon}{h} + \frac{h^2 M}{6} \tag{4.29}$$

An optimal step size can be determined by differentiating Eq. (4.29), setting the result equal to zero and solving for

$$h_{opt} = \sqrt[3]{\frac{3\varepsilon}{M}} \tag{4.30}$$

Example 2.5. Roundoff and Truncation Errors in Numerical Differentiation

Problem Statement. In Example 4.4, we used a centered difference approximation of $O(h^2)$ to estimate the first derivative of the following function at $x = 0.5$,

$$f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x + 1.2$$

Perform the same computation starting with $h = 1$. Then progressively divide the step size by a factor of 10 to demonstrate how roundoff becomes dominant as the step size is reduced. Relate your results to Eq. (4.30). Recall that the true value of the derivative is -0.9125.

Solution. We can develop the following M-file to perform the computations and plot the results. Notice that we pass both the function and its analytical derivative as arguments:

```
function diffex(func,dfunc,x,n)
format long
dftrue=dfunc(x);
h=1;
H(1)=h;
D(1)=(func(x+h)-func(x-h))/(2*h);
E(1)=abs(dftrue-D(1));
for i=2:n
      h=h/10;
      H(i)=h;
      D(i)=(func(x+h)-func(x-h))/(2*h);
      E(i)=abs(dftrue-D(i));
end
L=[H' D' E']';
fprintf(' step size finite difference true error\n');
fprintf('%14.10f %16.14f %16.13f \n', L);
loglog(H, E), xlabel('Step Size'), ylabel('Error')
title('Plot of Error Versus Step Size')
format short
```

The M-file can then be run using the following commands:

```
» ff=@(x) -0.1*x^4-0.15*x^3-0.5*x^2-0.25*x+1.2;
» df=@(x) -0.4*x^3-0.45*x^2-x-0.25;
» diffex(ff,df,0.5,11)

   step size    finite difference     true error
 1.0000000000  -1.26250000000000  0.3500000000000
 0.1000000000  -0.91600000000000  0.0035000000000
 0.0100000000  -0.91253500000000  0.0000350000000
 0.0010000000  -0.91250035000001  0.0000003500000
 0.0001000000  -0.91250000349985  0.0000000034998
 0.0000100000  -0.91250000003318  0.0000000000332
 0.0000010000  -0.91250000000542  0.0000000000054
 0.0000001000  -0.91249999945031  0.0000000005497
 0.0000000100  -0.91250000333609  0.0000000033361
 0.0000000010  -0.91250001998944  0.0000000199894
 0.0000000001  -0.91250007550059  0.0000000755006
```

As depicted in Fig. 4.12, the results are as expected. At first, roundoff is minimal and the estimate is dominated by truncation error. Hence, as in Eq. (4.29), the total error drops by a factor of 100 each time we divide the step by 10. However, starting at about $h = 0.0001$, we see roundoff error begin to creep in and erode the rate at which the error diminishes. A minimum error is reached at $h = 10^{-6}$. Beyond this point, the error increases as roundoff dominates.

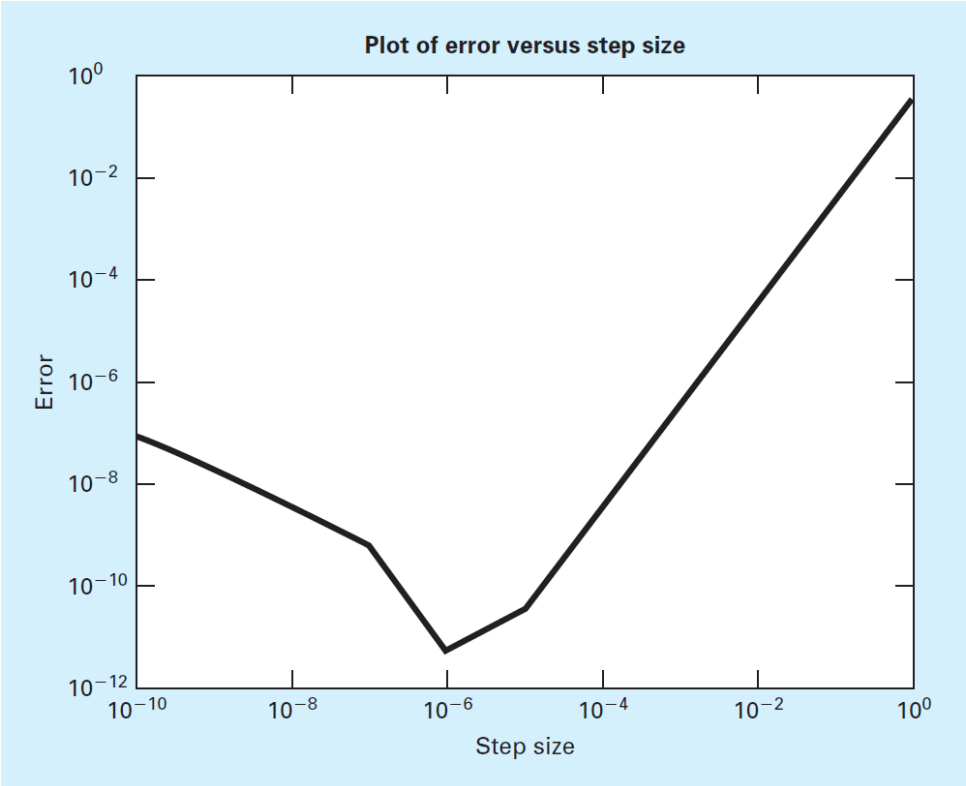
Because we are dealing with an easily differentiable function, we can also investigate whether these results are consistent with Eq. (4.30). First, we can estimate M by evaluating the function's third derivative as

$$M = \left|f^{(3)}(0.5)\right| = |-2.4(0.5) - 0.9| = 2.1$$

Because MATLAB has a precision of about 15 to 16 base-10 digits, a rough estimate of the upper bound on roundoff would be about $\varepsilon = 0.5 \times 10^{-16}$. Substituting these values into Eq. (4.30) gives

$$h_{opt} = \sqrt[3]{\frac{3(0.5 \times 10^{-16})}{2.1}} = 4.3 \times 10^{-6}$$

which is on the same order as the result of 1×10^{-6} obtained with MATLAB.



■

2.4.2. Control of Numerical Errors

For most practical cases, we do not know the exact error associated with numerical methods. The exception, of course, is when we know the exact solution, which makes our numerical approximations unnecessary. Therefore, for most engineering and scientific applications we must settle for some estimate of the error in our calculations.

There are no systematic and general approaches to evaluating numerical errors for all problems. In many cases error estimates are based on the experience and judgment of the engineer or scientist.

Although error analysis is to a certain extent an art, there are several practical programming guidelines we can suggest. First and foremost, avoid subtracting two nearly equal numbers. Loss of significance almost always occurs when this is done. Sometimes you can rearrange or reformulate the problem to avoid subtractive cancellation. If this is not possible, you may want to use extended-precision arithmetic. Furthermore, when adding and subtracting numbers, it is best to sort the numbers and work with the smallest numbers first. This avoids loss of significance.

Beyond these computational hints, one can attempt to predict total numerical errors using theoretical formulations. The Taylor series is our primary tool for analysis of such errors. Prediction of total numerical error is very complicated for even moderately sized problems and tends to be pessimistic. Therefore, it is usually attempted for only small-scale tasks.

The tendency is to push forward with the numerical computations and try to estimate the accuracy of your results. This can sometimes be done by seeing if the results satisfy some condition or equation as a check. Or it may be possible to substitute the results back into the original equation to check that it is actually satisfied.

Finally you should be prepared to perform numerical experiments to increase your awareness of computational errors and possible ill-conditioned problems. Such experiments may involve repeating the computations with a different step size or method and comparing the results. We may employ sensitivity analysis to see how our solution changes when we change model parameters or input values. We may want to try different numerical algorithms that have different theoretical foundations, are based on different computational strategies, or have different convergence properties and stability characteristics.

When the results of numerical computations are extremely critical and may involve loss of human life or have severe economic ramifications, it is appropriate to take special precautions. This may involve the use of two or more independent groups to solve the same problem so that their results can be compared.

The roles of errors will be a topic of concern and analysis in all sections of this book. We will leave these investigations to specific sections.

2.5. BLUNDERS, MODEL ERRORS, AND DATA UNCERTAINTY

Although the following sources of error are not directly connected with most of the numerical methods in this book, they can sometimes have great impact on the success of a modeling effort. Thus, they must always be kept in mind when applying numerical techniques in the context of real-world problems.

2.5.1. Blunders

We are all familiar with gross errors, or blunders. In the early years of computers, erroneous numerical results could sometimes be attributed to malfunctions of the computer itself. Today, this source of error is highly unlikely, and most blunders must be attributed to human imperfection.

Blunders can occur at any stage of the mathematical modeling process and can contribute to all the other components of error. They can be avoided only by sound knowledge of fundamental principles and by the care with which you approach and design your solution to a problem.

Blunders are usually disregarded in discussions of numerical methods. This is no doubt due to the fact that, try as we may, mistakes are to a certain extent unavoidable. However, we believe that there are a number of ways in which their occurrence can be minimized. In particular, the good programming habits that were outlined in Chap. 3 are extremely useful for mitigating programming blunders. In addition, there are usually simple ways to check whether a particular numerical method is working properly. Throughout this book, we discuss ways to check the results of numerical calculations.

2.5.2. Model Errors

Model errors relate to bias that can be ascribed to incomplete mathematical models. An example of a negligible model error is the fact that Newton's second law does not account for relativistic effects. This does not detract from the adequacy of the solution in Example 1.1 because these errors are minimal on the time and space scales associated with the bungee jumper problem. However, suppose that air resistance is not proportional to the square of the fall velocity, as in Eq. (1.7), but is related to velocity and other factors in a different way. If such were the case, both the analytical and numerical solutions obtained in Chap. 1 would be erroneous because of model error.You should be cognizant of this type of error

and realize that, if you are working with a poorly conceived model, no numerical method will provide adequate results.

2.5.3. Data Uncertainty

Errors sometimes enter into an analysis because of uncertainty in the physical data on which a model is based. For instance, suppose we wanted to test the bungee jumper model by having an individual make repeated jumps and then measuring his or her velocity after a specified time interval. Uncertainty would undoubtedly be associated with these measurements, as the parachutist would fall faster during some jumps than during others. These errors can exhibit both inaccuracy and imprecision. If our instruments consistently underestimate or overestimate the velocity, we are dealing with an inaccurate, or biased, device. On the other hand, if the measurements are randomly high and low, we are dealing with a question of precision.

Measurement errors can be quantified by summarizing the data with one or more wellchosen statistics that convey as much information as possible regarding specific characteristics of the data. These descriptive statistics are most often selected to represent (1) the location of the center of the distribution of the data and (2) the degree of spread of the data. As such, they provide a measure of the bias and imprecision, respectively.We will return to the topic of characterizing data uncertainty when we discuss regression in Part Four.

Although you must be cognizant of blunders, model errors, and uncertain data, the numerical methods used for building models can be studied, for the most part, independently of these errors. Therefore, for most of this book, we will assume that we have not made gross errors, we have a sound model, and we are dealing with error-free measurements. Under these conditions, we can study numerical errors without complicating factors.

PROBLEMS

4.1 The "divide and average" method, an old-time method for approximating the square root of any positive number a, can be formulated as

$$x = \frac{x + a/x}{2}$$

Write a well-structured function to implement this algorithm based on the algorithm outlined in Fig. 4.2.

4.2 Convert the following base-2 numbers to base 10: **(a)** 1011001, **(b)** 0.01011, and **(c)** 110.01001.

4.3 Convert the following base-8 numbers to base 10: 61,565 and 2.71.

4.4 For computers, the machine epsilon ε can also be thought of as the smallest number that when added to one gives a number greater than 1. An algorithm based on this idea can be developed as
Step 1: Set $\varepsilon = 1$
Step 2: If $1 + \varepsilon$ is less than or equal to 1, then go to Step 5. Otherwise go to Step 3.
Step 3: $\varepsilon = \varepsilon/2$
Step 4: Return to Step 2
Step 5: $\varepsilon = 2 \times \varepsilon$

Write your own M-file based on this algorithm to determine the machine epsilon. Validate the result by comparing it with the value computed with the built-in function `eps`.

4.5 In a fashion similar to Prob. 4.4, develop your own M-file to determine the smallest positive real number used in MATLAB. Base your algorithm on the notion that your computer will be unable to reliably distinguish between zero and a quantity that is smaller than this number. Note that the result you obtain will differ from the value computed with `realmin`.Challenge question: Investigate the results by taking the base-2 logarithm of the number generated by your code and those obtained with `realmin`.

4.6 Although it is not commonly used, MATLAB allows numbers to be expressed in single precision. Each value is stored in 4 bytes with 1 bit for the sign, 23 bits for the mantissa, and 8 bits for the signed exponent. Determine the smallest and largest positive floating-point numbers as well as the machine epsilon for single precision representation. Note that the exponents range from -126 to 127.

4.7 For the hypothetical base-10 computer in Example 4.2, prove that the machine epsilon is 0.05.

4.8 The derivative of $f(x) = 1/(1-3x^2)$ is given by

$$\frac{6x}{(1-3x^2)^2}$$

Do you expect to have difficulties evaluating this function at x = 0.577? Try it using 3- and 4-digit arithmetic with chopping.

4.9 (a) Evaluate the polynomial

$$y = x^3 - 7x^2 + 8x - 0.35$$

at x = 1.37. Use 3-digit arithmetic with chopping. Evaluate the percent relative error.

(b) Repeat **(a)** but express y as

$$y = ((x-7)x+8)x - 0.35$$

Evaluate the error and compare with part **(a)**.

4.10 The following infinite series can be used to approximate e^x:

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$$

(a) Prove that this Maclaurin series expansion is a special case of the Taylor series expansion (Eq. 4.13) with $x_i = 0$ and $h = x$.

(b) Use the Taylor series to estimate $f(x) = e^{-x}$ at $x_{i+1} = 1$ for $x_i = 0.25$. Employ the zero-, first-, second-, and third-order versions and compute the $|\varepsilon_t|$ for each case.

4.11 The Maclaurin series expansion for *cosx* is

$$cosx = 1 - \frac{x^2}{2} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \ldots$$

Starting with the simplest version, $cosx = 1$, add terms one at a time to estimate $cos(\pi/4)$. After each new term is added, compute the true and approximate percent relative errors. Use your pocket calculator or MATLAB to determine the true value. Add terms until the absolute value of the approximate error estimate falls below an error criterion conforming to two significant figures.

4.12 Perform the same computation as in Prob. 4.11, but use the Maclaurin series expansion for the *sinx* to estimate $sin(\pi/4)$.

$$sinx = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

4.13 Use zero- through third-order Taylor series expansions to predict $f(3)$ for

$$f(x) = 25x^3 - 6x^2 + 7x - 88$$

using a base point at $x = 1$. Compute the true percent relative error ε_t for each approximation.

4.14 Prove that Eq. (4.11) is exact for all values of x if $f(x) = ax^2 + bx + c$

4.15 Use zero- through fourth-order Taylor series expansions to predict $f(2)$ for $f(x) = \ln x$ using a base point at $x = 1$. Compute the true percent relative error ε_t for each approximation. Discuss the meaning of the results.

4.16 Use forward and backward difference approximations of $O(h)$ and a centered difference approximation of $O(h^2)$ to estimate the first derivative of the function examined in Prob. 4.13. Evaluate the derivative at $x = 2$ using a step size of $h = 0.25$. Compare your results with the true value of the derivative. Interpret your results on the basis of the remainder term of the Taylor series expansion.

4.17 Use a centered difference approximation of $O(h^2)$ to estimate the second derivative of the function examined in Prob. 4.13. Perform the evaluation at $x = 2$ using step sizes of $h = 0.2$ and 0.1. Compare your estimates with the true value of the second derivative. Interpret your results on the basis of the remainder term of the Taylor series expansion.

4.18 If $|x| < 1$ it is known that

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \ldots$$

Repeat Prob. 4.11 for this series for $x = 0.1$.

4.19 To calculate a planet's space coordinates, we have to solve the function

$$f(x) = x - 1 - 0.5sinx$$

Let the base point be $a = x_i = \pi/2$ on the interval $[0, \pi]$. Determine the highest-order Taylor series expansion resulting in a maximum error of 0.015 on the specified interval. The error is equal to the absolute value of the difference between the given function and the specific Taylor series expansion. (Hint: Solve graphically.)

4.20 Consider the function $f(x) = x^3 - 2x + 4$ on the interval [-2, 2] with $h = 0.25$. Use the forward, backward, and centered finite difference approximations for the first and second derivatives so as to graphically illustrate which approximation is most accurate. Graph all three first-derivative finite difference approximations along with the theoretical, and do the same for the second derivative as well.

4.21 Derive Eq. (4.30).

4.22 Repeat Example 4.5, but for $f(x) = cos(x)$ at $x = \pi/6$.

4.23 Repeat Example 4.5, but for the forward divided difference (Eq. 4.21).

4.24 One common instance where subtractive cancellation occurs involves finding the roots of a parabola, $ax^2 + bx + c$, with the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

For cases where $b^2 >> 4ac$, the difference in the numerator can be very small and roundoff errors can occur. In such cases, an alternative formulation can be used to minimize subtractive cancellation:

$$x = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$$

Use 5-digit arithmetic with chopping to determine the roots of the following equation with both versions of the quadratic formula.

$$x^2 - 5000.002x + 10$$

Part II

Roots and Optimization

2.6. OVERVIEW

Years ago, you learned to use the quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{PT2.1}$$

to solve

$$f(x) = ax^2 + bx + c = 0 \tag{PT2.2}$$

The values calculated with Eq. (PT2.1) are called the "roots" of Eq. (PT2.2). They represent the values of x that make Eq. (PT2.2) equal to zero. For this reason, roots are sometimes called the *zeros* of the equation.

Although the quadratic formula is handy for solving Eq. (PT2.2), there are many other functions for which the root cannot be determined so easily. Before the advent of digital computers, there were a number of ways to solve for the roots of such equations. For some cases, the roots could be obtained by direct methods, as with Eq. (PT2.1). Although there were equations like this that could be solved directly, there were many more that could not. In such instances, the only alternative is an approximate solution technique.

One method to obtain an approximate solution is to plot the function and determine where it crosses the x axis. This point, which represents the x value for which $f(x) = 0$, is the root. Although graphical methods are useful for obtaining rough estimates of roots, they are limited because of their lack of precision. An alternative approach is to use *trial* and *error*. This "technique" consists of guessing a value of x and evaluating whether $f(x)$ is zero. If not (as is almost always the case), another guess is made, and $f(x)$ is again evaluated to determine whether the new value provides a better estimate of the root. The process is repeated until a guess results in an $f(x)$ that is close to zero.

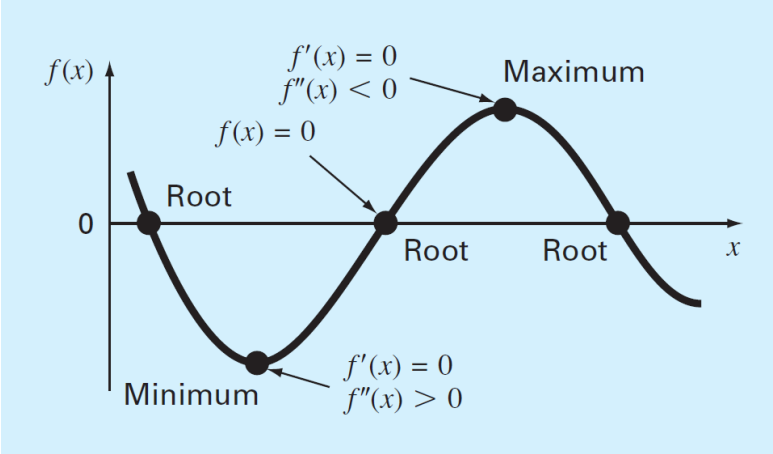


Figure 2.12: A function of a single variable illustrating the difference between roots and optima.

Such haphazard methods are obviously inefficient and inadequate for the requirements of engineering and science practice. Numerical methods represent alternatives that are also approximate but employ systematic strategies to home in on the true root. As elaborated in the following pages, the combination of these systematic methods and computers makes the solution of most applied roots-of-equations problems a simple and efficient task.

Besides roots, another feature of interest to engineers and scientists are a function's minimum and maximum values. The determination of such optimal values is referred to as *optimization*. As you learned in calculus, such solutions can be obtained analytically by determining the value at which the function is flat; that is, where its derivative is zero. Although such analytical solutions are sometimes feasible, most practical optimization problems require numerical, computer solutions. From a numerical standpoint, such optimization methods are similar in spirit to the root-location methods we just discussed. That is, both involve guessing and searching for a location on a function. The fundamental difference between the two types of problems is illustrated in Figure PT2.1. Root location involves searching for the location where the function equals zero. In contrast, optimization involves searching for the function's extreme points.

2.7. PART ORGANIZATION

The first two chapters in this part are devoted to root location. *Chapter 5* focuses on *bracketing methods* for finding roots. These methods start with guesses that bracket, or contain, the root and then systematically reduce the width of the bracket. Two specific methods are covered: *bisection* and *false position*. Graphical methods are used to provide visual insight into

the techniques. Error formulations are developed to help you determine how much computational effort is required to estimate the root to a prespecified level of precision.

Chapter 6 covers open methods. These methods also involve systematic trial-and-error iterations but do not require that the initial guesses bracket the root. We will discover that these methods are usually more computationally efficient than bracketing methods but that they do not always work. We illustrate several open methods including the *fixed-point iteration, Newton-Raphson*, and *secant* methods.

Following the description of these individual open methods, we then discuss a hybrid approach called *Brent's root-finding* method that exhibits the reliability of the bracketing methods while exploiting the speed of the open methods. As such, it forms the basis for MATLAB's root-finding function, `fzero`. After illustrating how `fzero` can be used for engineering and scientific problems solving, Chap. 6 ends with a brief discussion of special methods devoted to finding the roots of *polynomials*. In particular, we describe MATLAB's excellent built-in capabilities for this task.

Chapter 7 deals with *optimization*. First, we describe two bracketing methods, *goldensection search* and *parabolic interpolation*, for finding the optima of a function of a single variable. Then, we discuss a robust, hybrid approach that combines golden-section search and quadratic interpolation. This approach, which again is attributed to Brent, forms the basis for MATLAB's one-dimensional root-finding function: `fminbnd`. After describing and illustrating `fminbnd`, the last part of the chapter provides a brief description of optimization of multidimensional functions. The emphasis is on describing and illustrating the use of MATLAB's capability in this area: the `fminsearch` function. Finally, the chapter ends with an example of how MATLAB can be employed to solve optimization problems in engineering and science.

Chapter 3

Roots: Bracketing Methods

CHAPTER OBJECTIVES

The primary objective of this chapter is to acquaint you with bracketing methods for finding the root of a single nonlinear equation. Specific objectives and topics covered are

- Understanding what roots problems are and where they occur in engineering and science.
- Knowing how to determine a root graphically.
- Understanding the incremental search method and its shortcomings.
- Knowing how to solve a roots problem with the bisection method.
- Knowing how to estimate the error of bisection and why it differs from error estimates for other types of root-location algorithms.
- Understanding false position and how it differs from bisection.

YOU'VE GOT A PROBLEM

Medical studies have established that a bungee jumper's chances of sustaining a significant vertebrae injury increase significantly if the free-fall velocity exceeds 36 m/s after 4 s of free fall. Your boss at the bungee-jumping company wants you to determine the mass at which this criterion is exceeded given a drag coefficient of 0.25 kg/m.

You know from your previous studies that the following analytical solution can be used to predict fall velocity as a function of time:

$$v(t) = \sqrt{\frac{gm}{c_d}}\tanh\left(\sqrt{\frac{gc_d}{m}}t\right) \tag{5.1}$$

Try as you might, you cannot manipulate this equation to explicitly solve for m—that is, you cannot isolate the mass on the left side of the equation.

An alternative way of looking at the problem involves subtracting v(t) from both sides to give a new function:

$$f(m) = \sqrt{\frac{gm}{c_d}}\tanh\left(\sqrt{\frac{gc_d}{m}}t\right) - v(t) \tag{5.2}$$

Now we can see that the answer to the problem is the value of m that makes the function equal to zero. Hence, we call this a "roots" problem. This chapter will introduce you to how the computer is used as a tool to obtain such solutions.

3.1. ROOTS IN ENGINEERING AND SCIENCE

Although they arise in other problem contexts, roots of equations frequently occur in the area of design. Table 5.1 lists a number of fundamental principles that are routinely used in design work. As introduced in Chap. 1, mathematical equations or models derived from these principles are employed to predict dependent variables as a function of independent variables, forcing functions, and parameters. Note that in each case, the dependent variables reflect the state or performance of the system, whereas the parameters represent its properties or composition.

An example of such a model is the equation for the bungee jumper's velocity. If the parameters are known, Eq. (5.1) can be used to predict the jumper's velocity. Such computations can be performed directly because v is expressed *explicitly* as a function of the model parameters. That is, it is isolated on one side of the equal sign.

However, as posed at the start of the chapter, suppose that we had to determine the mass for a jumper with a given drag coefficient to attain a prescribed velocity in a set time period. Although Eq. (5.1) provides a mathematical

representation of the interrelationship among the model variables and parameters, it cannot be solved explicitly for mass. In such cases, m is said to be *implicit*.

Fundamental Principle	Dependent Variable	Independent Variable	Parameters
Heat balance	Temperature	Time and position	Thermal properties of material, system geometry
Mass balance	Concentration or quantity of mass	Time and position	Chemical behavior of material, mass transfer, system geometry
Force balance	Magnitude and direction of forces	Time and position	Strength of material, structural properties, system geometry
Energy balance	Changes in kinetic and potential energy	Time and position	Thermal properties, mass of material, system geometry
Newton's laws of motion	Acceleration, velocity, or location	Time and position	Mass of material, system geometry, dissipative parameters
Kirchhoff's laws	Currents and voltages	Time	Electrical properties (resistance, capacitance, inductance)

This represents a real dilemma, because many design problems involve specifying the properties or composition of a system (as represented by its parameters) to ensure that it performs in a desired manner (as represented by its variables). Thus, these problems often require the determination of implicit parameters.

The solution to the dilemma is provided by numerical methods for roots of equations. To solve the problem using numerical methods, it is conventional to reexpress Eq. (5.1) by subtracting the dependent variable v from both sides of the equation to give Eq. (5.2). The value of m that makes $f(m) = 0$ is, therefore, the root of the equation. This value also represents the mass that solves the design problem.

The following pages deal with a variety of numerical and graphical methods for determining roots of relationships such as Eq. (5.2). These techniques can be applied to many other problems confronted routinely in engineering and science.

3.2. GRAPHICAL METHODS

A simple method for obtaining an estimate of the root of the equation $f(x) = 0$ is to make a plot of the function and observe where it crosses the x axis. This point, which represents the x value for which $f(x) = 0$, provides a rough approximation of the root.

Example 3.1. The Graphical Approach

Problem Statement. Use the graphical approach to determine the mass of the bungee jumper with a drag coefficient of 0.25 kg/m to have a velocity of 36 m/s after 4 s of free fall. Note: The acceleration of gravity is $9.81 m/s^2$.

Solution. The following MATLAB session sets up a plot of Eq. (5.2) versus mass:

```
» cd = 0.25; g = 9.81; v = 36; t = 4;
» mp = linspace(50,200);
» fp = sqrt(g*mp/cd).*tanh(sqrt(g*cd./mp)*t)-v;
» plot(mp,fp),grid
```

The function crosses the m axis between 140 and 150 kg. Visual inspection of the plot provides a rough estimate of the root of 145 kg (about 320 lb). The validity of the graphical estimate can be checked by substituting it into Eq. (5.2) to yield

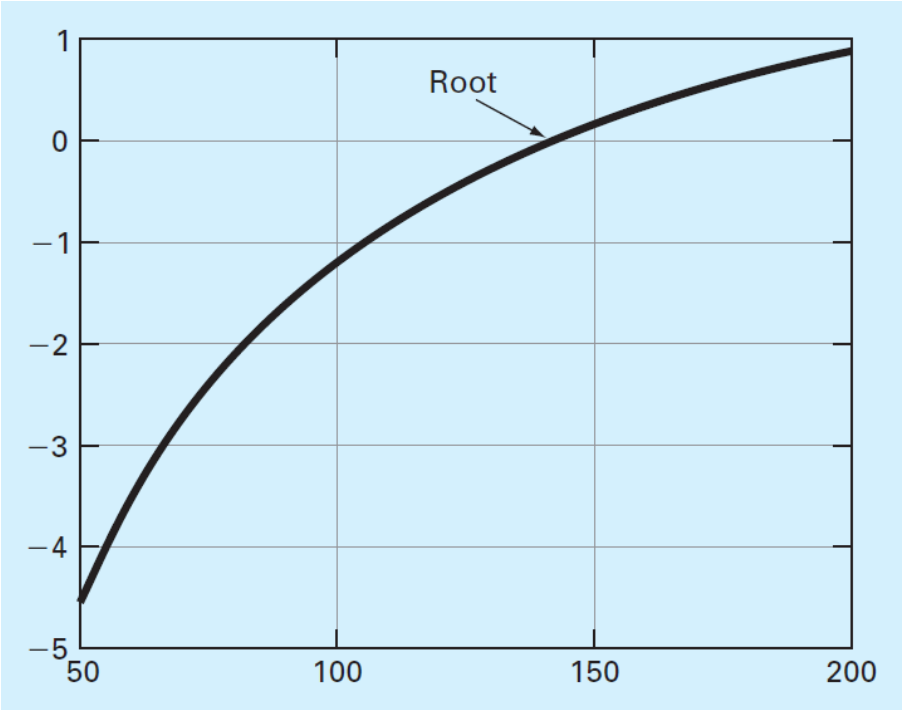
```
» sqrt(g*145/cd)*tanh(sqrt(g*cd/145)*t)-v
ans =
      0.0456
```

which is close to zero. It can also be checked by substituting it into Eq. (5.1) along with the parameter values from this example to give

```
» sqrt(g*145/cd)*tanh(sqrt(g*cd/145)*t)
ans =
      36.0456
```

which is close to the desired fall velocity of 36 m/s.

■



Graphical techniques are of limited practical value because they are not very precise. However, graphical methods can be utilized to obtain rough estimates of roots. These estimates can be employed as starting guesses for numerical methods discussed in this chapter.

Aside from providing rough estimates of the root, graphical interpretations are useful for understanding the properties of the functions and anticipating the pitfalls of the numerical methods. For example, Fig. 5.1 shows a number of ways in which roots can occur (or be absent) in an interval prescribed by a lower bound x_l and an upper bound x_u. Figure 5.1b depicts the case where a single root is bracketed by negative and positive values of f (x). However, Fig. 5.1d, where $f(x_l)$ and $f(x_u)$ are also on opposite sides of the x axis, shows three roots occurring within the interval. In general, if $f(x_l)$ and $f(x_u)$ have opposite signs, there are an odd number of roots in the interval. As indicated by Fig. 5.1a and c, $iff(x_l)$ and $f(x_u)$ have the same sign, there are either no roots or an even number of roots between the values.

Although these generalizations are usually true, there are cases where they do not hold. For example, functions that are tangential to the x axis (Fig. 5.2a) and discontinuous functions (Fig. 5.2b) can violate these principles. An example of a function that is tangential to the axis is the cubic equation $f(x) = (x-2)(x-2)(x-4)$. Notice that $x = 2$ makes two terms in this polynomial equal to zero. Mathematically, $x = 2$ is called a *multiple root*. Although they are beyond the scope of this book, there are special techniques that are expressly designed to locate multiple roots (Chapra and Canale, 2010).

The existence of cases of the type depicted in Fig. 5.2 makes it difficult to develop foolproof computer algorithms guaranteed to locate all the roots in an interval. However, when used in conjunction with graphical approaches, the methods described in the following sections are extremely useful for solving many problems confronted routinely by engineers, scientists, and applied mathematicians.

3.3. BRACKETING METHODS AND INITIAL GUESSES

If you had a roots problem in the days before computing, you'd often be told to use "trial and error" to come up with the root. That is, you'd repeatedly make guesses until the function was sufficiently close to zero. The process was greatly facilitated by the advent of software tools such as spreadsheets.
By allowing you to make many guesses rapidly, such tools can actually make the trial-and-error approach attractive for some problems.

But, for many other problems, it is preferable to have methods that come up with the correct answer automatically. Interestingly, as with trial and error, these approaches require an initial "guess" to get started. Then they systematically home in on the root in an iterative fashion.

The two major classes of methods available are distinguished by the type of initial guess. They are

- *Bracketing methods*. As the name implies, these are based on two initial guesses that "bracket" the root—that is, are on either side of the root.
- *Open methods*. These methods can involve one or more initial guesses, but there is no need for them to bracket the root.

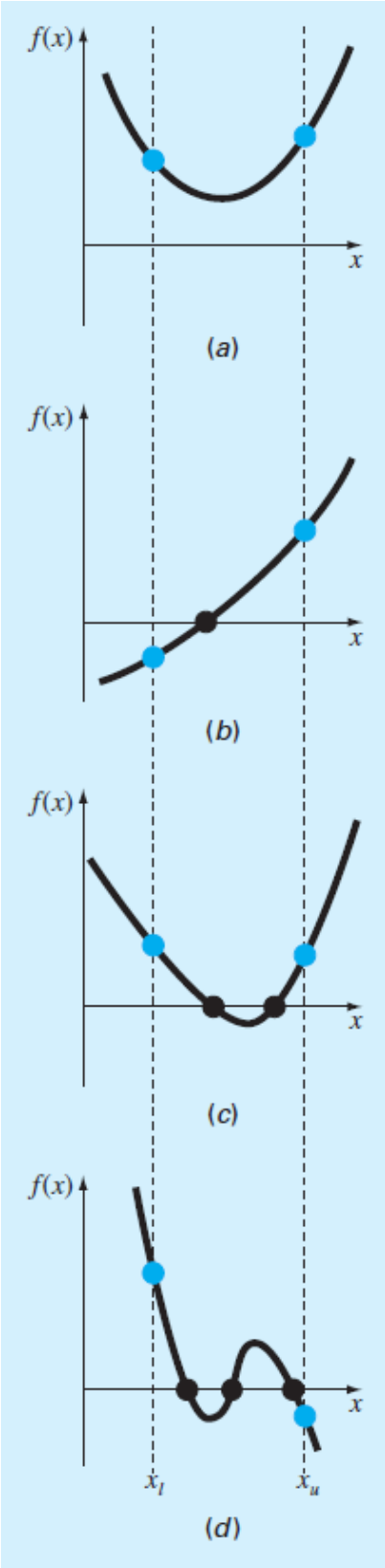


Figure 3.1: Illustration of a number of general ways that a root may occur in an interval prescribed by a lower bound x_l and an upper bound x_u. Parts (a) and (c) indicate that if both $f(x_l)$ and $f(x_u)$ have the same sign, either there will be no roots or there will be an even number of roots within the interval. Parts (b) and (d) indicate that if the function has different signs at the end points, there will be an odd number of roots in the interval.

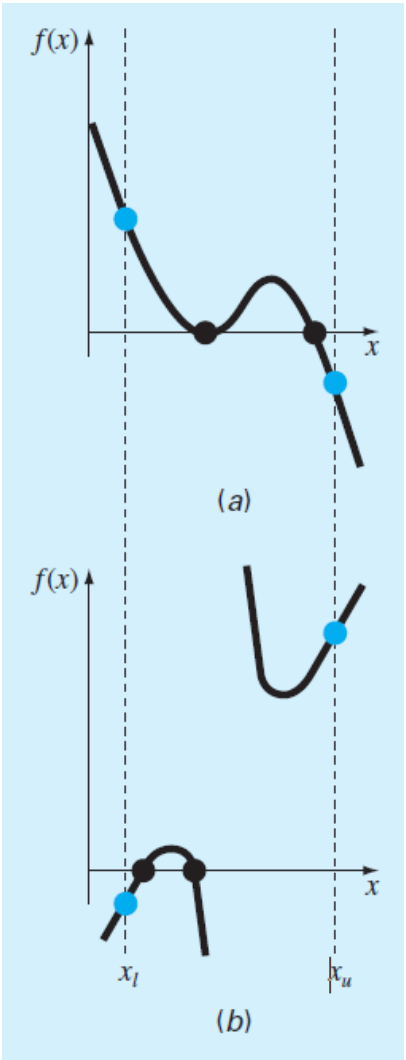


Figure 3.2: Illustration of some exceptions to the general cases depicted in Fig. 5.1. (a) Multiple roots that occur when the function is tangential to the x axis. For this case, although the end points are of opposite signs, there are an even number of axis interceptions for the interval. (b) Discontinuous functions where end points of opposite sign bracket an even number of roots. Special strategies are required for determining the roots for these cases.

For well-posed problems, the bracketing methods always work but converge slowly (i.e., they typically take more iterations to home in on the answer). In contrast, the open methods do not always work (i.e., they can diverge), but when they do they usually converge quicker.

In both cases, initial guesses are required. These may naturally arise from the physical context you are analyzing. However, in other cases, good initial guesses may not be obvious. In such cases, automated approaches to obtain guesses would be useful. The following section describes one such approach, the incremental search.

3.3.1. Incremental Search

When applying the graphical technique in Example 5.1, you observed that $f(x)$ changed sign on opposite sides of the root. In general, if $f(x)$ is real and continuous in the interval from x_l to x_u and $f(x_l)$ and $f(x_u)$ have opposite signs, that is,

$$f(x_l)f(x_u) < 0 \tag{5.3}$$

then there is at least one real root between x_l and x_u .

Incremental search methods capitalize on this observation by locating an interval where the function changes sign. A potential problem with an incremental search is the choice of the increment length. If the length is too small, the search can be very time consuming. On the other hand, if the length is too great, there is a possibility that closely spaced roots might be missed (Fig. 5.3). The problem is compounded by the possible existence of multiple roots.

An M-file can be developed[1] that implements an incremental search to locate the roots of a function `func` within the range from `xmin` to `xmax` (Fig. 5.4). An optional argument ns allows the user to specify the number of intervals within the range. If ns is omitted, it is automatically set to 50. A `for` loop is used to step through each interval. In the event that a sign change occurs, the upper and lower bounds are stored in an array `xb`.

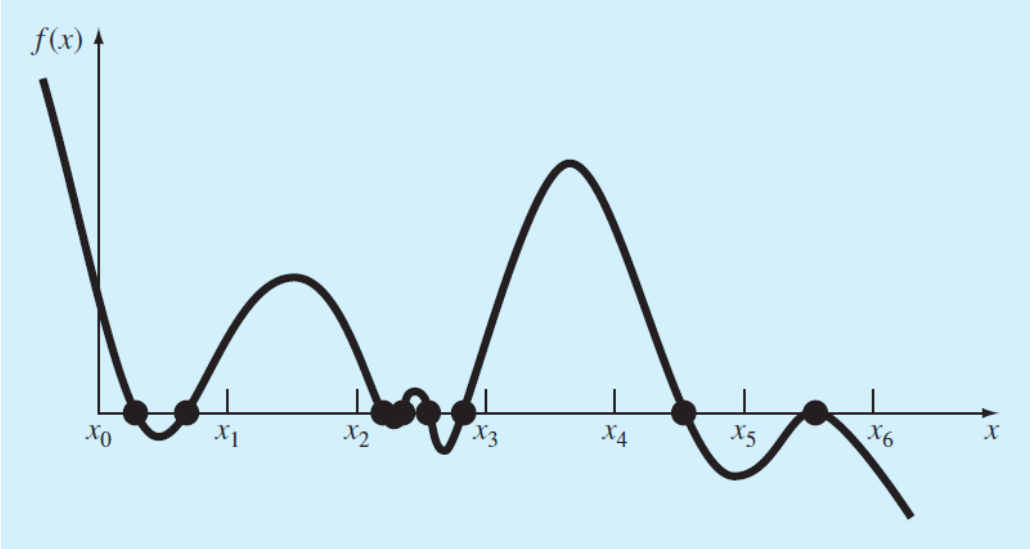


Figure 3.3: Cases where roots could be missed because the incremental length of the search procedure is too large. Note that the last root on the right is multiple and would be missed regardless of the increment length.

[1]This function is a modified version of an M-file originally presented by Recktenwald (2000).

```
function xb = incsearch(func,xmin,xmax,ns)
% incsearch: incremental search root locator
%   xb = incsearch(func,xmin,xmax,ns):
%      finds brackets of x that contain sign changes
%      of a function on an interval
% input:
%   func = name of function
%   xmin, xmax = endpoints of interval
%   ns = number of subintervals (default = 50)
% output:
%   xb(k,1) is the lower bound of the kth sign change
%   xb(k,2) is the upper bound of the kth sign change
%   If no brackets found, xb = [].
if nargin < 3, error('at least 3 arguments required'), end
if nargin < 4, ns = 50; end %if ns blank set to 50
% Incremental search
x = linspace(xmin,xmax,ns);
f = func(x);
nb = 0; xb = []; %xb is null unless sign change detected
for k = 1:length(x)-1
  if sign(f(k)) ~= sign(f(k+1)) %check for sign change
    nb = nb + 1;
    xb(nb,1) = x(k);
    xb(nb,2) = x(k+1);
  end
end
if isempty(xb)    %display that no brackets were found
  disp('no brackets found')
  disp('check interval or increase ns')
else
  disp('number of brackets:') %display number of brackets
  disp(nb)
end
```

Figure 3.4: An M-file to implement an incremental search.

Example 3.2. Incremental Search

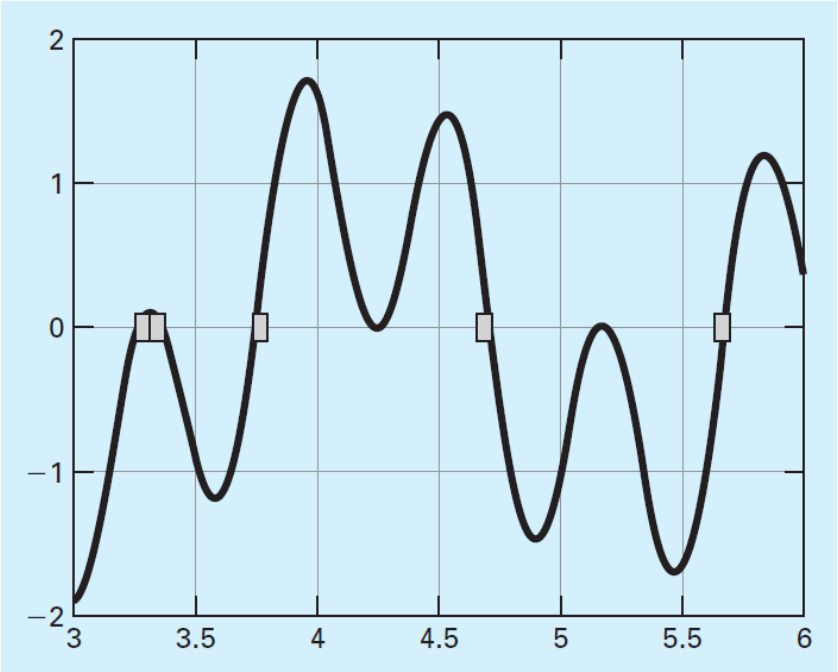
Problem Statement. Use the M-file incsearch (Fig. 5.4) to identify brackets within the interval [3, 6] for the function:

$$f(x) = sin(10x) + cos(3x) \tag{5.4}$$

Solution. The MATLAB session using the default number of intervals (50) is

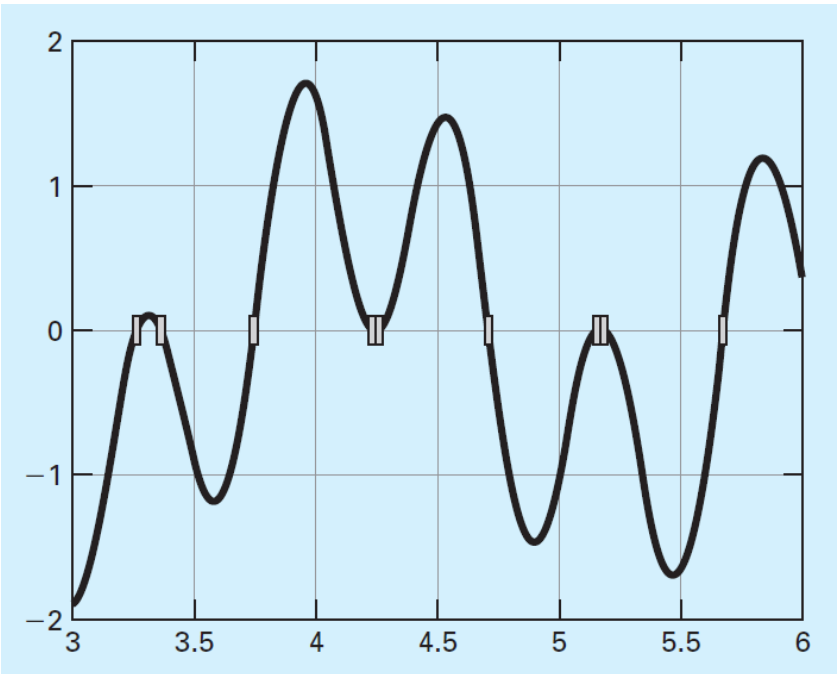
```
» incsearch(@(x) sin(10*x)+cos(3*x),3,6)
number of brackets:
      5
ans =
      3.2449       3.3061
      3.3061       3.3061
      3.7347       3.7959
      4.6531       4.7143
      5.6327       5.6939
```

A plot of Eq. (5.4) along with the root locations is shown here.



Although five sign changes are detected, because the subintervals are too wide, the function misses possible roots at $x \cong 4.25$ and 5.2. These possible roots look like they might be double roots. However, by using the zoom in tool, it is clear that each represents two real roots that are very close together. The function can be run again with more subintervals with the result that all nine sign changes are located

```
      » incsearch(@(x) sin(10*x)+cos(3*x),3,6,100)
number of brackets:
      9
ans =
      3.2424       3.2727
      3.3636       3.3939
      3.7273       3.7576
      4.2121       4.2424
      4.2424       4.2727
      4.6970       4.7273
      5.1515       5.1818
      5.1818       5.2121
      5.6667       5.6970
```



The foregoing example illustrates that brute-force methods such as incremental search are not foolproof. You would be wise to supplement such automatic techniques with any other information that provides insight into the location of the roots. Such information can be found by plotting the function and through understanding the physical problem from which the equation originated. ■

3.4. BISECTION

The *bisection method* is a variation of the incremental search method in which the interval is always divided in half. If a function changes sign over an interval, the function value at the midpoint is evaluated. The location of the root is then determined as lying within the subinterval where the sign change occurs. The subinterval then becomes the interval for the next iteration. The process is repeated until the root is known to the required precision. A graphical depiction of the method is provided in Fig. 5.5. The following example goes through the actual computations involved in the method.

Example 3.3. The Bisection Method

Problem Statement. Use bisection to solve the same problem approached graphically in Example 5.1.

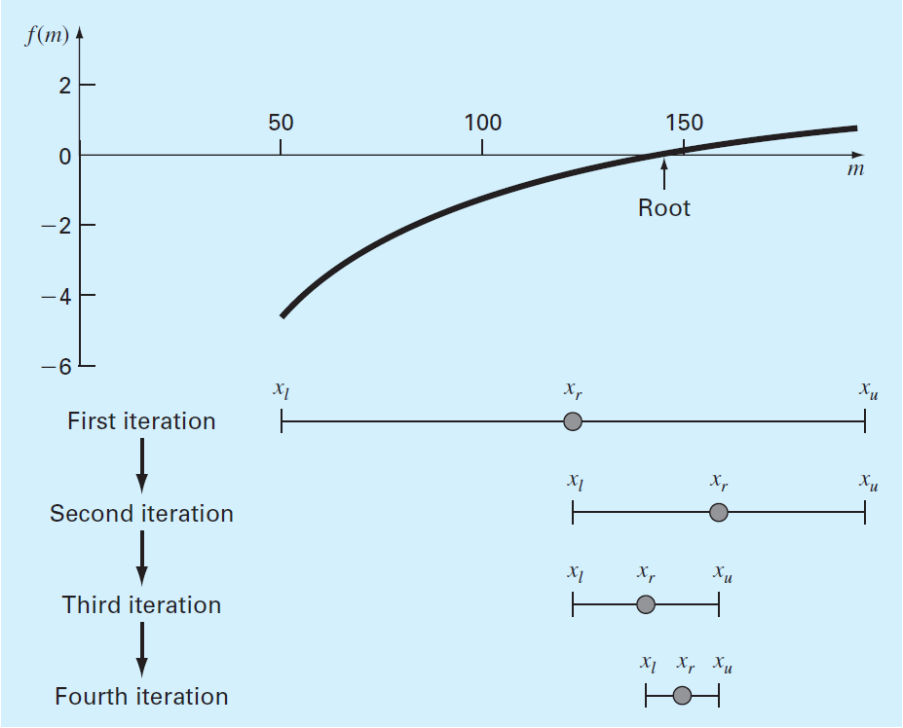


Figure 3.5: A graphical depiction of the bisection method. This plot corresponds to the first four iterations from Example 5.3.

Solution. The first step in bisection is to guess two values of the unknown (in the present problem, m) that give values for $f(m)$ with different signs. From the graphical solution in Example 5.1, we can see that the function changes sign between values of 50 and 200. The plot obviously suggests better initial guesses, say 140 and 150, but for illustrative purposes let's assume we don't have the benefit of the plot and have made conservative guesses. Therefore, the initial estimate of the root x_r lies at the midpoint of the interval

$$x_r = \frac{50+200}{2} = 125$$

Note that the exact value of the root is 142.7376. This means that the value of 125 calculated here has a true percent relative error of

$$|\varepsilon_t| = \left|\frac{142.7376-125}{142.7376}\right| \times 100\% = 12.43\%$$

Next we compute the product of the function value at the lower bound and at the midpoint:

$$f(50)f(125) = -4.579(-0.409) = 1.871$$

which is greater than zero, and hence no sign change occurs between the lower bound and the midpoint. Consequently, the root must be located in the upper interval between 125 and 200. Therefore, we create a new interval by redefining the lower bound as 125.

At this point, the new interval extends from $x_l = 125$ to $x_u = 200$. Arevised root estimate can then be calculated as

$$x_r = \frac{125+200}{2} = 162.5$$

which represents a true percent error of $|\varepsilon_t| = 13.85\%$. The process can be repeated to obtain refined estimates. For example,

$$f(125)f(162.5) = -0.409(0.359) = -0.147$$

Therefore, the root is now in the lower interval between 125 and 162.5. The upper bound is redefined as 162.5, and the root estimate for the third iteration is calculated as

$$x_r = \frac{125+162.5}{2} = 143.75$$

which represents a percent relative error of $\varepsilon_t = 0.709\%$. The method can be repeated until the result is accurate enough to satisfy your needs.

■

We ended Example 5.3 with the statement that the method could be continued to obtain a refined estimate of the root. We must now develop an objective criterion for deciding when to terminate the method.

An initial suggestion might be to end the calculation when the error falls below some prespecified level. For instance, in Example 5.3, the true relative error dropped from 12.43 to 0.709% during the course of the computation. We might decide that we should terminate when the error drops below, say, 0.5%. This strategy is flawed because the error estimates in the example were based on knowledge of the true root of the function. This would not be the case in an actual situation because there would be no point in using the method if we already knew the root.

Therefore, we require an error estimate that is not contingent on foreknowledge of the root. One way to do this is by estimating an approximate percent relative error as in [recall Eq. (4.5)]

$$|\varepsilon_a| = \left|\frac{x_r^{new} - x_r^{old}}{x_r^{new}}\right| 100\% \tag{5.5}$$

where x_r^{new} is the new root for the present iteration and x_r^{old} is the root from the previous iteration. When ε_a becomes less than a prespecified stopping criterion ε_s, the computation is terminated.

Example 3.4. Error Estimates for Bisection

Problem Statement. Continue Example 5.3 until the approximate error falls below a stopping criterion of $\varepsilon_s = 0.5\%$. Use Eq. (5.5) to compute the errors.

Solution. The results of the first two iterations for Example 5.3 were 125 and 162.5. Substituting these values into Eq. (5.5) yields

$$|\varepsilon_a| = \left|\frac{162.5 - 125}{162.5}\right| 100\% = 23.08\%$$

Recall that the true percent relative error for the root estimate of 162.5 was 13.85%. Therefore, $|\varepsilon_a|$ is greater than ε_t. This behavior is manifested for the other iterations:

Iteration	x_l	x_u	x_r	$\|\varepsilon_a\|$ (%)	$\|\varepsilon_t\|$ (%)
1	50	200	125		12.43
2	125	200	162.5	23.08	13.85
3	125	162.5	143.75	13.04	0.71
4	125	143.75	134.375	6.98	5.86
5	134.375	143.75	139.0625	3.37	2.58
6	139.0625	143.75	141.4063	1.66	0.93
7	141.4063	143.75	142.5781	0.82	0.11
8	142.5781	143.75	143.1641	0.41	0.30

Thus after eight iterations $|\varepsilon_a|$ finally falls below $\varepsilon_s = 0.5\%$, and the computation can be terminated.

These results are summarized in Fig. 5.6. The "ragged" nature of the true error is due to the fact that, for bisection, the true root can lie anywhere within the bracketing interval. The true and approximate errors are far apart when the interval happens to be centered on the true root. They are close when the true root falls at either end of the interval.

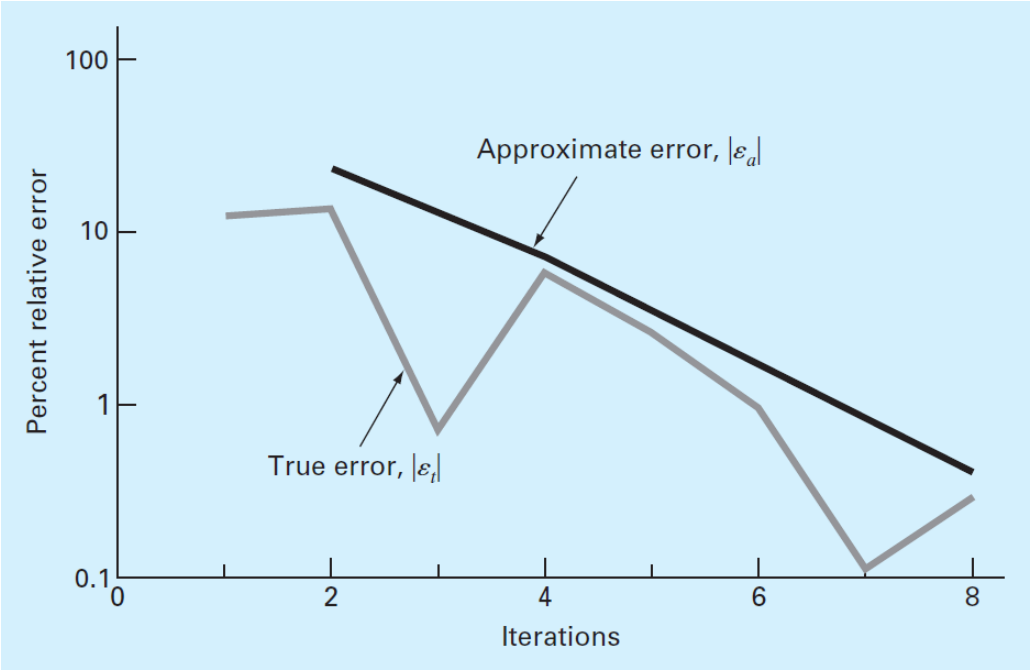


Figure 3.6: Errors for the bisection method. True and approximate errors are plotted versus the number of iterations.

■

Although the approximate error does not provide an exact estimate of the true error, Fig. 5.6 suggests that $|\varepsilon_a|$ captures the general downward trend of ε_t. In addition, the plot exhibits the extremely attractive characteristic that ε_a is always greater than ε_t. Thus, when ε_a falls below ε_s , the computation could be terminated with confidence that the root is known to be at least as accurate as the prespecified acceptable level.

While it is dangerous to draw general conclusions from a single example, it can be demonstrated that ε_a will always be greater than ε_t for bisection. This is due to the fact that each time an approximate root is located using bisection as $x_r = (x_l + x_u)/2$, we know that the true root lies somewhere within an interval of $\Delta x = x_u - x_l$. Therefore, the root must lie within $\pm\Delta x/2$ of our estimate. For instance, when Example 5.4 was terminated, we could make the definitive statement that

$$x_r = 143.1641 \pm \frac{143.7500 - 142.5781}{2} = 143.1641 \pm 0.5859$$

In essence, Eq. (5.5) provides an upper bound on the true error. For this bound to be exceeded, the true root would have to fall outside the bracketing interval, which by definition could never occur for bisection. Other root-locating techniques do not always behave as nicely. Although bisection is generally slower than other methods, the neatness of its error analysis is a positive feature that makes it attractive for certain engineering and scientific applications.

Another benefit of the bisection method is that the number of iterations required to attain an absolute error can be computed *a priori*—that is, before starting the computation. This can be seen by recognizing that before starting the technique, the absolute error is

$$E_a^0 = x_u^0 - x_l^0 = \Delta x^0$$

where the superscript designates the iteration. Hence, before starting the method we are at the "zero iteration". After the first iteration, the error becomes

$$E_a^1 = \frac{\Delta x^0}{2}$$

Because each succeeding iteration halves the error, a general formula relating the error and the number of iterations n is

$$E_a^n = \frac{\Delta x^0}{2^n}$$

If $E_{a,d}$ is the desired error, this equation can be solved for[2]

$$n = \frac{log(\Delta x^0/E_{a,d})}{log2} = log_2\left(\frac{\Delta x^0}{E_{a,d}}\right) \tag{5.6}$$

Let's test the formula. For Example 5.4, the initial interval was $\Delta x = 200 - 50 = 150$. After eight iterations, the absolute error was

$$E_a = \frac{|143.7500 - 142.5781|}{2} = 0.5859$$

We can substitute these values into Eq. (5.6) to give

$$n = log_2(150/0.5859) = 8$$

Thus, if we knew beforehand that an error of less than 0.5859 was acceptable, the formula tells us that eight iterations would yield the desired result.

Although we have emphasized the use of relative errors for obvious reasons, there will be cases where (usually through knowledge of the problem context) you will be able to specify an absolute error. For these cases, bisection along with Eq. (5.6) can provide a useful root-location algorithm.

3.4.1. MATLAB M-file: bisect

An M-file to implement bisection is displayed in Fig. 5.7. It is passed the function (`func`) along with lower (`xl`) and upper (`xu`) guesses. In addition, an optional stopping criterion (`es`) and maximum iterations (`maxit`) can be entered. The function first checks whether there are sufficient arguments and if the initial guesses bracket a sign change. If not, an error message is displayed and the function is terminated. It also assigns default values if `maxit` and `es` are not supplied. Then a `while...break` loop is employed to implement the bisection algorithm until the approximate error falls below es or the iterations exceed `maxit`.

We can employ this function to solve the problem posed at the beginning of the chapter. Recall that you need to determine the mass at which a bungee jumper's free-fall velocity exceeds 36 m/s after 4 s of free fall given a drag coefficient of 0.25 kg/m. Thus, you have to find the root of

$$f(m) = \sqrt{\frac{9.81m}{0.25}}\tanh\left(\sqrt{\frac{9.81(0.25)}{m}}4\right) - 36$$

In Example 5.1 we generated a plot of this function versus mass and estimated that the root fell between 140 and 150 kg. The `bisect` function from Fig. 5.7 can be used to determine the root as

```
» fm=@(m) sqrt(9.81*m/0.25)*tanh(sqrt(9.81*0.25/m)*4)-36;
» [mass fx ea iter]=bisect(fm,40,200)
mass =
      142.74
fx =
      4.6089e-007
ea =
      5.345e-005
iter =
      21
```

Thus, a result of m = 142.74 kg is obtained after 21 iterations with an approximate relative error of $\varepsilon_a = 0.00005345\%$, and a function value close to zero.

[2]MATLAB provides the `log2` function to evaluate the base-2 logarithm directly. If the pocket calculator or computer language you are using does not include the base-2 logarithm as an intrinsic function, this equation shows a handy way to compute it. In general, $log_b(x) = log(x)/log(b)$.

```
function [root,fx,ea,iter]=bisect(func,xl,xu,es,maxit,varargin)
% bisect: root location zeroes
%   [root,fx,ea,iter]=bisect(func,xl,xu,es,maxit,p1,p2,...):
%      uses bisection method to find the root of func
% input:
%   func = name of function
%   xl, xu = lower and upper guesses
%   es = desired relative error (default = 0.0001%)
%   maxit = maximum allowable iterations (default = 50)
%   p1,p2,... = additional parameters used by func
% output:
%   root = real root
%   fx = function value at root
%   ea = approximate relative error (%)
%   iter = number of iterations
if nargin<3,error('at least 3 input arguments required'),end
test = func(xl,varargin{:})*func(xu,varargin{:});
if test>0,error('no sign change'),end
if nargin<4|isempty(es), es=0.0001;end
if nargin<5|isempty(maxit), maxit=50;end
iter = 0; xr = xl; ea = 100;
while (1)
  xrold = xr;
  xr = (xl + xu)/2;
  iter = iter + 1;
  if xr ~= 0,ea = abs((xr - xrold)/xr) * 100;end
  test = func(xl,varargin{:})*func(xr,varargin{:});
  if test < 0
    xu = xr;
  elseif test > 0
    xl = xr;
  else
    ea = 0;
  end
  if ea <= es | iter >= maxit,break,end
end
root = xr; fx = func(xr, varargin{:});
```

Figure 3.7: An M-file to implement the bisection method.

3.5. FALSE POSITIVE

False position (also called the linear interpolation method) is another well-known bracketing method. It is very similar to bisection with the exception that it uses a different strategy to come up with its new root estimate. Rather than bisecting the interval, it locates the root by joining $f(x_l)$ and $f(x_u)$ with a straight line (Fig. 5.8). The intersection of this line with the x axis represents an improved estimate of the root. Thus, the shape of the function influences the new root estimate. Using similar triangles, the intersection of the straight line with the x axis can be estimated as (see Chapra and Canale, 2010, for details),

$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)} \tag{5.7}$$

This is the *false-position formula*. The value of x_r computed with Eq. (5.7) then replaces whichever of the two initial guesses, x_l or x_u, yields a function value with the same sign as $f(x_r)$. In this way the values of x_l and x_u always bracket the true root. The process is repeated until the root is estimated adequately. The algorithm is identical to the one for bisection (Fig. 5.7) with the exception that Eq. (5.7) is used.

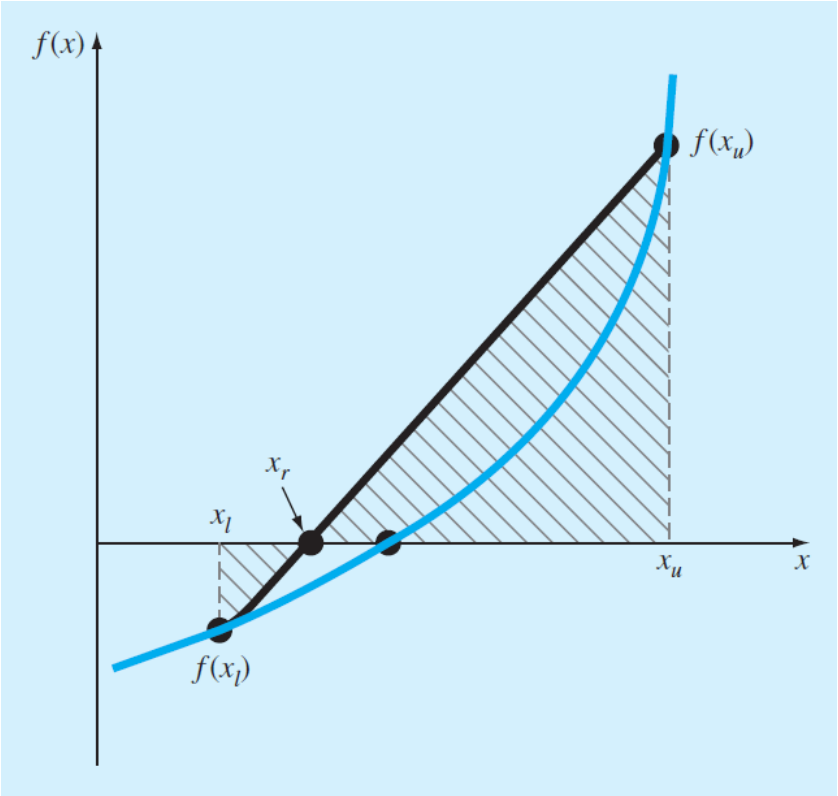


Figure 3.8: False position.

Example 3.5. The False-Position Method

Problem Statement. Use false position to solve the same problem approached graphically and with bisection in Examples 5.1 and 5.3.

Solution. As in Example 5.3, initiate the computation with guesses of $x_l = 50$ and $x_u = 200$.

First iteration:

$$x_l = 50 \qquad f(x_l)$$

$$x_u = 200 \qquad f(x_u) = 0.860291$$

$$x_r = 200 - \frac{0.860291(50 - 200)}{-4.579387 - 0.860291} = 176.2773$$

which has a true relative error of 23.5%.

Second iteration:

$$f(x_l)f(x_r) = -2.592732$$

Therefore, the root lies in the first subinterval, and x_r becomes the upper limit for the next iteration, $x_u = 176.2773$.

$$x_l = 50 \qquad f(x_l) = -4.579387$$

$$x_u = 176.2773 \qquad f(x_u) = 0.566174$$

$$x_r = 176.2773 - \frac{0.566174(50 - 176.2773)}{-4.579387 - 0.566174} = 162.3828$$

which has true and approximate relative errors of 13.76% and 8.56%, respectively. Additional iterations can be performed to refine the estimates of the root. ■

Although false position often performs better than bisection, there are other cases where it does not. As in the following example, there are certain cases where bisection yields superior results.

Example 3.6. A Case Where Bisection Is Preferable to False Position

Problem Statement. Use bisection and false position to locate the root of

$$f(x) = x^{10} - 1$$

between $x = 0$ and 1.3.

Solution. Using bisection, the results can be summarized as

Iteration	x_l	x_u	x_r	ε_a (%)	ε_t (%)
1	0	1.3	0.65	100.0	35
2	0.65	1.3	0.975	33.3	2.5
3	0.975	1.3	1.1375	14.3	13.8
4	0.975	1.1375	1.05625	7.7	5.6
5	0.975	1.05625	1.015625	4.0	1.6

Thus, after five iterations, the true error is reduced to less than 2%. For false position, a very different outcome is obtained:

Iteration	x_l	x_u	x_r	ε_a (%)	ε_t (%)
1	0	1.3	0.09430		90.6
2	0.09430	1.3	0.18176	48.1	81.8
3	0.18176	1.3	0.26287	30.9	73.7
4	0.26287	1.3	0.33811	22.3	66.2
5	0.33811	1.3	0.40788	17.1	59.2

After five iterations, the true error has only been reduced to about 59%. Insight into these results can be gained by examining a plot of the function. As in Fig. 5.9, the curve violates the premise on which false position was based—that is, if $f(x_l)$ is much closer to zero than $f(x_u)$, then the root should be much closer to x_l than to x_u (recall Fig. 5.8). Because of the shape of the present function, the opposite is true.

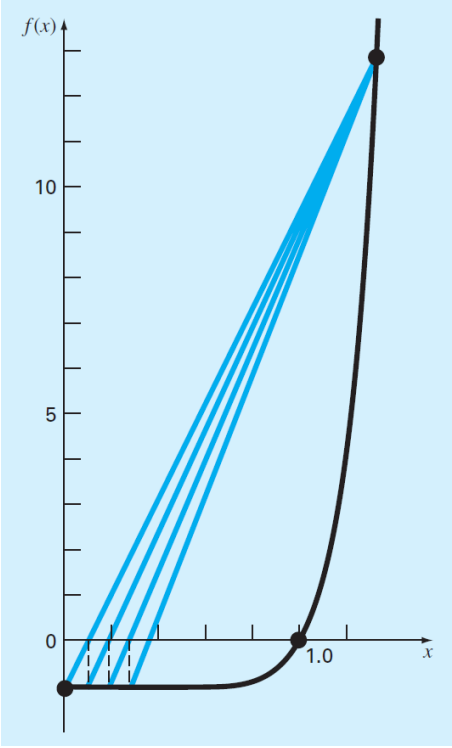


Figure 3.9: Plot of $f(x) = x^{10} - 1$, illustrating slow convergence of the false-position method.

■

The foregoing example illustrates that blanket generalizations regarding rootlocation methods are usually not possible. Although a method such as false position is often superior to bisection, there are invariably cases that violate this general conclusion. Therefore, in addition to using Eq. (5.5), the results should always be checked by substituting the root estimate into the original equation and determining whether the result is close to zero.

The example also illustrates a major weakness of the false-position method: its onesidedness. That is, as iterations are proceeding, one of the bracketing points will tend to stay fixed. This can lead to poor convergence, particularly for functions with significant curvature. Possible remedies for this shortcoming are available elsewhere (Chapra and Canale, 2010).

3.6. CASE STUDY: GREENHOUSE GASES AND RAINWATER

Background. It is well documented that the atmospheric levels of several so-called "greenhouse" gases have been increasing over the past 50 years. For example, Fig. 5.10 shows data for the partial pressure of carbon dioxide (CO_2) collected at Mauna Loa, Hawaii from 1958 through 2008. The trend in these data can be nicely fit with a quadratic polynomial, [3]

$$p_{CO_2} = 0.012226(t-1983)^2 + 1.418542(t-1983) + 342.38309$$

where $p_{CO_2} = CO_2$ partial pressure (ppm). These data indicate that levels have increased a little over 22% over the period from 315 to 386 ppm. One question that we can address is how this trend is affecting the pH of rainwater. Outside of urban and industrial areas, it is well documented that carbon dioxide is the primary determinant of the pH of the rain. pH is the measure of the activity of hydrogen ions and, therefore, its acidity or alkalinity. For dilute aqueous solutions, it can be computed as

$$pH = -log_{10}[H^+] \tag{5.8}$$

where $[H^+]$ is the molar concentration of hydrogen ions.
The following five equations govern the chemistry of rainwater:

$$K_1 = 10^6 \frac{[H^+][HCO_3^-]}{K_H p_{CO_2}} \tag{5.9}$$

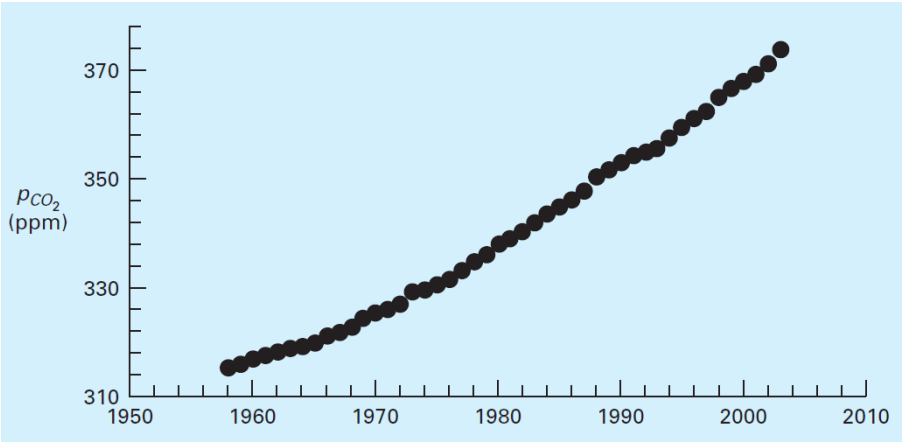


Figure 3.10: Average annual partial pressures of atmospheric carbon dioxide (ppm) measured at Mauna Loa, Hawaii.

$$K_2 = \frac{[H^+][CO_3^{-2}]}{HCO_3^-} \tag{5.10}$$

$$K_w = [H^+][OH^-] \tag{5.11}$$

$$c_T = \frac{K_H p_{CO_2}}{10^6} + [HCO_3^-] + [CO_3^{-2}] \tag{5.12}$$

$$0 = [HCO_3^-] + 2[CO_3^{-2}] + [OH^-] - [H^+] \tag{5.13}$$

[3] In Part Four, we will learn how to determine such polynomials.

where K_H = Henry's constant, and K_1, K_2, and K_w are equilibrium coefficients. The five unknowns are c_T = total inorganic carbon, $[HCO_3^-]$ = bicarbonate, $[CO_3^{-2}]$ = carbonate, $[H^+]$ = hydrogen ion, and $[OH^-]$ = hydroxyl ion. Notice how the partial pressure of CO_2 shows up in Eqs. (5.9) and (5.12).

Use these equations to compute the pH of rainwater given that $K_H = 10^{-1.46}$, $K_1 = 10^{-6.3}$, $K_2 = 10^{-10.3}$, and $K_w = 10^{-14}$. Compare the results in 1958 when the p_{CO_2} was 315 and in 2008 when it was 386 ppm. When selecting a numerical method for your computation, consider the following:

- You know with certainty that the pH of rain in pristine areas always falls between 2 and 12.
- You also know that pH can only be measured to two places of decimal precision.

Solution. There are a variety of ways to solve this system of five equations. One way is to eliminate unknowns by combining them to produce a single function that only depends on $[H^+]$. To do this, first solve Eqs. (5.9) and (5.10) for

$$[HCO_3^-] = \frac{K_1}{10^6[H^+]} K_H p_{CO_2} \tag{5.14}$$

$$[CO_3^{-2}] = \frac{K_2[HCO_3^-]}{[H^+]} \tag{5.15}$$

Substitute Eq. (5.14) into (5.15)

$$[CO_3^{-2}] = \frac{K_2 K_1}{10^6[H^+]^2} K_H p_{CO_2} \tag{5.16}$$

Equations (5.14) and (5.16) can be substituted along with Eq. (5.11) into Eq. (5.13) to give

$$0 = \frac{K_1}{10^6[H^+]} K_H p_{CO_2} + 2\frac{K_2 K_1}{10^6[H^+]^2} K_H p_{CO_2} + \frac{K_w}{[H^+]} - [H^+] \tag{5.17}$$

Although it might not be immediately apparent, this result is a third-order polynomial in $[H^+]$. Thus, its root can be used to compute the pH of the rainwater.

Now we must decide which numerical method to employ to obtain the solution. There are two reasons why bisection would be a good choice. First, the fact that the pH always falls within the range from 2 to 12, provides us with two good initial guesses. Second, because the pH can only be measured to two decimal places of precision, we will be satisfied with an absolute error of $E_{a,d} = \pm 0.005$. Remember that given an initial bracket and the desired error, we can compute the number of iteration *a priori*. Substituting the present values into Eq. (5.6) gives

```
» dx=12-2;
» Ead=0.005;
» n=log2(dx/Ead)
n =
      10.9658
```

Eleven iterations of bisection will produce the desired precision.
Before implementing bisection, we must first express Eq. (5.17) as a function. Because it is relatively complicated, we will store it as an M-file:

```
function f = fpH(pH,pCO2)
K1=10^-6.3;K2=10^-10.3;Kw=10^-14;
KH=10^-1.46;
H=10^-pH;
f=K1/(1e6*H)*KH*pCO2+2*K2*K1/(1e6*H)*KH*pCO2+Kw/H-H;
```

We can then use the M-file from Fig. 5.7 to obtain the solution. Notice how we have set the value of the desired relative error ($\varepsilon_a = 1 \times 10^-8$) at a very low level so that the iteration limit (`maxit`) is reached first so that exactly 11 iterations are implemented

```
» [pH1958 fx ea iter]=bisect(@fpH,2,12,1e-8,11,315)
pH1958 =
      5.6279
fx =
      -2.7163e-008
ea =
      0.08676
iter =
      11
```

Thus, the pH is computed as 5.6279 with a relative error of 0.0868%. We can be confident that the rounded result of 5.63 is correct to two decimal places. This can be verified by performing another run with more iterations. For example, setting `maxit` to 50 yields

```
» [pH1958 fx ea iter] = bisect(@fpH,2,12,1e-8,50,315)
pH1958 =
      5.6304
fx =
      1.615e-015
ea =
      5.169e-009
iter =
35
```

For 2008, the result is

```
» [pH2008 ea iter]=bisect(@fpH,2,12,1e-8,50,386)
pH2008 =
      5.5864
fx =
      3.2926e-015
ea =
      5.2098e-009
iter =
      35
```

Interestingly, the results indicate that the 22.5% rise in atmospheric CO_2 levels has produced only a 0.78% drop in pH. Although this is certainly true, remember that the pH represents a logarithmic scale as defined by Eq. (5.8). Consequently, a unit drop in pH represents an order-of-magnitude (i.e., a 10-fold) increase in the hydrogen ion. The concentration can be computed as $[H^+] = 10^{-pH}$ and its percent change can be calculated as.

```
» ((10^-pH2008-10^-pH1958)/10^-pH1958)*100
ans =
      10.6791
```

Therefore, the hydrogen ion concentration has increased about 10.7%.

There is quite a lot of controversy related to the meaning of the greenhouse gas trends. Most of this debate focuses on whether the increases are contributing to global warming. However, regardless of the ultimate implications, it is sobering to realize that something as large as our atmosphere has changed so much over a relatively short time period. This case study illustrates how numerical methods and MATLAB can be employed to analyze and interpret such trends. Over the coming years, engineers and scientists can hopefully use such tools to gain increased understanding of such phenomena and help rationalize the debate over their ramifications.

PROBLEMS

5.1 Use bisection to determine the drag coefficient needed so that an 80-kg bungee jumper has a velocity of 36 m/s after 4 s of free fall. Note: The acceleration of gravity is $9.81 m/s^2$. Start with initial guesses of $x_l = 0.1$ and $x_u = 0.2$ and iterate until the approximate relative error falls below 2%.

5.2 Develop your own M-file for bisection in a similar fashion to Fig. 5.7. However, rather than using the maximum iterations and Eq. (5.5), employ Eq. (5.6) as your stopping criterion. Make sure to round the result of Eq. (5.6) up to the next highest integer. Test your function by solving Prob. 5.1 using $E_{a,d} = 0.0001$.

5.3 Repeat Prob. 5.1, but use the false-position method to obtain your solution.

5.4 Develop an M-file for the false-position method. Test it by solving Prob. 5.1.

5.5 (a) Determine the roots of $f(x) = -12 - 21x + 18x^2 - 2.75x^3$ graphically. In addition, determine the first root of the function with **(b)** bisection and **(c)** false position. For **(b)** and **(c)** use initial guesses of $x_l = -1$ and $x_u = 0$ and a stopping criterion of 1%.

5.6 Locate the first nontrivial root of $sin(x) = x^2$ where x is in radians. Use a graphical technique and bisection with the initial interval from 0.5 to 1. Perform the computation until Îţa is less than $\varepsilon_s = 2\%$.

5.7 Determine the positive real root of $ln(x^2) = 0.7$ **(a)** graphically, **(b)** using three iterations of the bisection method, with initial guesses of $x_l = 0.5$ and $x_u = 2$, and

(c) using three iterations of the false-position method, with the same initial guesses as in **(b)**.

5.8 The saturation concentration of dissolved oxygen in freshwater can be calculated with the equation

$$\ln o_{sf} = -139.34411 + \frac{1.575701 \times 10^5}{T_a} - \frac{6.642308 \times 10^7}{T_a^2} + \frac{1.243800 \times 10^{10}}{T_a^3} - \frac{8.621949 \times 10^{11}}{T_a^4}$$

where o_{sf} = the saturation concentration of dissolved oxygen in freshwater at 1 atm (mgL^{-1}); and T_a = absolute temperature (K). Remember that $T_a = T + 273.15$, where T = temperature (°C). According to this equation, saturation decreases with increasing temperature. For typical natural waters in temperate climates, the equation can be used to determine that oxygen concentration ranges from 14.621 mg/L at 0 °C to 6.949 mg/L at 35 °C. Given a value of oxygen concentration, this formula and the bisection method can be used to solve for temperature in °C.
(a) If the initial guesses are set as 0 and 35 °C, how many bisection iterations would be required to determine temperature to an absolute error of 0.05 °C?
(b) Based on **(a)**, develop and test a bisection M-file function to determine T as a function of a given oxygen concentration. Test your function for o_{sf} = 8, 10 and 14 mg/L. Check your results.

5.9 A beam is loaded as shown in Fig. P5.9. Use the bisection method to solve for the position inside the beam where there is no moment.

5.10 Water is flowing in a trapezoidal channel at a rate of $Q = 20m^3/s$. The critical depth y for such a channel must satisfy the equation

$$0 = 1 - \frac{Q^2}{gA_c^3}B$$

where $g = 9.81m/s^2$, A_c = the cross-sectional area (m_2), and B = the width of the channel at the surface (m). For this case, the width and the cross-sectional area can be related to depth y by

$$B = 3 + y$$

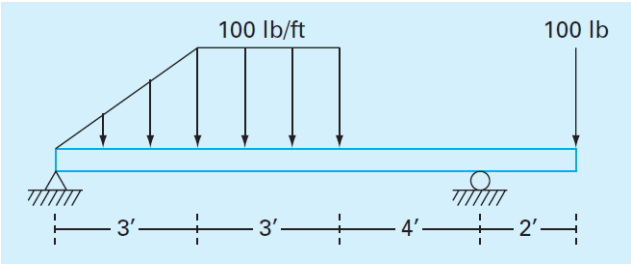


Figure P5.9

and

$$A_c = 3y + \frac{y^2}{2}$$

Solve for the critical depth using **(a)** the graphical method, **(b)** bisection, and **(c)** false position. For **(b)** and **(c)** use initial guesses of $x_l = 0.5$ and $x_u = 2.5$, and iterate until the approximate error falls below 1% or the number of iterations exceeds 10. Discuss your results.

5.11 The Michaelis-Menten model describes the kinetics of enzyme mediated reactions:

$$\frac{dS}{dt} = -v_m \frac{S}{k_s + S}$$

where S = substrate concentration (moles/L), v_m = maximum uptake rate (moles/L/d), and k_s = the half-saturation constant, which is the substrate level at which uptake is half of the maximum [moles/L]. If the initial substrate level at t = 0 is S_0, this differential equation can be solved for

$$S = S_0 - v_m t + k_s \ln(S_0/S)$$

Develop an M-file to generate a plot of S versus t for the case where S_0 = 8 moles/L, v_m = 0.7 moles/L/d, and k_s = 2.5 moles/L.

5.12 A reversible chemical reaction

$$2A + B \leftrightarrows C$$

can be characterized by the equilibrium relationship

$$K = \frac{c_c}{c_a^2 c_b}$$

where the nomenclature c_i represents the concentration of constituent i. Suppose that we define a variable x as representing the number of moles of C that are produced. Conservation of mass can be used to reformulate the equilibrium relationship as

$$K = \frac{(c_{c,0} + x)}{(c_{a,0-2x})^2 (c_{b,0} - x)}$$

where the subscript 0 designates the initial concentration of each constituent. If $K = 0.016$, $c_{a,0} = 42$, $c_{b,0} = 28$, and $c_{c,0} = 4$, determine the value of x.
(a) Obtain the solution graphically.
(b) On the basis of **(a)**, solve for the root with initial guesses of $x_l = 0$ and $x_u = 20$ to $\varepsilon_s = 0.5\%$. Choose either bisection or false position to obtain your solution. Justify your choice.

5.13 Figure P5.13a shows a uniform beam subject to a linearly increasing distributed load. The equation for the resulting elastic curve is (see Fig. P5.13b)

$$y = \frac{w_0}{120EIL}(-x^5 + 2L^2x^3 - L^4x) \qquad \text{(P5.13)}$$

Use bisection to determine the point of maximum deflection (i.e., the value of x where $dy/dx = 0$). Then substitute this value into Eq. (P5.13) to determine the value of the maximum deflection. Use the following parameter values in your computation: $L = 600cm$, $E = 50{,}000kN/cm^2$, $I = 30{,}000cm^4$, and $w_0 = 2.5kN/cm$.

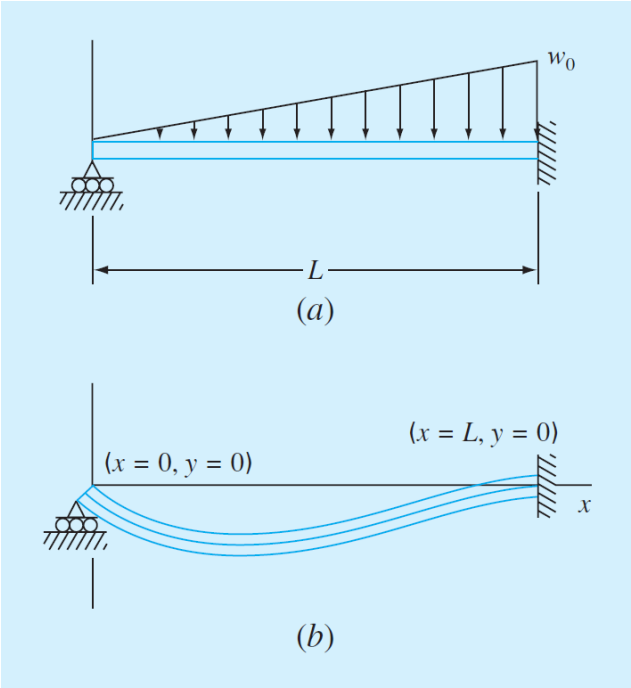


Figure P5.13

5.14 You buy a $35,000 vehicle for nothing down at $8,500 per year for 7 years. Use the `bisect` function from Fig. 5.7 to determine the interest rate that you are paying. Employ initial guesses for the interest rate of 0.01 and 0.3 and a stopping criterion of 0.00005. The formula relating present worth P, annual payments A, number of years n, and interest rate i is

$$A = P\frac{i(1+i)^n}{(i+i)^n - 1}$$

5.15 Many fields of engineering require accurate population estimates. For example, transportation engineers might find it necessary to determine separately the population growth trends of a city and adjacent suburb. The population of the urban area is declining with time according to

$$P_u(t) = P_{u,max}e^{k_u t} + P_{u,min}$$

while the suburban population is growing, as in

$$P_s(t) = \frac{P_{s,max}}{1 + [P_{s,max}/P_0 - 1]e^{-k_s t}}$$

where $P_{u,max}$, k_u, $P_{s,max}$, P_0, and k_s = empirically derived parameters. Determine the time and corresponding values of $P_u(t)$ and $P_s(t)$ when the suburbs are 20% larger than the city. The parameter values are $O_{u,max} = 80,000$, $k_u = 0.05/yr$, $P_{u,min} = 110,000$ people, $P_{s,max} = 320,000$ people, $P_0 = 10,000$ people, and $k_s = 0.09/yr$. To obtain your solutions, use **(a)** graphical, and **(b)** false-position methods.

5.16 The resistivity ρ of doped silicon is based on the charge q on an electron, the electron density n, and the electron mobility μ. The electron density is given in terms of the doping density N and the intrinsic carrier density n_i. The electron mobility is described by the temperature T, the reference temperature T_0, and the reference mobility $_0$. The equations required to compute the resistivity are

$$\rho = \frac{1}{qn\mu}$$

where

$$n = \frac{1}{2}\left(N + \sqrt{N^2 + 4n_i^2}\right) \quad \text{and} \quad \mu = \mu_0\left(\frac{T}{T_0}\right)^{-2.42}$$

Determine N, given $T_0 = 300$ K, $T = 1000$ K, $\mu_0 = 1360 cm^2$ (V s)$^-1$, $q = 1.7 \times 10^{-19}$ C, $n_i = 6.21 \times 10^9 cm^{-3}$, and a desired $\rho = 6.5 \times 10^6$ V s cm/C. Employ initial guesses of $N = 0$ and 2.5×10^{10}. Use **(a)** bisection and **(b)** the false position method.

5.17 Atotal charge Q is uniformly distributed around a ring-shaped conductor with radius a. A charge q is located at a distance x from the center of the ring (Fig. P5.17). The force exerted on the charge by the ring is given by

$$F = \frac{1}{4\pi e_0}\frac{qQx}{(x^2 + a^2)^{3/2}}$$

where $e_0 = 8.9 \times 10^{-12} C^2/(\text{N m}^2)$. Find the distance x where the force is 1.25 N if q and Q are 2×10^{-5} C for a ring with a radius of 0.85m.

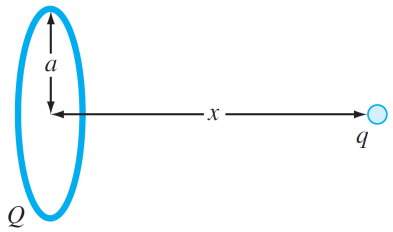


Figure P5.17

5.18 For fluid flow in pipes, friction is described by a dimensionless number, the *Fanning friction factor f*. The Fanning friction factor is dependent on a number of parameters related to the size of the pipe and the fluid, which can all be represented by another dimensionless quantity, the *Reynolds number* Re. A formula that predicts f given Re is the *von Karman equation*:

$$\frac{1}{\sqrt{f}} = 4log_{10}(Re\sqrt{f}) - 0.4$$

Typical values for the Reynolds number for turbulent flow are 10,000 to 500,000 and for the Fanning friction factor are 0.001 to 0.01. Develop a function that uses bisection to solve for f given a user-supplied value of Re between 2,500 and 1,000,000. Design the function so that it ensures that the absolute error in the result is $E_{a,d} < 0.000005$.

5.19 Mechanical engineers, as well as most other engineers, use thermodynamics extensively in their work. The following polynomial can be used to relate the zero-pressure specific heat of dry air c_p kJ/(kg K) to temperature (K):

$$c_p = 0.99403 + 1.671 \times 10^{-4}T + 9.7215 \times 10^{-8}T^2 - 9.5838 \times 10^{-11}T^3 + 1.9520 \times 10^{-14}T^4$$

Develop a plot of c_p versus a range of $T = 0$ to 1200 K, and then use bisection to determine the temperature that

corresponds to a specific heat of 1.1 kJ/(kg K).

5.20 The upward velocity of a rocket can be computed by the following formula:

$$v = u\ln\frac{m_0}{m_0 - qt} - gt$$

where v = upward velocity, u = the velocity at which fuel is expelled relative to the rocket, m_0 = the initial mass of the rocket at time $t = 0$, q=the fuel consumption rate, and g=the downward acceleration of gravity (assumed constant = $9.81 m/s^2$). If $u = 1800 m/s$, $m_0 = 160{,}000 kg$, and $q = 2600 kg/s$, compute the time at which $v = 750 m/s$. (Hint: t is somewhere between 10 and 50 s.) Determine your result so that it is within 1% of the true value. Check your answer.

5.21 Although we did not mention it in Sec. 5.6, Eq. (5.13) is an expression of *electroneutrality*—that is, that positive and negative charges must balance. This can be seen more clearly by expressing it as

$$[H^+] = [HCP_3^-] + 2[CO_3^{2-}] + [OH^-]$$

In other words, the positive charges must equal the negative charges. Thus, when you compute the pH of a natural water body such as a lake, you must also account for other ions that may be present. For the case where these ions originate from nonreactive salts, the net negative minus positive charges due to these ions are lumped together in a quantity called *alkalinity*, and the equation is reformulated as

$$Alk + [H^+] = [HCO_3^-] + 2[CO_3^{2-}] + [OH^-] \qquad \text{(P5.21)}$$

where Alk = alkalinity (eq/L). For example, the alkalinity of Lake Superior is approximately 0.4×10^{-3} eq/L. Perform the same calculations as in Sec. 5.6 to compute the pH of Lake Superior in 2008. Assume that just like the raindrops, the lake is in equilibrium with atmospheric CO_2 but account for the alkalinity as in Eq. (P5.21).

5.22 According to *Archimedes' principle*, the *buoyancy* force is equal to the weight of fluid displaced by the submerged portion of the object. For the sphere depicted in Fig. P5.22, use bisection to determine the height, h, of the portion that is above water. Employ the following values for your computation: $r = 1$ m, ρ_s = density of sphere = $200 kg/m^3$, and ρ_w= density of water = $1{,}000 kg/m^3$. Note that the volume of the above-water portion of the sphere can be computed with

$$V = \frac{\pi h^2}{3}(3r - h)$$

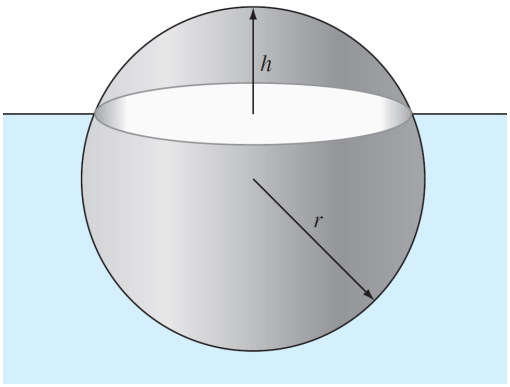


Figure P5.22

5.23 Perform the same computation as in Prob. 5.22, but for the frustrum of a cone as depicted in Fig. P5.23. Employ the following values for your computation: $r_1 = 0.5$ m, $r_2 = 1$ m, $h = 1$ m, ρ_f = frustrum density = $200 kg/m^3$, and ρ_w = water density = $1{,}000 kg/m^3$. Note that the volume of a frustrum is given by

$$V = \frac{\pi h}{3}(r_1^2 + r_2^2 + r_1 r_2)$$

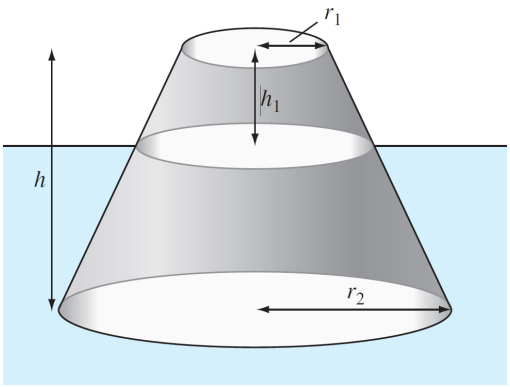


Figure P5.23

Chapter 4

Roots: Open Methods

CHAPTER OBJECTIVES

The primary objective of this chapter is to acquaint you with open methods for finding the root of a single nonlinear equation. Specific objectives and topics covered are

- Recognizing the difference between bracketing and open methods for root location.
- Understanding the fixed-point iteration method and how you can evaluate its convergence characteristics.
- Knowing how to solve a roots problem with the Newton-Raphson method and appreciating the concept of quadratic convergence.
- Knowing how to implement both the secant and the modified secant methods.
- Understanding how Brent's method combines reliable bracketing methods with fast open methods to locate roots in a robust and efficient manner.
- Knowing how to use MATLAB's `fzero` function to estimate roots.
- Learning how to manipulate and determine the roots of polynomials with MATLAB.

For the bracketing methods in Chap. 5, the root is located within an interval prescribed by a lower and an upper bound. Repeated application of these methods always results in closer estimates of the true value of the root. Such methods are said to be *convergent* because they move closer to the truth as the computation progresses (Fig. 6.1a).

In contrast, the *open methods* described in this chapter require only a single starting value or two starting values that do not necessarily bracket the root. As such, they sometimes *diverge* or move away from the true root as the computation progresses (Fig. 6.1b). However, when the open methods converge (Fig. 6.1c) they usually do so much more quickly than the bracketing methods. We will begin our discussion of open techniques with a simple approach that is useful for illustrating their general form and also for demonstrating the concept of convergence.

4.1. SIMPLE FIXED-POINT ITERATION

As just mentioned, open methods employ a formula to predict the root. Such a formula can be developed for simple *fixed-point iteration* (or, as it is also called, *one-point iteration* or *successive substitution*) by rearranging the function $f(x) = 0$ so that x is on the left-hand side of the equation:

$$x = g(x) \tag{6.1}$$

This transformation can be accomplished either by algebraic manipulation or by simply adding x to both sides of the original equation.

The utility of Eq. (6.1) is that it provides a formula to predict a new value of x as a function of an old value of x. Thus, given an initial guess at the root x_i , Eq. (6.1) can be used to compute a new estimate x_{i+1} as expressed by the iterative formula

$$x_{i+1} = g(x_i) \tag{6.2}$$

As with many other iterative formulas in this book, the approximate error for this equation can be determined using the error estimator:

$$\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| 100\% \tag{6.3}$$

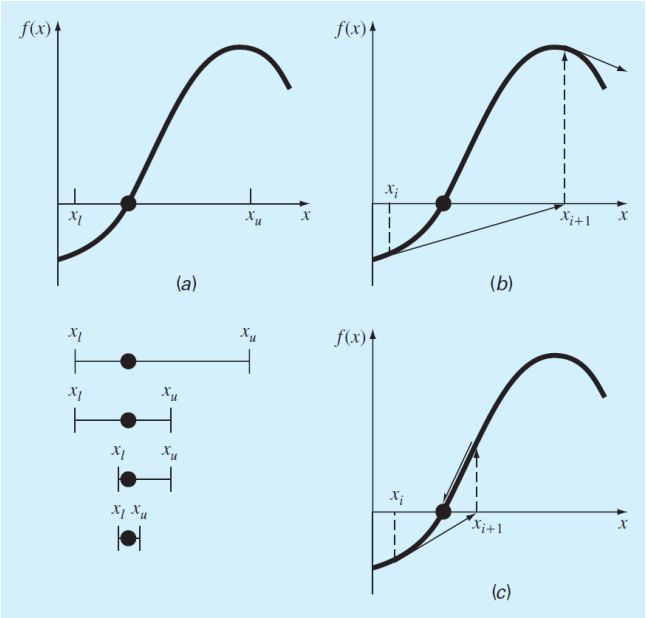


Figure 4.1: Graphical depiction of the fundamental difference between the (a) bracketing and (b) and (c) open methods for root location. In (a), which is bisection, the root is constrained within the interval prescribed by x_l and x_u . In contrast, for the open method depicted in (b) and (c), which is Newton-Raphson, a formula is used to project from x_i to $x_i + 1$ in an iterative fashion. Thus the method can either (b) diverge or (c) converge rapidly, depending on the shape of the function and the value of the initial guess.

Example 4.1. Simple Fixed-Point Iteration

Problem Statement. Use simple fixed-point iteration to locate the root of $f(x) = e^{-x} - x$

Solution. The function can be separated directly and expressed in the form of Eq. (6.2) as

$$x_{i+1} = e^{-x_i}$$

Starting with an initial guess of $x_0 = 0$, this iterative equation can be applied to compute:

i	x_i	$\lvert\varepsilon_a\rvert$, %	$\lvert\varepsilon_t\rvert$, %	$\lvert\varepsilon_t\rvert_i / \lvert\varepsilon_t\rvert_{i-1}$
0	0.0000		100.000	
1	1.0000	100.000	76.322	0.763
2	0.3679	171.828	35.135	0.460
3	0.6922	46.854	22.050	0.628
4	0.5005	38.309	11.755	0.533
5	0.6062	17.447	6.894	0.586
6	0.5454	11.157	3.835	0.556
7	0.5796	5.903	2.199	0.573
8	0.5601	3.481	1.239	0.564
9	0.5711	1.931	0.705	0.569
10	0.5649	1.109	0.399	0.566

Thus, each iteration brings the estimate closer to the true value of the root: 0.56714329. ■

Notice that the true percent relative error for each iteration of Example 6.1 is roughly proportional (for this case, by a factor of about 0.5 to 0.6) to the error from the previous iteration. This property, called *linear convergence*, is characteristic of fixed-point iteration.

Aside from the "rate" of convergence, we must comment at this point about the "possibility" of convergence. The concepts of convergence and divergence can be depicted graphically. Recall that in Section 5.2, we graphed a function to visualize its structure and behavior. Such an approach is employed in Fig. 6.2a for the function $f(x) = e^{-x} - x$. An alternative graphical approach is to separate the equation into two component parts, as in

$$f_1(x) = f_2(x)$$

Then the two equations

$$y_1 = f_1(x) \tag{6.4}$$

and

$$y_2 = f_2(x) \tag{6.5}$$

can be plotted separately (Fig. 6.2b). The x values corresponding to the intersections of these functions represent the roots of $f(x) = 0$.

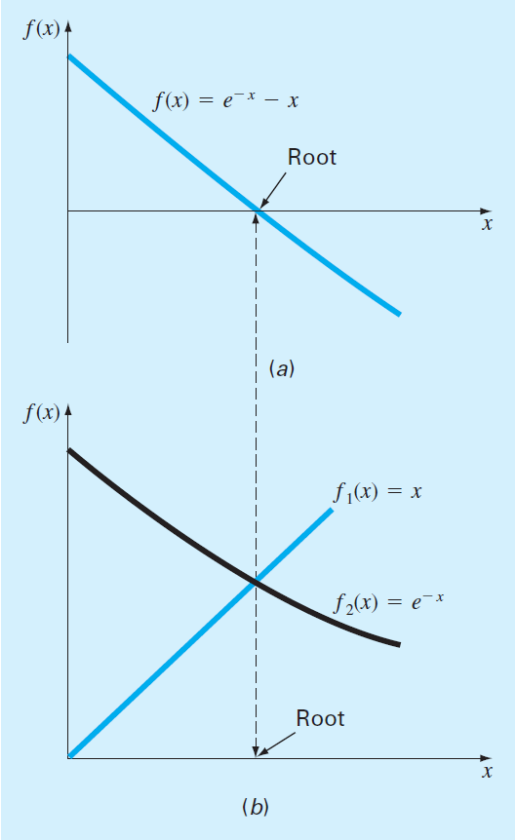


Figure 4.2: Two alternative graphical methods for determining the root of $f(x) = e^{-x} - x$. (a) Root at the point where it crosses the x axis; (b) root at the intersection of the component functions.

The two-curve method can now be used to illustrate the convergence and divergence of fixed-point iteration. First, Eq. (6.1) can be reexpressed as a pair of equations $y_1 = x$ and $y_2 = g(x)$. These two equations can then be plotted separately. As was the case with Eqs. (6.4) and (6.5), the roots of $f(x) = 0$ correspond to the abscissa value at the intersection of the two curves. The function $y_1 = x$ and four different shapes for $y_2 = g(x)$ are plotted in Fig. 6.3.

For the first case (Fig. 6.3a), the initial guess of x_0 is used to determine the corresponding point on the y_2 curve $[x_0, g(x_0)]$. The point $[x_1, x_1]$ is located by moving left horizontally to the y_1 curve. These movements are equivalent to the first iteration of the fixed-point method:

$$x_1 = g(x_0)$$

Thus, in both the equation and in the plot, a starting value of x_0 is used to obtain an estimate of x_1. The next iteration consists of moving to $[x_1, g(x_1)]$ and then to $[x_2, x_2]$. This iteration is equivalent to the equation

$$x_2 = g(x_1)$$

The solution in Fig. 6.3a is *convergent* because the estimates of x move closer to the root with each iteration. The same is true for Fig. 6.3b. However, this is not the case for Fig. 6.3c and d, where the iterations diverge from the root.

A theoretical derivation can be used to gain insight into the process. As described in Chapra and Canale (2010), it can be shown that the error for any iteration is linearly proportional to the error from the previous iteration multiplied by the absolute value of the slope of g:

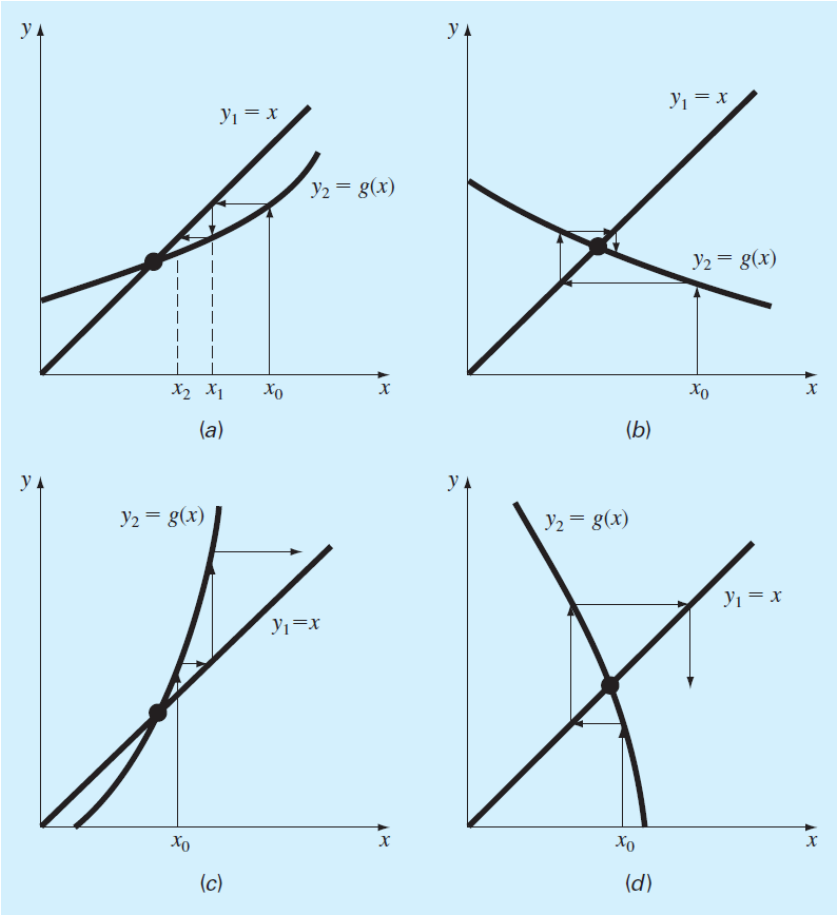


Figure 4.3: Graphical depiction of (a) and (b) convergence and (c) and (d) divergence of simple fixed-point iteration. Graphs (a) and (c) are called monotone patterns whereas (b) and (c) are called oscillating or spiral patterns. Note that convergence occurs when $|g'(x)| < 1$

$$E_{i+1} = g'(\xi)E_i$$

Consequently, if $|g'| < 1$, the errors decrease with each iteration. For $|g'| > 1$ the errors grow. Notice also that if the derivative is positive, the errors will be positive, and hence the errors will have the same sign (Fig. 6.3a and c). If the derivative is negative, the errors will change sign on each iteration (Fig. 6.3b and d).

4.2. NEWTON-RAPHSON

Perhaps the most widely used of all root-locating formulas is the *Newton-Raphson method* (Fig. 6.4). If the initial guess at the root is x_i, a tangent can be extended from the point $[x_i, f(x_i)]$. The point where this tangent crosses the x axis usually represents an improved estimate of the root.

The Newton-Raphson method can be derived on the basis of this geometrical interpretation. As in Fig. 6.4, the first derivative at x is equivalent to the slope:

$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}}$$

which can be rearranged to yield

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{6.6}$$

which is called the *Newton-Raphson formula.*

Example 4.2. Newton-Raphson Method

Problem Statement Use the Newton-Raphson method to estimate the root of $f(x) = e^{-x} - x$ employing an initial guess of $x_0 = 0$.

Solution. The first derivative of the function can be evaluated as

$$f'(x) = -e^{-x} - 1$$

which can be substituted along with the original function into Eq. (6.6) to give

$$x_{i+1} = x_i - \frac{e^{-x_i} - x_i}{-e^{-x_i} - 1}$$

Starting with an initial guess of $x_0 = 0$, this iterative equation can be applied to compute

i	x_i	$\lvert\varepsilon_t\rvert$, %
0	0	100
1	0.500000000	11.8
2	0.566311003	0.147
3	0.567143165	0.0000220
4	0.567143290	$<10^{-8}$

Thus, the approach rapidly converges on the true root. Notice that the true percent relative error at each iteration decreases much faster than it does in simple fixed-point iteration (compare with Example 6.1). ■

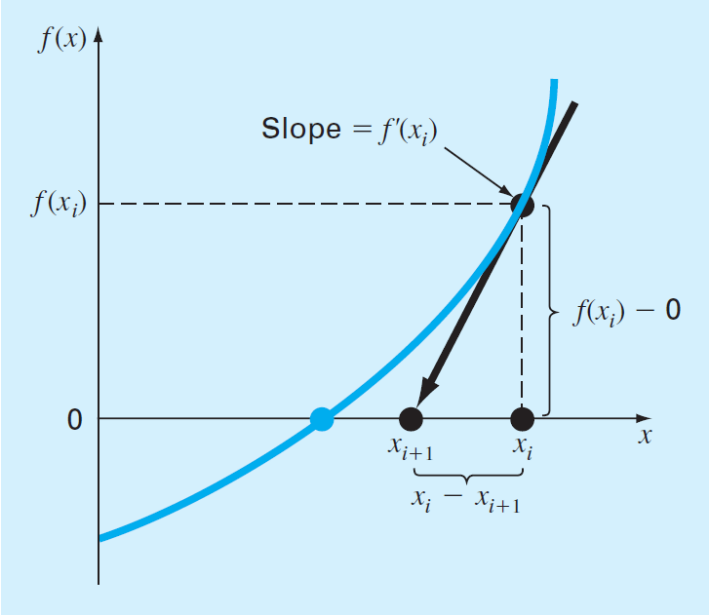


Figure 4.4: Graphical depiction of the Newton-Raphson method. A tangent to the function of x_i [that is,$f'(x)$] is extrapolated down to the x axis to provide an estimate of the root at x_{i+1}.

As with other root-location methods, Eq. (6.3) can be used as a termination criterion. In addition, a theoretical analysis (Chapra and Canale, 2010) provides insight regarding the rate of convergence as expressed by

$$E_{t,i+1} = \frac{-f''(x_r)}{2f'(x_r)} E_{t,i}^2 \tag{6.7}$$

Thus, the error should be roughly proportional to the square of the previous error. In other words, the number of significant figures of accuracy approximately doubles with each iteration. This behavior is called *quadratic convergence* and is one of the major reasons for the popularity of the method.

Although the Newton-Raphson method is often very efficient, there are situations where it performs poorly. A special case—multiple roots—is discussed elsewhere (Chapra and Canale, 2010). However, even when dealing with simple roots, difficulties can also arise, as in the following example.

Example 4.3. A Slowly Converging Function with Newton-Raphson

Problem Statement. Determine the positive root of $f(x) = x^{10} - 1$ using the Newton- Raphson method and an initial guess of $x = 0.5$.

Solution. The Newton-Raphson formula for this case is

$$x_{i+1} = x_i - \frac{x_i^{10} - 1}{10x_i^9}$$

which can be used to compute

i	x_i	$\lvert\varepsilon_a\rvert$, %
0	0.5	
1	51.65	99.032
2	46.485	11.111
3	41.8365	11.111
4	37.65285	11.111
⋮		
40	1.002316	2.130
41	1.000024	0.229
42	1	0.002

Thus, after the first poor prediction, the technique is converging on the true root of 1, but at a very slow rate.

Why does this happen? As shown in Fig. 6.5, a simple plot of the first few iterations is helpful in providing insight. Notice how the first guess is in a region where the slope is near zero. Thus, the first iteration flings the solution far away from the initial guess to a new value ($x = 51.65$) where $f(x)$ has an extremely high value. The solution then plods along for over 40 iterations until converging on the root with adequate accuracy.

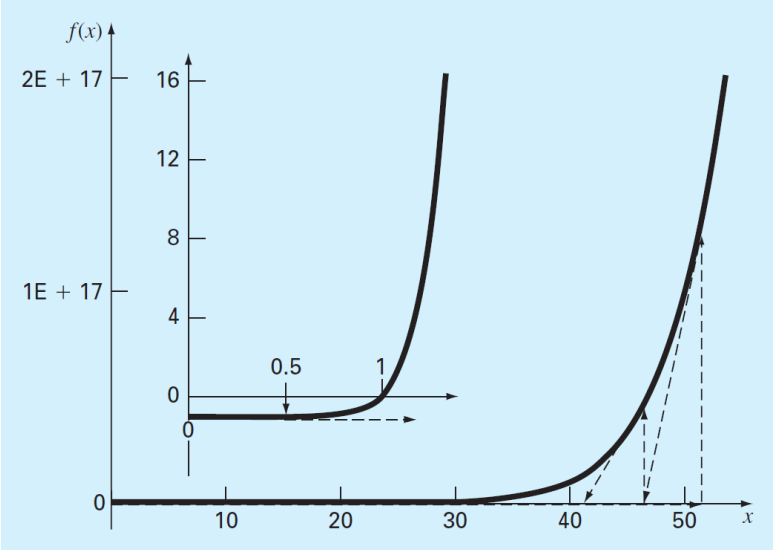


Figure 4.5: Graphical depiction of the Newton-Raphson method for a case with slow convergence. The inset shows how a near-zero slope initially shoots the solution far from the root. Thereafter, the solution very slowly converges on the root.

■

Aside from slow convergence due to the nature of the function, other difficulties can arise, as illustrated in Fig. 6.6. For example, Fig. 6.6a depicts the case where an inflection point (i.e., $f'(x) = 0$) occurs in the vicinity of a root. Notice that iterations beginning at x0 progressively diverge from the root. Fig. 6.6b illustrates the tendency of the Newton-Raphson technique to oscillate around a local maximum or minimum. Such oscillations may persist, or, as in Fig. 6.6b, a near-zero slope is reached whereupon the solution is sent far from the area of interest. Figure 6.6c shows how an initial guess that is close to one root can jump to a location several roots away. This tendency to move away from the area of interest is due to the fact that near-zero slopes are encountered. Obviously, a zero slope $[f'(x) = 0]$ is a real disaster because it causes division by zero in the Newton-Raphson formula [Eq. (6.6)]. As in Fig. 6.6d, it means that the solution shoots off horizontally and never hits the x axis.

Thus, there is no general convergence criterion for Newton-Raphson. Its convergence depends on the nature of the function and on the accuracy of the initial guess. The only remedy is to have an initial guess that is "sufficiently" close to the root. And for some functions, no guess will work! Good guesses are usually predicated on knowledge of the physical problem setting or on devices such as graphs that provide insight into the behavior of the solution. It also suggests that good computer software should be designed to recognize slow convergence or divergence.

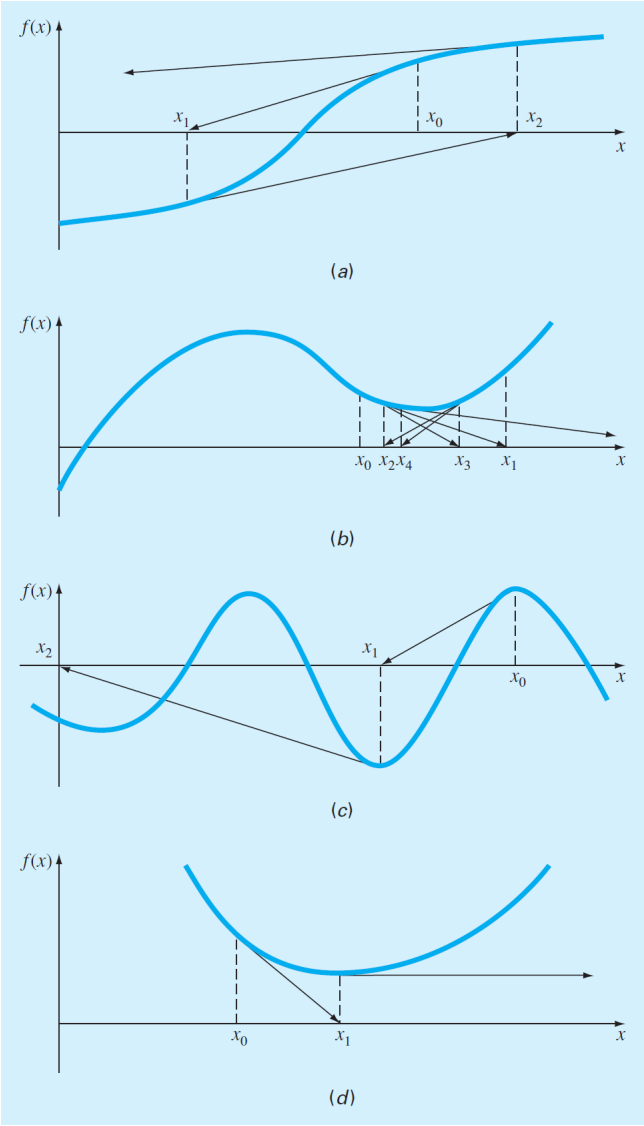


Figure 4.6: Four cases where the Newton-Raphson method exhibits poor convergence.

4.2.1. MATLAB M-file: newtraph

An algorithm for the Newton-Raphson method can be easily developed (Fig. 6.7). Note that the program must have access to the function (`func`) and its first derivative (`dfunc`). These can be simply accomplished by the inclusion of user-defined functions to compute these quantities. Alternatively, as in the algorithm in Fig. 6.7, they can be passed to the function as arguments.

After the M-file is entered and saved, it can be invoked to solve for root. For example, for the simple function $x^2 - 9$, the root can be determined as in

```
» newtraph(@(x) x^2-9,@(x) 2*x,5)
ans =
      3
```

Example 4.4. Newton-Raphson Bungee Jumper Problem

Problem Statement. Use the M-file function from Fig. 6.7 to determine the mass of the bungee jumper with a drag coefficient of 0.25 kg/m to have a velocity of 36 m/s after 4 s of free fall. The acceleration of gravity is $9.81 m/s^2$.

Solution. The function to be evaluated is

$$f(m) = \sqrt{\frac{gm}{c_d}} tanh\left(\sqrt{\frac{gc_d}{m}} t\right) - v(t) \tag{E6.4.1}$$

To apply the Newton-Raphson method, the derivative of this function must be evaluated with respect to the unknown, m:

$$\frac{df(m)}{dm} = \frac{1}{2}\sqrt{\frac{g}{mc_d}}tanh\left(\sqrt{\frac{gc_d}{m}}t\right) - \frac{g}{2m}t\ \text{sech}^2\left(\sqrt{\frac{gc_d}{m}}t\right) \tag{E6.4.2}$$

```
function [root,ea,iter]=newtraph(func,dfunc,xr,es,maxit,varargin)
% newtraph: Newton-Raphson root location zeroes
%  [root,ea,iter]=newtraph(func,dfunc,xr,es,maxit,p1,p2,...):
%        uses Newton-Raphson method to find the root of func
% input:
%   func = name of function
%   dfunc = name of derivative of function
%   xr = initial guess
%   es = desired relative error (default = 0.0001%)
%   maxit = maximum allowable iterations (default = 50)
%   p1,p2,... = additional parameters used by function
% output:
%   root = real root
%   ea = approximate relative error (%)
%   iter = number of iterations

if nargin<3,error('at least 3 input arguments required'),end
if nargin<4|isempty(es),es=0.0001;end
if nargin<5|isempty(maxit),maxit=50;end
iter = 0;
while (1)
  xrold = xr;
  xr = xr - func(xr)/dfunc(xr);
  iter = iter + 1;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  if ea <= es | iter >= maxit, break, end
end
root = xr;
```

Figure 4.7: An M-file to implement the Newton-Raphson method.

We should mention that although this derivative is not difficult to evaluate in principle, it involves a bit of concentration and effort to arrive at the final result.

The two formulas can now be used in conjunction with the function `newtraph` to evaluate the root:

```
» y = @m sqrt(9.81*m/0.25)*tanh(sqrt(9.81*0.25/m)*4)-36;
» dy = @m 1/2*sqrt(9.81/(m*0.25))*tanh((9.81*0.25/m)...
      ^(1/2)*4)-9.81/(2*m)*sech(sqrt(9.81*0.25/m)*4)^2;
» newtraph(y,dy,140,0.00001)
ans =
      142.7376
```

■

4.3. SECANT METHODS

As in Example 6.4, a potential problem in implementing the Newton-Raphson method is the evaluation of the derivative. Although this is not inconvenient for polynomials and many other functions, there are certain functions whose derivatives may be difficult or inconvenient to evaluate. For these cases, the derivative can be approximated by a backward finite divided difference:

$$f'(x_i) \cong \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}$$

This approximation can be substituted into Eq. (6.6) to yield the following iterative equation:

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)} \tag{6.8}$$

Equation (6.8) is the formula for the *secant method*. Notice that the approach requires two initial estimates of x. However, because $f(x)$ is not required to change signs between the estimates, it is not classified as a bracketing method.

Rather than using two arbitrary values to estimate the derivative, an alternative approach involves a fractional perturbation of the independent variable to estimate $f'(x)$,

$$f'(x_i) \cong \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i}$$

where δ = a small perturbation fraction. This approximation can be substituted into Eq. (6.6) to yield the following iterative equation:

$$x_{i+1} = x_i - \frac{\delta x_i f(x_i)}{f(x_i + \delta x_i) - f(x_i)} \tag{6.9}$$

We call this the *modified secant method*. As in the following example, it provides a nice means to attain the efficiency of Newton-Raphson without having to compute derivatives.

Example 4.5. Modified Secant Method

Problem Statement. Use the modified secant method to determine the mass of the bungee jumper with a drag coefficient of 0.25 kg/m to have a velocity of 36 m/s after 4 s of free fall. Note: The acceleration of gravity is $9.81 m/s^2$. Use an initial guess of 50 kg and a value of 10^{-6} for the perturbation fraction.

Solution. Inserting the parameters into Eq. (6.9) yields

First iteration:

$$x_0 = 50 \qquad f(x_0) = -4.57938708$$

$$x_0 + \delta x_0 = 50.00005 \qquad f(x_0 + \delta x_0) = -4.579381118$$

$$x_1 = 50 - \frac{10^{-6}(50)(-4.57938708))}{-4.579381118 - (-4.57938708)} = 88.39931 (|\varepsilon_t| = 38.1\%; |\varepsilon_a| = 43.4\%)$$

Second iteration:

$$x_1 = 88.39931 \qquad f(x_1) = -1.69220771$$

$$x_1 + \delta x_1 = 88.39940 \qquad f(x_1 + \delta x_1) = -1.692203516$$

$$x_2 = 88.39931 - \frac{10^{-6}(88.39931)(-1.69220771)}{-1.692203516 - (-1.69220771)} = 124.08970 (|\varepsilon_t| = 13.1\%; |\varepsilon_a| = 28.76\%)$$

The calculation can be continued to yield

i	x_i	$\lvert\varepsilon_t\rvert$, %	$\lvert\varepsilon_a\rvert$, %
0	50.0000	64.971	
1	88.3993	38.069	43.438
2	124.0897	13.064	28.762
3	140.5417	1.538	11.706
4	142.7072	0.021	1.517
5	142.7376	4.1×10^{-6}	0.021
6	142.7376	3.4×10^{-12}	4.1×10^{-6}

The choice of a proper value for δ is not automatic. If δ is too small, the method can be swamped by round-off error caused by subtractive cancellation in the denominator of Eq. (6.9). If it is too big, the technique can become inefficient and even divergent. However, if chosen correctly, it provides a nice alternative for cases where evaluating the derivative is difficult and developing two initial guesses is inconvenient.

Further, in its most general sense, a univariate function is merely an entity that returns a single value in return for values sent to it. Perceived in this sense, functions are not always simple formulas like the one-line equations solved in the preceding examples in this chapter. For example, a function might consist of many lines of code that could take a significant amount of execution time to evaluate. In some cases, the function might even represent an independent computer program. For such cases, the secant and modified secant methods are valuable.

■

4.4. BRENT'S METHOD

Wouldn't it be nice to have a hybrid approach that combined the reliability of bracketing with the speed of the open methods? *Brent's root-location method* is a clever algorithm that does just that by applying a speedy open method wherever possible, but reverting to a reliable bracketing method if necessary. The approach was developed by Richard Brent (1973) based on an earlier algorithm of Theodorus Dekker (1969).

The bracketing technique is the trusty bisection method (Sec. 5.4), whereas two different open methods are employed. The first is the secant method described in Sec. 6.3. As explained next, the second is inverse quadratic interpolation.

4.4.1. Inverse Quadratic Interpolation

Inverse quadratic interpolation is similar in spirit to the secant method. As in Fig. 6.8a, the secant method is based on computing a straight line that goes through two guesses. The intersection of this straight line with the x axis represents the new root estimate. For this reason, it is sometimes referred to as a *linear interpolation method.*

Now suppose that we had three points. In that case, we could determine a quadratic function of x that goes through the three points (Fig. 6.8b). Just as with the linear secant method, the intersection of this parabola with the x axis would represent the new root estimate. And as illustrated in Fig. 6.8b, using a curve rather than a straight line often yields a better estimate.

Although this would seem to represent a great improvement, the approach has a fundamental flaw: it is possible that the parabola might not intersect the x axis! Such would be the case when the resulting parabola had complex roots. This is illustrated by the parabola, $y = f(x)$, in Fig. 6.9.

The difficulty can be rectified by employing inverse quadratic interpolation. That is, rather than using a parabola in x, we can fit the points with a parabola in y. This amounts to reversing the axes and creating a "sideways" parabola [the curve, $x = f(y)$, in Fig. 6.9].

If the three points are designated as (x_{i-2}, y_{i-2}), (x_{i-1}, y_{i-1}), and (x_i, y_i), a quadratic function of y that passes through the points can be generated as

$$g(y) = \frac{(y-y_{i-1})(y-y_i)}{(y_{i-2}-y_{i-1})(y_{i-2}-y_i)}x_{i-2} + \frac{(y-y_{i-2})(y-y_i)}{(y_{i-1}-y_{i-2})(y_{i-1}-y_i)}x_{i-1} + \frac{(y-y_{i-2})(y-y_{i-1})}{(y_i-y_{i-2})(y_i-y_{i-1})}x_i \tag{6.10}$$

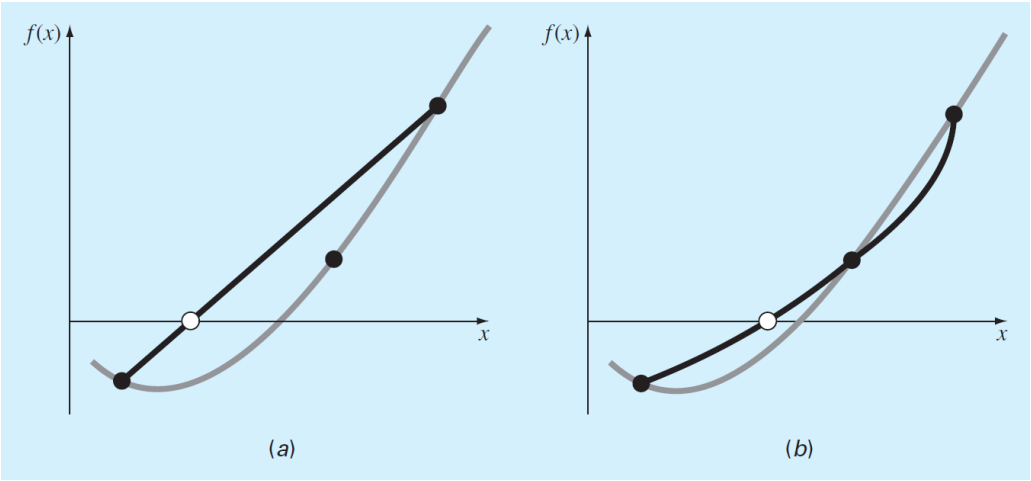


Figure 4.8: Comparison of (a) the secant method and (b) inverse quadratic interpolation. Note that the approach in (b) is called "inverse" because the quadratic function is written in y rather than in x.

As we will learn in Sec. 18.2, this form is called a *Lagrange polynomial*. The root, x_{i+1}, corresponds to $y = 0$, which when substituted into Eq. (6.10) yields

$$x_{i+1} = \frac{y_{i-1}y_i}{(y_{i-2}-y_{i-1})(y_{i-2}-y_i)}x_{i-2} + \frac{y_{i-2}y_i}{(y_{i-1}-y_{i-2})(y_{i-1}-y_i)}x_{i-1} + \frac{y_{i-2}y_{i-1}}{(y_i-y_{i-2})(y_i-y_{i-1})}x_i \tag{6.11}$$

As shown in Fig. 6.9, such a "sideways" parabola always intersects the x axis.

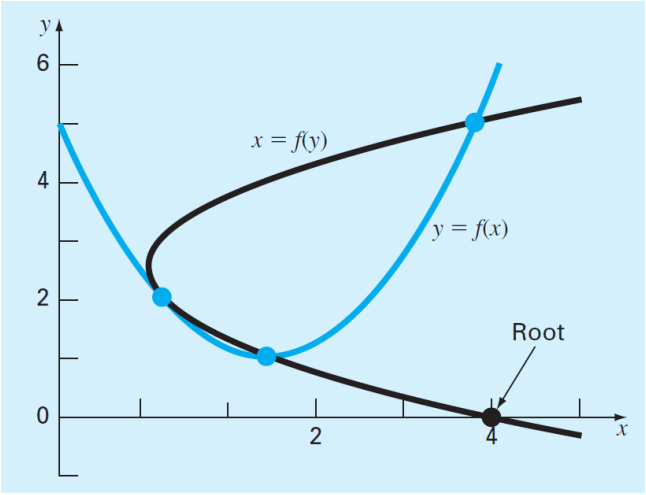


Figure 4.9: Two parabolas fit to three points. The parabola written as a function of $x, y = f(x)$, has complex roots and hence does not intersect the x axis. In contrast, if the variables are reversed, and the parabola developed as $x = f(y)$, the function does intersect the x axis.

Example 4.6. Inverse Quadratic Interpolation

Problem Statement. Develop quadratic equations in both x and y for the data points depicted in Fig. 6.9: $(1,2)$, $(2,1)$, and $(4,5)$. For the first, $y = f(x)$, employ the quadratic formula to illustrate that the roots are complex. For the latter, $x = g(y)$, use inverse quadratic interpolation (Eq. 6.11) to determine the root estimate.

Solution. By reversing the x's and y's, Eq. (6.10) can be used to generate a quadratic in x as

$$f(x) = \frac{(x-2)(x-4)}{(1-2)(1-4)}2 + \frac{(x-1)(x-4)}{(2-1)(2-4)}1 + \frac{(x-1)(x-2)}{(4-1)(4-2)}5$$

or collecting terms

$$f(x) = x^2 - 4x + 5$$

This equation was used to generate the parabola, $y = f(x)$, in Fig. 6.9. The quadratic formula can be used to determine that the roots for this case are complex,

$$x = \frac{4 \pm \sqrt{(-4)^2 - 4(1)(5)}}{2} = 2 \pm i$$

Equation (6.10) can be used to generate the quadratic in y as

$$g(y) = \frac{(y-1)(y-5)}{(2-1)(2-5)}1 + \frac{(y-2)(y-5)}{(1-2)(1-5)}2 + \frac{(y-2)(y-1)}{(5-2)(5-1)}4$$

or collecting terms:

$$g(y) = 0.5x^2 - 2.5x + 4$$

Finally, Eq. (6.11) can be used to determine the root as

$$x_{i+1} = \frac{-1(-5)}{(2-1)(2-5)}1 + \frac{-2(-5)}{(1-2)(1-5)}2 + \frac{-2(-1)}{(5-2)(5-1)}4 = 4$$

■

Before proceeding to Brent's algorithm, we need to mention one more case where inverse quadratic interpolation does not work. If the three y values are not distinct (i.e., $y_{i-2} = y_{i-1}$ or $y_{i-1} = y_i$), an inverse quadratic function does not exist. So this is where the secant method comes into play. If we arrive at a situation where the y values are not distinct, we can always revert to the less efficient secant method to generate a root using two of the points. If $y_{i-2} = y_{i-1}$, we use the secant method with x_{i-1} and x_i. If $y_{i-1} = y_i$, we use x_{i-2} and x_{i-1}.

4.5. BRENT'S METHOD ALGORITHM

The general idea behind the *Brent's root-finding method* is whenever possible to use one of the quick open methods. In the event that these generate an unacceptable result (i.e., a root estimate that falls outside the bracket), the algorithm reverts to the more conservative bisection method. Although bisection may be slower, it generates an estimate guaranteed to fall within the bracket. This process is then repeated until the root is located to within an acceptable tolerance. As might be expected, bisection typically dominates at first but as the root is approached, the technique shifts to the faster open methods.

Figure 6.10 presents a function based on a MATLAB M-file developed by Cleve Moler (2004). It represents a stripped down version of the `fzero` function which is the professional root-location function employed in MATLAB. For that reason, we call the simplified version: `fzerosimp`. Note that it requires another function `f` that holds the equation for which the root is being evaluated.

The `fzerosimp` function is passed two initial guesses that must bracket the root. Then, the three variables defining the search interval `(a,b,c)` are initialized, and `f` is evaluated at the endpoints.

```
function b = fzerosimp(xl,xu)
a = xl; b = xu; fa = f(a); fb = f(b);
c = a; fc = fa; d = b - c; e = d;
while (1)
  if fb == 0, break, end
  if sign(fa) == sign(fb) %If needed, rearrange points
    a = c; fa = fc; d = b - c; e = d;
  end
  if abs(fa) < abs(fb)
    c = b; b = a; a = c;
    fc = fb; fb = fa; fa = fc;
  end
  m = 0.5*(a - b); %Termination test and possible exit
  tol = 2 * eps * max(abs(b), 1);
  if abs(m) <= tol | fb == 0.
    break
  end
  %Choose open methods or bisection
  if abs(e) >= tol & abs(fc) > abs(fb)
    s = fb/fc;
    if a == c                  %Secant method
      p = 2*m*s;
      q = 1 - s;
    else             %Inverse quadratic interpolation
      q = fc/fa; r = fb/fa;
      p = s * (2*m*q * (q - r) - (b - c)*(r - 1));
      q = (q - 1)*(r - 1)*(s - 1);
    end
    if p > 0, q = -q; else p = -p; end;
    if 2*p < 3*m*q - abs(tol*q) & p < abs(0.5*e*q)
      e = d; d = p/q;
    else
      d = m; e = m;
    end
  else                                %Bisection
    d = m; e = m;
  end
  c = b; fc = fb;
  if abs(d) > tol, b=b+d; else b=b-sign(b-a)*tol; end
  fb = f(b);
end
```

Figure 4.10: Function for Brent's root-finding algorithm based on a MATLAB M-file developed by Cleve Moler (2005).

A main loop is then implemented. If necessary, the three points are rearranged to satisfy the conditions required for the algorithm to work effectively. At this point, if the stopping criteria are met, the loop is terminated. Otherwise, a decision structure chooses among the three methods and checks whether the outcome is acceptable. Afinal section then evaluates `f` at the new point and the loop is repeated. Once the stopping criteria are met, the loop terminates and the final root estimate is returned.

4.6. MATLAB FUNCTION: FZERO

The `fzero` function is designed to find the real root of a single equation. A simple representation of its syntax is

```
fzero(function, x0)
```

where `function` is the name of the function being evaluated, and `x0` is the initial guess. Note that two guesses that bracket the root can be passed as a vector:

```
fzero(function,[x0 x1])
```

where x0 and x1 are guesses that bracket a sign change.

Here is a simple MATLAB session that solves for the root of a simple quadratic: $x^2 - 9$. Clearly two roots exist at -3 and 3. To find the negative root:

```
» x = fzero(@(x) x^2-9,-4)
x =
      3
```

If we want to find the positive root, use a guess that is near it:

```
» x = fzero(@(x) x^2-9,4
)   x =
      3
```

If we put in an initial guess of zero, it finds the negative root:

```
» x = fzero(@(x) x^2-9,0)
x =
      -3
```

If we wanted to ensure that we found the positive root, we could enter two guesses as in

```
» x = fzero(@(x) x^2-9,[0 4])
x =
      3
```

Also, if a sign change does not occur between the two guesses, an error message is displayed

```
» x = fzero(@(x) x^2-9,[-4 4])
???  Error using ==> fzero
The function values at the interval endpoints must differ in sign.
```

The `fzero` function works as follows. If a single initial guess is passed, it first performs a search to identify a sign change. This search differs from the incremental search described in Section 5.3.1, in that the search starts at the single initial guess and then takes increasingly bigger steps in both the positive and negative directions until a sign change is detected.

Thereafter, the fast methods (secant and inverse quadratic interpolation) are used unless an unacceptable result occurs (e.g., the root estimate falls outside the bracket). If an unacceptable result happens, bisection is implemented until an acceptable root is obtained with one of the fast methods. As might be expected, bisection typically dominates at first but as the root is approached, the technique shifts to the faster methods.

A more complete representation of the fzero syntax can be written as

```
[x,fx] = fzero(function,x0,options,p1,p2,...)
```

where `[x,fx]` = a vector containing the root `x` and the function evaluated at the root `fx`, `options` is a data structure created by the `optimset` function, and `p1, p2...` are any parameters that the function requires. Note that if you desire to pass in parameters but not use the `options`, pass an empty vector [] in its place.

The `optimset` function has the syntax

```
options = optimset('par1',val1,'par2',val2,...)
```

where the parameter par_i has the value val_i. A complete listing of all the possible parameters can be obtained by merely entering `optimset` at the command prompt. The parameters that are commonly used with the `fzero` function are

`display`: When set to `'iter'` displays a detailed record of all the iterations.
`tolx`: A positive scalar that sets a termination tolerance on `x`.

Example 4.7. The `fzero` and `optimset` Functions

Problem Statement. Recall that in Example 6.3, we found the positive root of $f(x) = x^{10} - 1$ using the Newton-Raphson method with an initial guess of 0.5. Solve the same problem with `optimset` and `fzero`.

Solution. An interactive MATLAB session can be implemented as follows:

```
» options = optimset('display','iter');
» [x,fx] = fzero(@(x) x^10-1,0.5,options)
```

```
Func-count      x          f(x)         Procedure
    1            0.5     -0.999023       initial
    2          0.485858  -0.999267       search
    3          0.514142  -0.998709       search
    4             0.48   -0.999351       search
    5             0.52   -0.998554       search
    6          0.471716  -0.999454       search
    •
    •
    •
   23          0.952548  -0.385007       search
   24             -0.14         -1       search
   25              1.14    2.70722       search

    Looking for a zero in the interval [-0.14, 1.14]

   26          0.205272         -1       interpolation
   27          0.672636  -0.981042       bisection
   28          0.906318  -0.626056       bisection
   29           1.02316   0.257278       bisection
   30          0.989128  -0.103551       interpolation
   31          0.998894  -0.0110017      interpolation
   32           1.00001  7.68385e-005    interpolation
   33                 1 -3.83061e-007    interpolation
   34                 1  -1.3245e-011    interpolation
   35                 1             0    interpolation

    Zero found in the interval:  [-0.14, 1.14].
    x =
          1
    fx =
          0
```

Thus, after 25 iterations of searching, `fzero` finds a sign change. It then uses interpolation and bisection until it gets close enough to the root so that interpolation takes over and rapidly converges on the root. Suppose that we would like to use a less stringent tolerance. We can use the `optimset` function to set a low maximum tolerance and a less accurate estimate of the root results:

```
» options = optimset ('tolx', 1e-3);
» [x,fx] = fzero(@(x) x^10-1,0.5,options)
x =
      1.0009
fx =
      0.0090
```

■

4.7. POLYNOMIALS

Chapter 5

Optimization

CHAPTER OBJECTIVES

The primary objective of this chapter is to introduce you to how optimization can be used to determine minima and maxima of both one-dimensional and multidimensional functions. Specific objectives and topics covered are

- Understanding why and where optimization occurs in engineering and scientific problem solving.
- Recognizing the difference between one-dimensional and multidimensional optimization.
- Distinguishing between global and local optima.
- Knowing how to recast a maximization problem so that it can be solved with a minimizing algorithm.
- Being able to define the golden ratio and understand why it makes onedimensional optimization efficient.
- Locating the optimum of a single-variable function with the golden-section search.
- Locating the optimum of a single-variable function with parabolic interpolation.
- Knowing how to apply the `fminbnd` function to determine the minimum of a one-dimensional function.
- Being able to develop MATLAB contour and surface plots to visualize twodimensional functions.
- Knowing how to apply the `fminsearch` function to determine the minimum of a multidimensional function.

YOUâĂŹVE GOT A PROBLEM

An object like a bungee jumper can be projected upward at a specified velocity. If it is subject to linear drag, its altitude as a function of time can be computed as

$$z = z_0 + \frac{m}{c}(v_0 + \frac{mg}{c})(1 - e^{-(c/m)t}) - \frac{mg}{c}t \tag{7.1}$$

Order n	$f^{(n)}(x)$	$f(\pi/3)$	$\lvert\varepsilon_t\rvert$
0	$\cos x$	0.707106781	41.4
1	$-\sin x$	0.521986659	4.40
2	$-\cos x$	0.497754491	0.449
3	$\sin x$	0.499869147	2.62×10^{-2}
4	$\cos x$	0.500007551	1.51×10^{-3}
5	$-\sin x$	0.500000304	6.08×10^{-5}
6	$-\cos x$	0.499999988	2.44×10^{-6}

Figure 5.1: Elevation as a function of time for an object initially projected upward with an initial velocity.

where z = altitude (m) above the earth's surface (defined as $z = 0$), z_0 = the initial altitude (m), m = mass (kg), c = a linear drag coefficient (kg/s), v_0 = initial velocity (m/s), and t = time (s). Note that for this formulation, positive velocity is considered to be in the upward direction. Given the following parameter values: $g = 9.81 m/s^2$, z_0 = 100 m, v_0 = 55 m/s, m = 80 kg, and c = 15 kg/s, Eq. (7.1) can be used to calculate the jumper's altitude. As displayed in Fig. 7.1, the jumper rises to a peak elevation of about 190 m at about t = 4 s. Suppose that you are given the job of determining the exact time of the peak elevation. The determination of such extreme values is referred to as optimization. This chapter will introduce you to how the computer is used to make such determinations.

5.1. A SIMPLE MATHEMATICAL MODEL

In the most general sense, optimization is the process of creating something that is as effective as possible. As engineers, we must continuously design devices and products that perform tasks in an efficient fashion for the least cost. Thus, engineers are always confronting optimization problems that attempt to balance performance and limitations. In addition, scientists have interest in optimal phenomena ranging from the peak elevation of projectiles to the minimum free energy.

From a mathematical perspective, optimization deals with finding the maxima and minima of a function that depends on one or more variables. The goal is to determine the values of the variables that yield maxima or minima for the function. These can then be substituted back into the function to compute its optimal values.

Although these solutions can sometimes be obtained analytically, most practical optimization problems require numerical, computer solutions. From a numerical standpoint, optimization is similar in spirit to the root-location methods we just covered in Chaps. 5 and 6. That is, both involve guessing and searching for a point on a function. The fundamental difference between the two types of problems is illustrated in Fig. 7.2. Root location involves searching for the location where the function equals zero. In contrast, optimization involves searching for the function's extreme points.

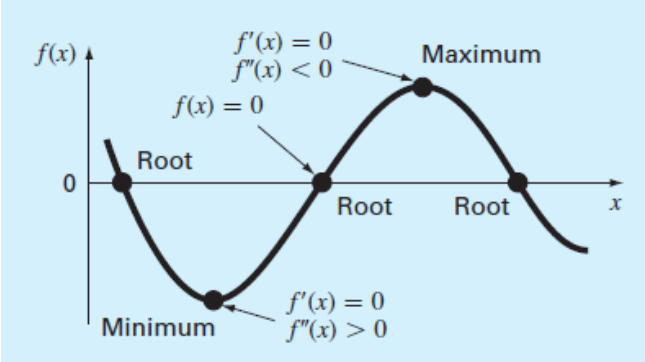


Figure 5.2: A function of a single variable illustrating the difference between roots and optima.

As can be seen in Fig. 7.2, the optimums are the points where the curve is flat. In mathematical terms, this corresponds to the x value where the derivative $f'(x)$ is equal to zero. Additionally, the second derivative, $f''(x)$, indicates whether the optimum is a minimum or a maximum: if $f''(x) > 0$, the point is a maximum; if $f''(x) > 0$, the point is a minimum.

Now, understanding the relationship between roots and optima would suggest a possible strategy for finding the latter. That is, you can differentiate the function and locate the root (i.e., the zero) of the new function. In fact, some optimization methods do just this by solving the root problem: $f''(x) = 0$.

Example 1. Determining the Optimum Analytically by Root Location

Problem Statement: Determine the time and magnitude of the peak elevation based on Eq. (7.1). Use the following parameter values for your calculation: $g = 9.81 m/s^2$, z0 = 100 m, v0 = 55 m/s, m = 80 kg, and c = 15 kg/s.
Solution: Equation (7.1) can be differentiated to give.

$$\frac{dz}{dt} = v_0 e^{-(c/m)t} - \frac{mg}{c}(1 - e^{-(c/m)t})$$

Note that because $v = dz/dt$, this is actually the equation for the velocity. The maximum elevation occurs at the value of t that drives this equation to zero. Thus, the problem amounts to determining the root. For this case, this can be accomplished by setting the derivative to zero and solving Eq. (E7.1.1) analytically for

$$t = \frac{m}{c} ln(1 + \frac{cv_0}{mg})$$

Substituting the parameters gives

$$t = \frac{80}{15} ln(1 + \frac{15(55)}{80(9.81)}) = 3.83166s$$

This value along with the parameters can then be substituted into Eq. (7.1) to compute the maximum elevation as

$$z = 100 + \frac{80}{15}(50 + \frac{80(9.81)}{15})(1 - e^{-(15/80)3.83166}) - \frac{80(9.81)}{15}(3.83166) = 192.8609m$$

We can verify that the result is a maximum by differentiating Eq. (E7.1.1) to obtain the second derivative

$$\frac{d^2z}{dt^2} = -\frac{c}{m} v_0 e^{-(c/m)t} - g e^{-(c/m)t} = -9.81 \frac{m}{s^2}$$

The fact that the second derivative is negative tells us that we have a maximum. Further, the result makes physical sense since the acceleration should be solely equal to the force of gravity at the maximum when the vertical velocity (and hence drag) is zero.

Although an analytical solution was possible for this case, we could have obtained the same result using the root-location methods described in Chaps. 5 and 6. This will be left as a homework exercise.

Although it is certainly possible to approach optimization as a roots problem, a variety of direct numerical optimization methods are available. These methods are available for both one-dimensional and multidimensional problems. As the name implies, one-dimensional problems involve functions that depend on a single dependent variable. As in Fig. 7.3a, the search then consists of climbing or descending one-dimensional peaks and valleys. Multidimensional problems involve functions that depend on two or more dependent variables.

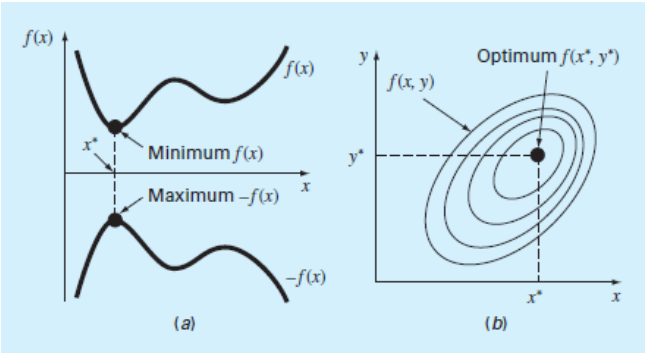


Figure 5.3: (a) One-dimensional optimization. This figure also illustrates how minimization of $f(x)$ is equivalent to the maximization of $-f(x)$. (b) Two-dimensional optimization. Note that this figure can be taken to represent either a maximization (contours increase in elevation up to the maximum like a mountain) or a minimization (contours decrease in elevation down to the minimum like a valley).

In the same spirit, a two-dimensional optimization can again be visualized as searching out peaks and valleys (Fig. 7.3b). However, just as in real hiking, we are not constrained to walk a single direction; instead the topography is examined to efficiently reach the goal.

Finally, the process of finding a maximum versus finding a minimum is essentially identical because the same value $x*$ both minimizes $f(x)$ and maximizes $-f(x)$. This equivalence is illustrated graphically for a one-dimensional function in Fig. 7.3a.

In the next section, we will describe some of the more common approaches for onedimensional optimization. Then we will provide a brief description of how MATLAB can be employed to determine optima for multidimensional functions.

5.2. ONE-DIMENSIONAL OPTIMIZATION

This section will describe techniques to find the minimum or maximum of a function of a single variable $f(x)$. A useful image in this regard is the one-dimensional "roller coaster" -like function depicted in Fig. 7.4. Recall from Chaps. 5 and 6 that root location was complicated by the fact that several roots can occur for a single function. Similarly, both local and global optima can occur in optimization.

A *global optimum* represents the very best solution. A *local optimum*, though not the very best, is better than its immediate neighbors. Cases that include local optima are called *multimodal*. In such cases, we will almost always be interested in finding the global optimum. In addition, we must be concerned about mistaking a local result for the global optimum.

Just as in root location, optimization in one dimension can be divided into bracketing and open methods. As described in the next section, the golden-section search is an example of a bracketing method that is very similar in spirit to the bisection method for root location. This is followed by a somewhat more sophisticated bracketing approach-parabolic interpolation. We will then show how these two methods are combined and implemented with MATLAB's `fminbnd` function.

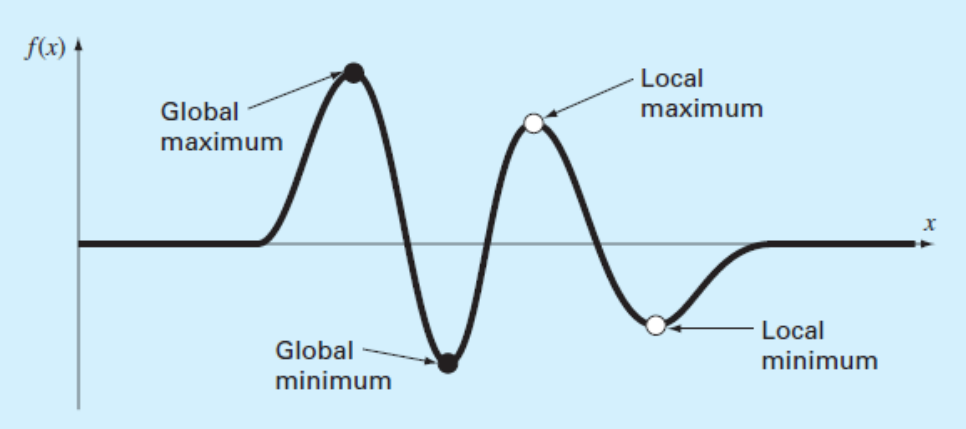


Figure 5.4: (a) A function that asymptotically approaches zero at plus and minus ∞ and has two maximum and two minimum points in the vicinity of the origin. The two points to the right are local optima, whereas the two to the left are global.

5.2.1. Golden-Section Search

In many cultures, certain numbers are ascribed magical qualities. For example, we in theWest are all familiar with "lucky 7" and "Friday the 13th." Beyond such superstitious quantities, there are several well-known numbers that have such interesting and powerful mathematical properties that they could truly be called "magical". The most common of these are the ratio of a circle's circumference to its diameter π and the base of the natural logarithm e.

Although not as widely known, the golden ratio should surely be included in the pantheon of remarkable numbers. This quantity, which is typically represented by the Greek letter ϕ (pronounced: fee), was originally defined by Euclid (ca. 300 BCE) because of its role in the construction of the pentagram or five-pointed star. As depicted in Fig. 7.5, Euclid's definition reads: "A straight line is said to have been cut in extreme and mean ratio when, as the whole line is to the greater segment, so is the greater to the lesser."

The actual value of the golden ratio can be derived by expressing Euclid's definition as

$$\frac{l_1+l_2}{l_1}=\frac{l_1}{l_2} \tag{7.2}$$

Multiplying by l_1/l_2 and collecting terms yields

$$\phi^2 - \phi - 1 = 0 \tag{7.3}$$

where $\phi = l_1/l_2$. The positive root of this equation is the golden ratio:

$$\phi = \frac{1+\sqrt{5}}{2} = 1.61803398874989... \tag{7.4}$$

The golden ratio has long been considered aesthetically pleasing in Western cultures. In addition, it arises in a variety of other contexts including biology. For our purposes, it provides the basis for the golden-section search, a simple, general-purpose method for determining the optimum of a single-variable function.

The golden-section search is similar in spirit to the bisection approach for locating roots in Chap. 5. Recall that bisection hinged on defining an interval, specified by a lower guess (x_l) and an upper guess (x_u) that bracketed a single root. The presence of a root between these bounds was verified by determining that $f(x_l)$ and $f(x_u)$ had different signs. The root was then estimated as the midpoint of this interval:

$$x_r = \frac{x_l + x_u}{2} \tag{7.5}$$

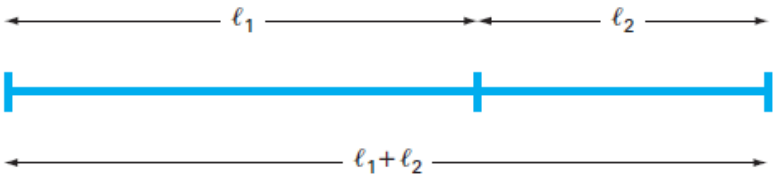


Figure 5.5: Euclid's definition of the golden ratio is based on dividing a line into two segments so that the ratio of the whole line to the larger segment is equal to the ratio of the larger segment to the smaller segment. This ratio is called the golden ratio.

The final step in a bisection iteration involved determining a new smaller bracket. This was done by replacing whichever of the bounds x"$_l$ or x_u had a function value with the same sign as $f(x_r)$. Akey advantage of this approach was that the new value x_r replaced one of the old bounds.

Now suppose that instead of a root, we were interested in determining the minimum of a one-dimensional function. As with bisection, we can start by defining an interval that contains a single answer. That is, the interval should contain a single minimum, and hence is called *unimodal*. We can adopt the same nomenclature as for bisection, where x_l and x_u defined the lower and upper bounds, respectively, of such an interval. However, in contrast to bisection, we need a new strategy for finding a minimum within the interval. Rather than using a single intermediate value (which is sufficient to detect a sign change, and hence a zero), we would need two intermediate function values to detect whether a minimum occurred.

The key to making this approach efficient is the wise choice of the intermediate points. As in bisection, the goal is to minimize function evaluations by replacing old values with new values. For bisection, this was accomplished by choosing the midpoint. For the golden-section search, the two intermediate points are chosen according to the golden ratio:

$$x_1 = x_l + d \tag{7.6}$$

$$x_2 = x_u - d \tag{7.7}$$

where

$$d = (\phi - 1)(x_u - x_l) \tag{7.8}$$

The function is evaluated at these two interior points. Two results can occur:

1. If, as in Fig. 7.6a, $f(x_1) < f(x_2)$, then $f(x_1)$ is the minimum, and the domain of x to the left of x_2, from x_l to x_2, can be eliminated because it does not contain the minimum. For this case, x_2 becomes the new x_l for the next round.

2. If $f(x_2) < f(x_1)$, then $f(x_2)$ is the minimum and the domain of x to the right of x_1, from x_1 to x_u would be eliminated. For this case, x_1 becomes the new x_u for the next round.

Now, here is the real benefit from the use of the golden ratio. Because the original x_1 and x_2 were chosen using the golden ratio, we do not have to recalculate all the function values for the next iteration. For example, for the case illustrated in Fig. 7.6, the old x_1 becomes the new x_2. This means that we already have the value for the new $f(x_2)$, since it is the same as the function value at the old x_1.

To complete the algorithm, we need only determine the new x_1. This is done with Eq. (7.6) with d computed with Eq. (7.8) based on the new values of x_l and x_u. A similar approach would be used for the alternate case where the optimum fell in the left subinterval. For this case, the new x_2 would be computed with Eq. (7.7).

As the iterations are repeated, the interval containing the extremum is reduced rapidly. In fact, each round the interval is reduced by a factor of ϕ - 1(about 61.8%). That means that after 10 rounds, the interval is shrunk to about

0.618^{10} or 0.008 or 0.8% of its initial length. After 20 rounds, it is about 0.0066%. This is not quite as good as the reduction achieved with bisection (50%), but this is a harder problem.

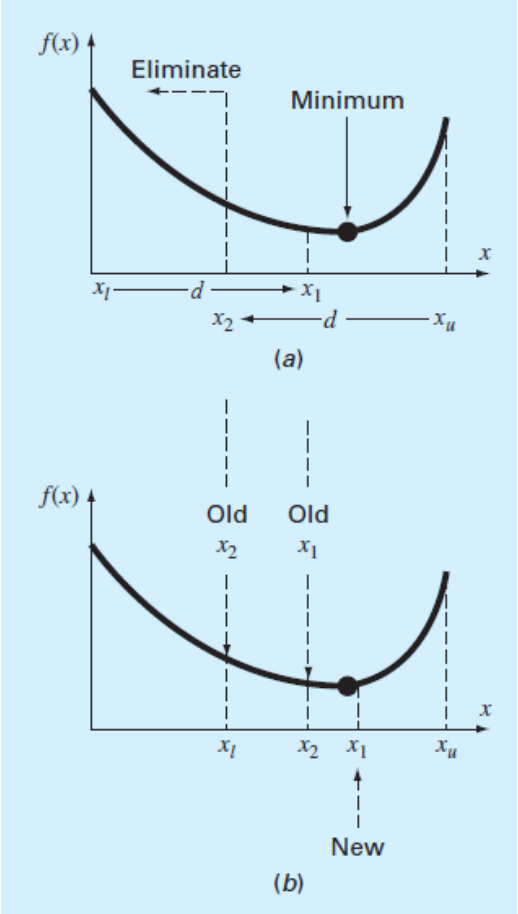


Figure 5.6: (a) The initial step of the golden-section search algorithm involves choosing two interior points according to the golden ratio. (b) The second step involves defining a new interval that encompasses the optimum.

Example 2. Golden-Section Search

Problem Statement: Use the golden-section search to find the minimum of

$$f(x) = \frac{x^2}{10} - 2sinx$$

within the interval from $x_l = 0$ to $x_u = 4$

Solution: First, the golden ratio is used to create the two interior points:

$$d = 0.61803(4-0) = 2.4721$$

$$x_1 = 0 + 2.4721 = 2.4721$$

$$x_2 = 4 - 2.4721 = 1.5279$$

The function can be evaluated at the interior points:

$$f(x_2) = \frac{1.5279^2}{10} - 2sin(1.5279) = -1.7647)$$

$$f(x_1) = \frac{2.4721^2}{10} - 2sin(2.4721) = -0.6300)$$

Because $f(x_2) < f(x_1)$, our best estimate of the minimum at this point is that it is located at x = 1.5279 with a value of $f(x) = -1.7647$. In addition, we also know that the minimum is in the interval defined by x_l, x_2, and x_1. Thus, for the next iteration, the lower bound remains $x_l = 0$, and x_1 becomes the upper bound, that is, $x_u = 2.4721$. In addition, the former x_2 value becomes the new x_1, that is, $x_1 = 1.5279$. In addition, we do not have to recalculate $f(x_1)$, it was determined on the previous iteration as $f(1.5279) = -1.7647$.

All that remains is to use Eqs. (7.8) and (7.7) to compute the new value of d and x_2:

$$d = 0.61803(2.4721-0) = 1.5279$$

$$x_2 = 2.4721 - 1.5279 = 0.9443$$

The function evaluation at x_2 is $f(0.9943) = -1.5310$. Since this value is less than the function value at x1, the minimum is $f(1.5279) = -1.7647$, and it is in the interval prescribed by x_2, x_1, and x_u. The process can be repeated, with the results tabulated here:

i	x_l	$f(x_l)$	x_2	$f(x_2)$	x_l	$f(x_l)$	x_u	$f(x_u)$	d
1	0	0	1.5279	-1.7647	2.4721	-0.6300	4.0000	3.1136	2.4721
2	0	0	0.9443	-1.5310	1.5279	-1.7647	2.4721	-0.6300	1.5279
3	0.9443	-1.5310	1.5279	-1.7647	1.8885	-1.5432	2.4721	-0.6300	0.9443
4	0.9443	-1.5310	1.3050	-1.7595	1.5279	-1.7647	1.8885	-1.5432	0.5836
5	1.3050	-1.7595	1.5279	-1.7647	1.6656	-1.7136	1.8885	-1.5432	0.3607
6	1.3050	-1.7595	1.4427	-1.7755	1.5279	-1.7647	1.6656	-1.7136	0.2229
7	1.3050	-1.7595	1.3901	-1.7742	1.4427	-1.7755	1.5279	-1.7647	0.1378
8	1.3901	-1.7742	1.4427	-1.7755	1.4752	-1.7732	1.5279	-1.7647	0.0851

Note that the current minimum is highlighted for every iteration. After the eighth iteration, the minimum occurs at x = 1.4427 with a function value of -1.7755. Thus, the result is converging on the true value of -1.7757 at x = 1.4276.

Recall that for bisection (Sec. 5.4), an exact upper bound for the error can be calculated at each iteration. Using similar reasoning, an upper bound for golden-section search can be derived as follows: Once an iteration is complete, the optimum will either fall in one of two intervals. If the optimum function value is at x_2, it will be in the lower interval (x_l, x_2, x_1). If the optimum function value is at x_1, it will be in the upper interval (x_2, x_1, x_u). Because the interior points are symmetrical, either case can be used to define the error.

Looking at the upper interval (x_2, x_1, x_u), if the true value were at the far left, the maximum distance from the estimate would be

$$\begin{aligned}\Delta x_a &= x_1 - x_2\\ &= x_l + (\phi-1)(x_u - x_l) - x_u + (\phi-1)(x_u - x_l)\\ &= (x_l - x_u) + 2(\phi-1)(x_u - x_l)\\ &= (2\phi - 3)(x_u - x_l)\end{aligned}$$

or 0.2361 $(x_u - x_l)$. If the true value were at the far right, the maximum distance from the estimate would be

$$\begin{aligned}\Delta x_b &= x_u - x_1\\ &= x_u - x_l - (\phi-1)(x_u - x_l)\\ &= (x_u - x_l) - (\phi-1)(x_u - x_l)\\ &= (2-\phi)(x_u - x_l)\end{aligned}$$

or 0.3820 $(x_u - x_l)$. Therefore, this case would represent the maximum error. This result can then be normalized to the optimal value for that iteration x_opt to yield

$$\varepsilon_a = (2-\phi)\left|\frac{x_u - x_l}{x_{opt}}\right| \times 100\% \tag{7.9}$$

This estimate provides a basis for terminating the iterations.

An M-file function for the golden-section search for minimization is presented in Fig. 7.7. The function returns the location of the minimum, the value of the function, the approximate error, and the number of iterations.

The M-file can be used to solve the problem from Example 7.1.

```
>> g=9.81;v0=55;m=80;c=15;z0=100;
>> z=@(t) -(z0+m/c*(v0+m*g/c)*(1-exp(-c/m*t))-m*g/c*t);
>> [xmin,fmin,ea,iter]=goldmin(z,0,8)

xmin =
   3.8317
fmin =
  -192.8609
ea =
   6.9356e-005
```

Notice how because this is a maximization, we have entered the negative of Eq. (7.1). Consequently, `fmin` corresponds to a maximum height of 192.8609.

You may be wondering why we have stressed the reduced function evaluations of the golden-section search. Of course, for solving a single optimization, the speed savings would be negligible. However, there are two important contexts where minimizing the number of function evaluations can be important. These are

1. Many evaluations. There are cases where the golden-section search algorithm may be a part of a much larger calculation. In such cases, it may be called many times. Therefore, keeping function evaluations to a minimum could pay great dividends for such cases.

Figure 5.7: An M-file to determine the minimum of a function with the golden-section search.

2. Time-consuming evaluation. For pedagogical reasons, we use simple functions in most of our examples. You should understand that a function can be very complex and time-consuming to evaluate. For example, optimization can be used to estimate the parameters of a model consisting of a system of differential equations. For such cases, the "function" involves time-consuming model integration. Any method that minimizes such evaluations would be advantageous.

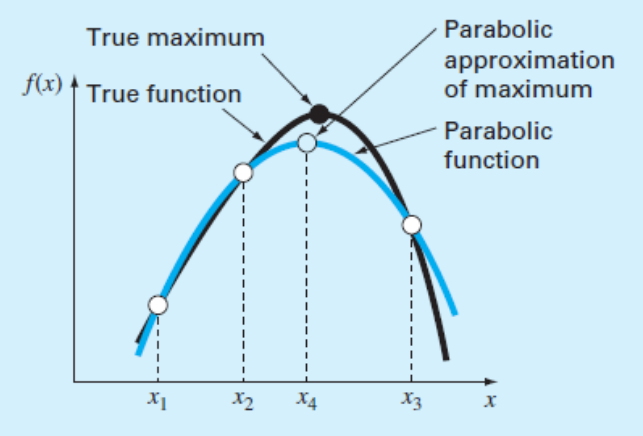


Figure 5.8: Graphical depiction of parabolic interpolation.

5.2.2. Parabolic Interpolation

Parabolic interpolation takes advantage of the fact that a second-order polynomial often provides a good approximation to the shape of f (x) near an optimum (Fig. 7.8).

Just as there is only one straight line connecting two points, there is only one parabola connecting three points. Thus, if we have three points that jointly bracket an optimum, we can fit a parabola to the points. Then we can differentiate it, set the result equal to zero, and solve for an estimate of the optimal x. It can be shown through some algebraic manipulations that the result is

$$x_4 = x_2 - \frac{1}{2}\frac{(x_2-x_1)^2[f(x_2)-f(x_3)]-(x_2-x_3)^2[f(x_2)-f(x_1)]}{(x_2-x_1)[f(x_2)-f(x_3)]-(x_2-x_3)[f(x_2)-f(x_1)]} \tag{7.10}$$

where $x1$, $x2$, and $x3$ are the initial guesses, and $x4$ is the value of x that corresponds to the optimum value of the parabolic fit to the guesses.

5.2.3. Quantification of Error of Linear Regression

Any line other than the one computed in Example 14.4 results in a larger sum of the squares of the residuals. Thus, the line is unique and in terms of our chosen criterion is a "best" line through the points. A number of additional properties of this fit can be elucidated by examining more closely the way in which residuals were computed. Recall that the sum of the squares is defined as [Eq. (14.12)]

$$S_r = \sum_{i=1}^{n}(y_i - a_0 - a_1 x_i)^2 \tag{14.17}$$

Notice the similarity between this equation and Eq. (14.4)

$$S_t = \sum(y_i - \bar{y})^2 \tag{14.18}$$

In Eq. (14.18), the square of the residual represented the square of the discrepancy between the data and a single estimate of the measure of central tendency-the mean. In Eq. (14.17), the square of the residual represents the square of the vertical distance between the data and another measure of central tendency-the straight line (5.9).

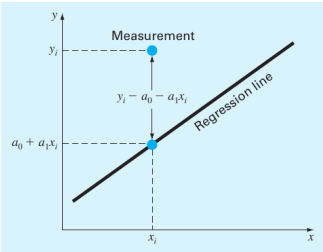


Figure 5.9: The residual in linear regression represents the vertical distance between a data point and the straight line.

The analogy can be extended further for cases where (1) the spread of the points around the line is of similar magnitude along the entire range of the data and (2) the distribution of these points about the line is normal. It can be demonstrated that if these criteria are met, least-squares regression will provide the best (i.e., the most likely) estimates of a_0 and a_1 (Draper and Smith, 1981). This is called the maximum likelihood principle in statistics. In addition, if these criteria are met, a "standard deviation" for the regression line can be determined as [compare with Eq. (14.3)]

$$s_{y/x} = \sqrt{\frac{S_r}{n-2}} \tag{14.19}$$

where $s_{y/x}$ is called the *standard error of the estimate*. The subscript notation "y/x" designates that the error is for a predicted value of y corresponding to a particular value of x. Also, notice that we now divide by $n-2$ because two data-derived estimates - a_0 and a_1 âĂŤ were used to compute S_r; thus, we have lost two degrees of freedom. As with our discussion of the standard deviation, another justification for dividing by $n-2$ is that there is no such thing as the âĂIJspread of dataâĂİ around a straight line connecting two points. Thus, for the case where $n = 2$, Eq. (14.19) yields a meaningless result of infinity.

Just as was the case with the standard deviation, the standard error of the estimate quantifies the spread of the data. However, $s_{y/x}$ quantifies the spread *around the regression line* as shown in 5.10 in contrast to the standard deviation s_y that quantified the *spread around the mean* (5.10).

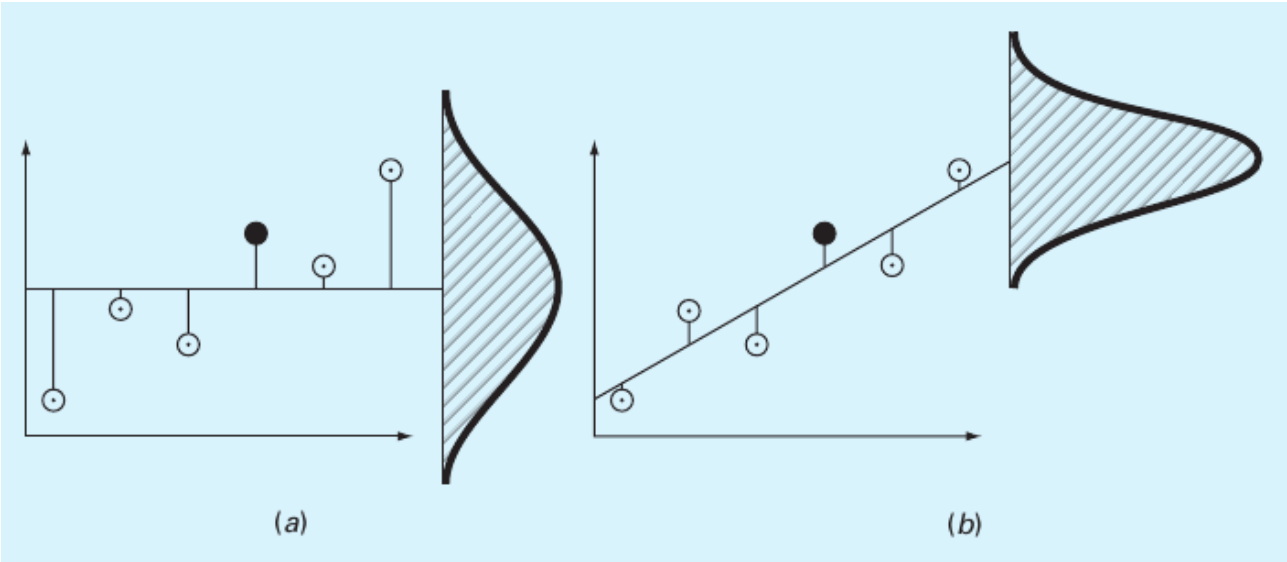


Figure 5.10: Regression data showing (a) the spread of the data around the mean of the dependent variable and (b) the spread of the data around the best-fit line. The reduction in the spread in going from (a) to (b), as indicated by the bell-shaped curves at the right, represents the improvement due to linear regression.

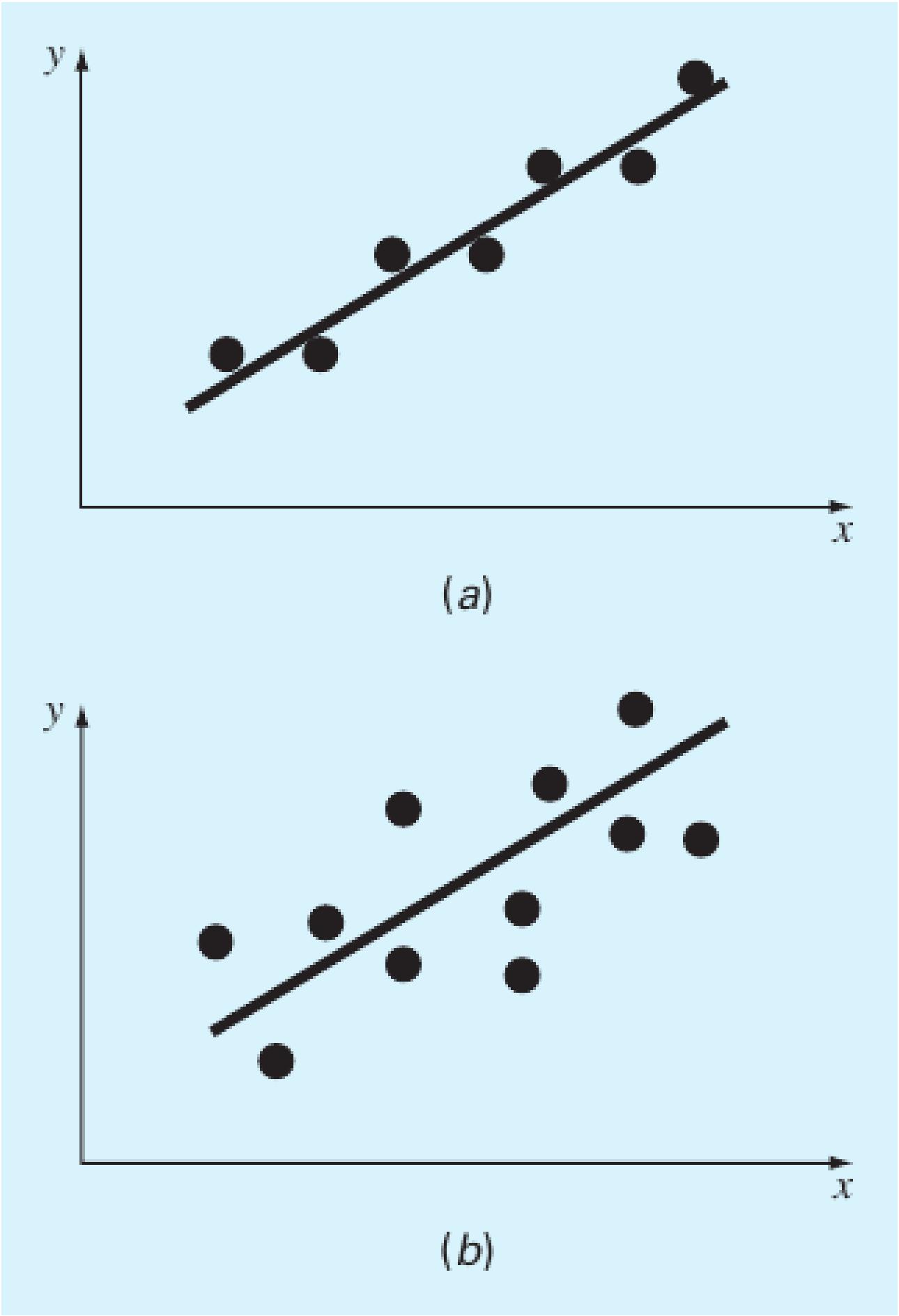


Figure 5.11: Examples of linear regression with (a) small and (b) large residual errors.

These concepts can be used to quantify the "goodness" of our fit. This is particularly useful for comparison of

several regressions (5.11). To do this, we return to the original data and determine the total sum of the squares around the mean for the dependent variable (in our case, y). As was the case for Eq. (14.18), this quantity is designated S_t. This is the magnitude of the residual error associated with the dependent variable prior to regression. After performing the regression, we can compute S_r , the sum of the squares of the residuals around the regression line with Eq. (14.17). This characterizes the residual error that remains after the regression. It is, therefore, sometimes called the unexplained sum of the squares. The difference between the two quantities, $S_t - S_r$, quantifies the improvement or error reduction due to describing the data in terms of a straight line rather than as an average value. Because the magnitude of this quantity is scale-dependent, the difference is normalized to S_t to yield

$$r^2 = \frac{S_t - S_r}{S_t} \tag{14.20}$$

where r^2 is called the *coefficient of determination* and r is the *correlation coefficient* ($= \sqrt{r^2}$). For a perfect fit, $S_r = 0$ and $r^2 = 1$, signifying that the line explains 100% of the variability of the data. For $r^2 = 0$, $S_r = S_t$ and the fit represents no improvement. An alternative formulation for r that is more convenient for computer implementation is

$$r = \frac{n\sum(x_i y_i) - (\sum x_i)(\sum y_i)}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \tag{14.21}$$

Before proceeding, a word of caution is in order. Although the coefficient of determination provides a handy measure of goodness-of-fit, you should be careful not to ascribe more meaning to it than is warranted. Just because r^2 is "close" to 1 does not mean that the fit is necessarily "good". For example, it is possible to obtain a relatively high value of r^2 when the underlying relationship between y and x is not even linear. Draper and Smith (1981) provide guidance and additional material regarding assessment of results for linear regression. In addition, at the minimum, you should always inspect a plot of the data along with your regression curve.

A nice example was developed by Anscombe (1973). As in Fig. 14.12, he came up with four data sets consisting of 11 data points each. Although their graphs are very different, all have the same best-fit equation, $y = 3 + 0.5x$, and the same coefficient of determination, $r^2 = 0.67$! This example dramatically illustrates why developing plots is so valuable.

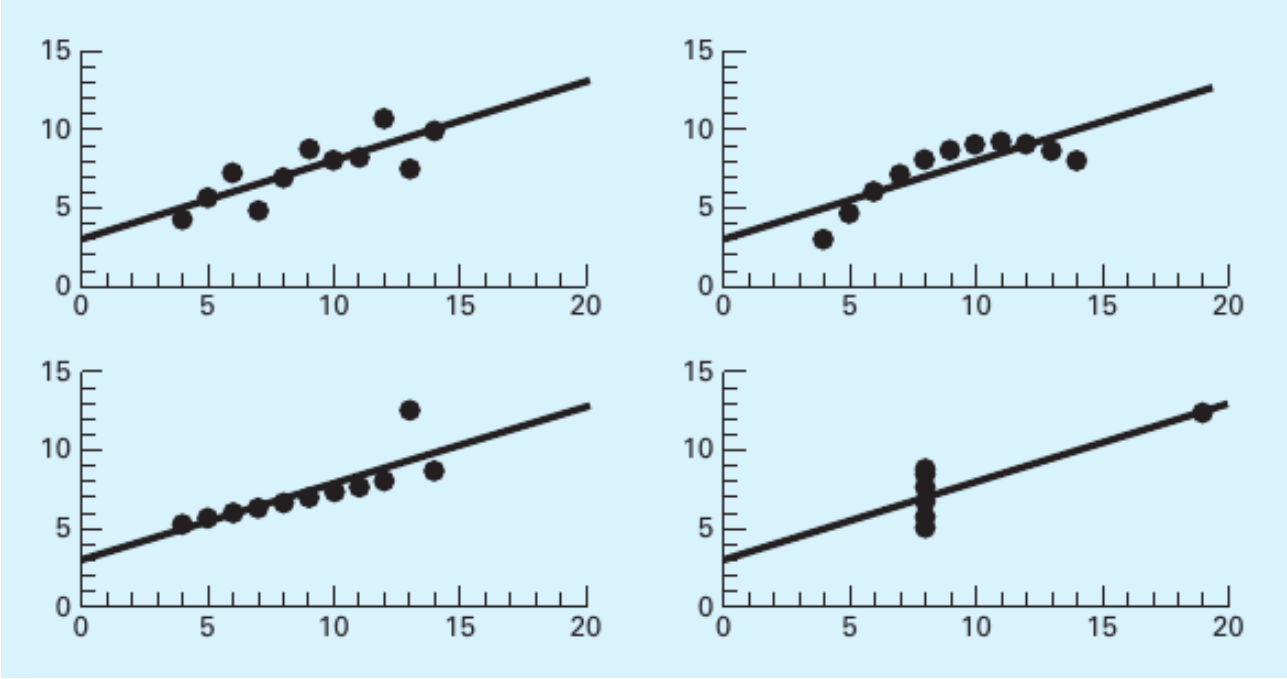


Figure 5.12: AnscombeâĂŹs four data sets along with the best-fit line, $y = 3 + 0.5x$.

5.3. LINEARIZATION OF NONLINEAR RELATIONSHIPS

Linear regression provides a powerful technique for fitting a best line to data. However, it is predicated on the fact that the relationship between the dependent and independent variables is linear. This is not always the case, and the first step in any regression analysis should be to plot and visually inspect the data to ascertain whether a linear model applies. In some cases, techniques such as polynomial regression, which is described in Chap. 15, are appropriate. For others, transformations can be used to express the data in a form that is compatible with linear regression.

One example is the *exponential model*:

$$/tag14.22y = a_1 e^{\beta_1 x} \tag{5.1}$$

where α and β_1 are constants. This model is used in many fields of engineering and science to characterize quantities that increase (positive β_1) or decrease (negative β_1) at a rate that is directly proportional to their own magnitude. For example, population growth or radioactive decay can exhibit such behavior. As depicted in Fig. 14.13a, the equation represents a nonlinear relationship (for $\beta_1 \neq 0$) between y and x.

Another example of a nonlinear model is the simple *power equation:*

$$y = a_2 x^{\beta_2} \tag{14.23}$$

where α_2 and β_2 are constant coefficients. This model has wide applicability in all fields of engineering and science. It is very frequently used to fit experimental data when the underlying model is not known. As depicted in Fig. 14.13b, the equation (for $\beta_2 \neq 0$) is nonlinear.

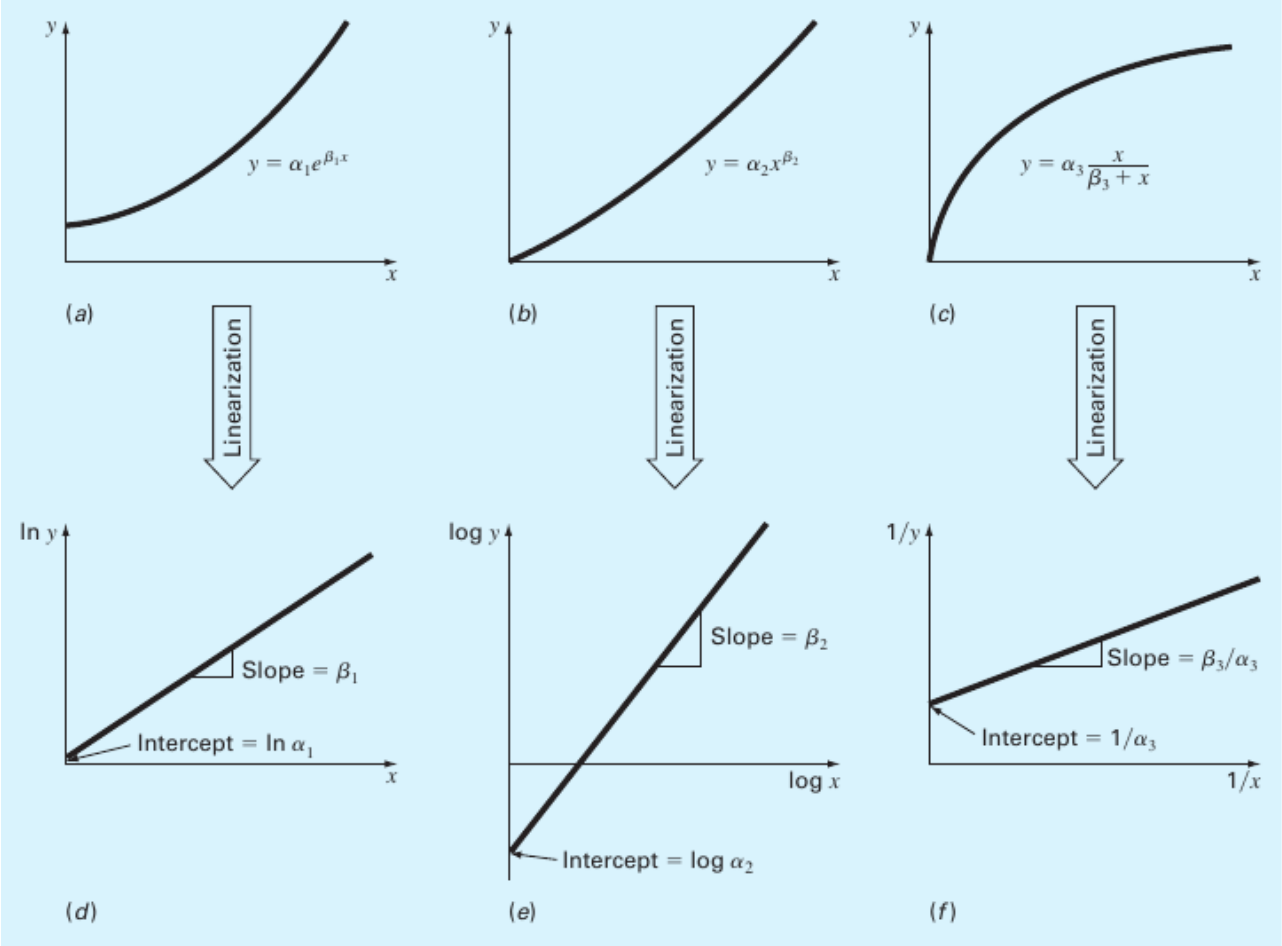


Figure 5.13: (a) The exponential equation, (b) the power equation, and (c) the saturation-growth-rate equation. Parts (d), (e), and (f) are linearized versions of these equations that result from simple transformations.

A third example of a nonlinear model is the *saturation-growth-rate equation:*

$$y = \alpha_3 \frac{x}{\beta_3 + x} \tag{14.24}$$

where α_3 and β_3 are constant coefficients. This model, which is particularly well-suited for characterizing population growth rate under limiting conditions, also represents a nonlinear relationship between y and x (Fig. 14.13c) that levels off, or âĂIJsaturates,âĂİ as x increases. It has many applications, particularly in biologically related areas of both engineering and science.

Nonlinear regression techniques are available to fit these equations to experimental data directly. However, a simpler alternative is to use mathematical manipulations to trans form the equations into a linear form. Then linear regression can be employed to fit the equations to data.

For example, Eq. (14.22) can be linearized by taking its natural logarithm to yield

$$\ln y = \ln \alpha_1 + \beta_1 x \tag{14.25}$$

Thus, a plot of $\ln y$ versus x will yield a straight line with a slope of β_1 and an intercept of $\ln \alpha_1$ (Fig. 14.13d).

Equation (14.23) is linearized by taking its base-10 logarithm to give

$$\log y = \log \alpha_2 + \beta_2 \log x \tag{14.26}$$

Thus, a plot of log y versus log x will yield a straight line with a slope of β_2 and an intercept of $\log \alpha_2$ (Fig. 14.13e). Note that any base logarithm can be used to linearize this model. However, as done here, the base-10 logarithm is most commonly employed.

Equation (14.24) is linearized by inverting it to give

$$\frac{1}{y} = \frac{1}{\alpha_3} + \frac{\beta_3}{\alpha_3}\frac{1}{x} \tag{14.27}$$

Thus, a plot of $1/y$ versus $1/x$ will be linear, with a slope of β_3/α_3 and an intercept of $1/\alpha_3$ (Fig. 14.13f).

In their transformed forms, these models can be fit with linear regression to evaluate the constant coefficients. They can then be transformed back to their original state and used for predictive purposes. The following illustrates this procedure for the power model.

The fits in Example 14.6 (Fig. 14.14) should be compared with the one obtained previously in Example 14.4 (Fig. 14.8) using linear regression on the untransformed data. Although both results would appear to be acceptable, the transformed result has the advantage that it does not yield negative force predictions at low velocities. Further, it is known from the discipline of fluid mechanics that the drag force on an object moving through a fluid is often well described by a model with velocity squared. Thus, knowledge from the field you are studying often has a large bearing on the choice of the appropriate model equation you use for curve fitting.

5.3.1. General Comments on Linear Regression

Before proceeding to curvilinear and multiple linear regression, we must emphasize the introductory nature of the foregoing material on linear regression. We have focused on the simple derivation and practical use of equations to fit data. You should be cognizant of the fact that there are theoretical aspects of regression that are of practical importance but are beyond the scope of this book. For example, some statistical assumptions that are inherent in the linear least-squares procedures are

1. Each x has a fixed value; it is not random and is known without error.
2. The y values are independent random variables and all have the same variance.
3. The y values for a given x must be normally distributed.

Such assumptions are relevant to the proper derivation and use of regression. For example, the first assumption means that (1) the x values must be error-free and (2) the regression of y versus x is not the same as x versus y. You are urged to consult other references such as Draper and Smith (1981) to appreciate aspects and nuances of regression that are beyond the scope of this book.

5.4. COMPUTER APPLICATIONS

Linear regression is so commonplace that it can be implemented on most pocket calculators. In this section, we will show how a simple M-file can be developed to determine the slope and intercept as well as to create a plot of the data and the best-fit line. We will also show how linear regression can be implemented with the built-in *polyfit* function.

5.4.1. MATLAB M-file: *linregr*

An algorithm for linear regression can be easily developed (Fig. 14.15). The required summations are readily computed with MATLABâĂŹs *sum* function. These are then used to compute the slope and the intercept with Eqs. (14.15) and (14.16). The routine displays the intercept and slope, the coefficient of determination, and a plot of the best-fit line along with the measurements.

A simple example of the use of this M-file would be to fit the force-velocity data analyzed in Example 14.4:

5.4.2. MATLAB Functions: *polyfit* and *polyval*

MATLAB has a built-in function *polyfit* that fits a least-squares nth-order polynomial to data. It can be applied as in

```
>> p = polyfit(x, y, n)
```

where x and y are the vectors of the independent and the dependent variabless, respectively, and n = the order of the polynomial. The function returns a vector p containing the polynomial's coefficients. We should note that it represents the polynomial using decreasing powers of x as in the following representation:

$$f(x) = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1} \tag{5.2}$$

Because a straight line is a first-order polynomial, `polyfit(x,y,1)` will return the slope and the intercept of the best-fit straight line.

```
>> x = [10 20 30 40 50 60 70 80];
>> y = [25 70 380 550 610 1220 830 1450];
>> a = polyfit(x,y,1)
a =
19.4702 -234.2857
```

Thus, the slope is 19.4702 and the intercept is -234.2857.

Another function, `polyval`, can then be used to compute a value using the coefficients. It has the general format:

```
>> y = polyval(p, x)
```

where p = the polynomial coefficients, and y = the best-fit value at x. For example,

```
>> y = polyval(a,45)
y =
641.8750
```

Background. *Enzymes* act as catalysts to speed up the rate of chemical reactions in living cells. In most cases, they convert one chemical, the *substrate*,into another, the *product*. The *Michaelis-Menten* equation is commonly used to describe such reactions:

$$v = \frac{v_m[S]}{k_s + [S]} \tag{5.3}$$

where v = the initial reaction velocity, v_m = the maximum initial reaction velocity, $[S]$ = substrate concentration, and k_s = a half-saturation constant. As in Fig. 14.16, the equation describes a saturating relationship which levels off with increasing $[S]$. The graph also illustrates that the *half-saturation constant* corresponds to the substrate concentration at which the velocity is half the maximum.

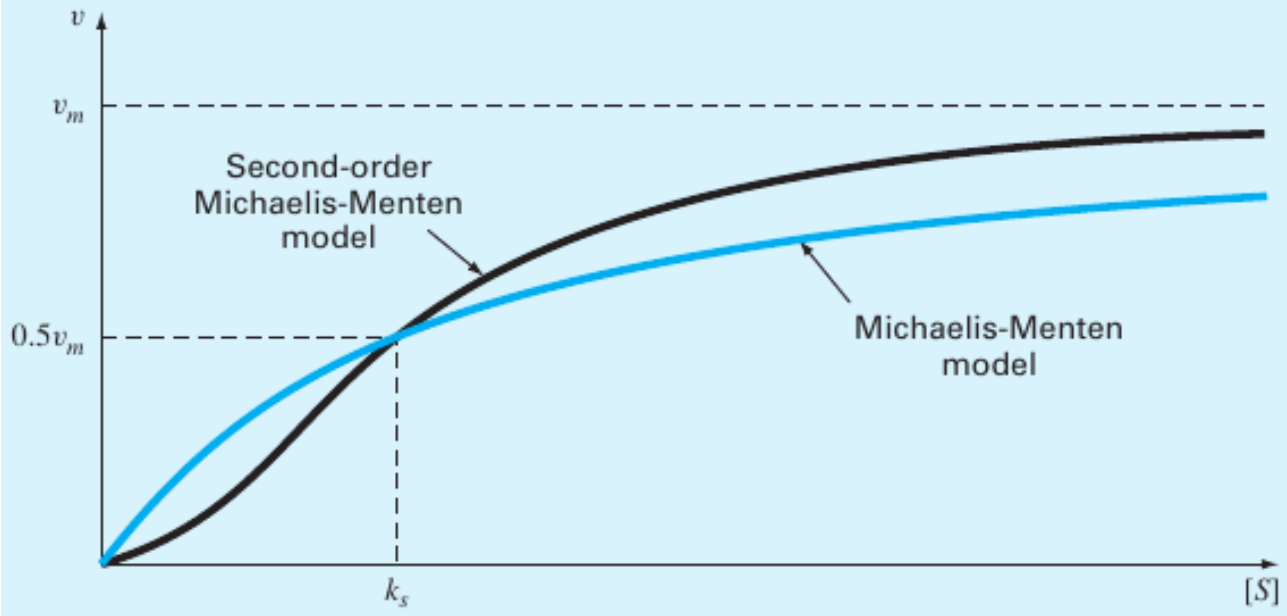


Figure 5.14: Two versions of the Michaelis-Menten model of enzyme kinetics.

Although the Michaelis-Menten model provides a nice starting point, it has been refined and extended to incorporate additional features of enzyme kinetics. One simple extension involves so-called *allosteric enzymes*, where the binding of a substrate molecule at one site leads to enhanced binding of subsequent molecules at other sites. For cases with two interacting bonding sites, the following second-order version often results in a better fit:

$$v = \frac{v_m[S]^2}{k_s^2 + [S]^2} \tag{5.4}$$

This model also describes a saturating curve but, as depicted in Fig. 14.16, the squared concentrations tend to make the shape more *sigmoid*, or S-shaped.

Suppose that you are provided with the following data:

```
[S] 1.3  1.8  3    4.5   6     8    9
v   0.07 0.13 0.22 0.275 0.335 0.35 0.36
```

Employ linear regression to fit these data with linearized versions of Eqs. (14.28) and (14.29). Aside from estimating the model parameters, assess the validity of the fits with both statistical measures and graphs.

Solution. Equation (14.28), which is in the format of the saturation-growth-rate model (Eq. 14.24), can be linearized by inverting it to give (recall Eq. 14.27)

$$\frac{1}{v} = \frac{1}{v_m} + \frac{k_s}{v_m}\frac{1}{[S]} \tag{5.5}$$

The `linregr` function from Fig. 14.15 can then be used to determine the least-squares fit:

```
>> S=[1.3 1.8 3 4.5 6 8 9];
>> v=[0.07 0.13 0.22 0.275 0.335 0.35 0.36];
>> [a,r2]=linregr(1./S,1./v)
a =
16.4022
 0.1902
r2 =
0.9344
```

The model coefficients can then be calculated as

```
>> vm=1/a(2)
vm =
5.2570
>> ks=vm*a(1)
ks =
86.2260
```

Thus, the best-fit model is

$$v = \frac{5.2570[S]}{86.2260 + [S]} \tag{5.6}$$

Although the high value of r^2 might lead you to believe that this result is acceptable, inspection of the coefficients might raise doubts. For example, the maximum velocity (5.2570) is much greater than the highest observed velocity (0.36). In addition, the half-saturation rate (86.2260) is much bigger than the maximum substrate concentration (9).

The problem is underscored when the fit is plotted along with the data. Figure 14.17a shows the transformed version. Although the straight line follows the upward trend, the data clearly appear to be curved. When the original equation is plotted along with the data in the untransformed version (Fig. 14.17b), the fit is obviously unacceptable. The data are clearly leveling off at about 0.36 or 0.37. If this is correct, an eyeball estimate would suggest that v_m should be about 0.36, and k_s should be in the range of 2 to 3.

Beyond the visual evidence, the poorness of the fit is also reflected by statistics like the coefficient of determination. For the untransformed case, a much less acceptable result of $r^2 = 0.6406$ is obtained.

The foregoing analysis can be repeated for the second-order model. Equation (14.28) can also be linearized by inverting it to give

$$\frac{1}{v} = \frac{1}{v_m} + \frac{k_s^2}{v_m}\frac{1}{[S]^2} \tag{5.7}$$

The `linregr` function from Fig. 14.15 can again be used to determine the least-squares fit:

```
>> [a,r2]=linregr(1./S.^2,1./v)
a =
19.3760
 2.4492
r2 =
0.9929
```

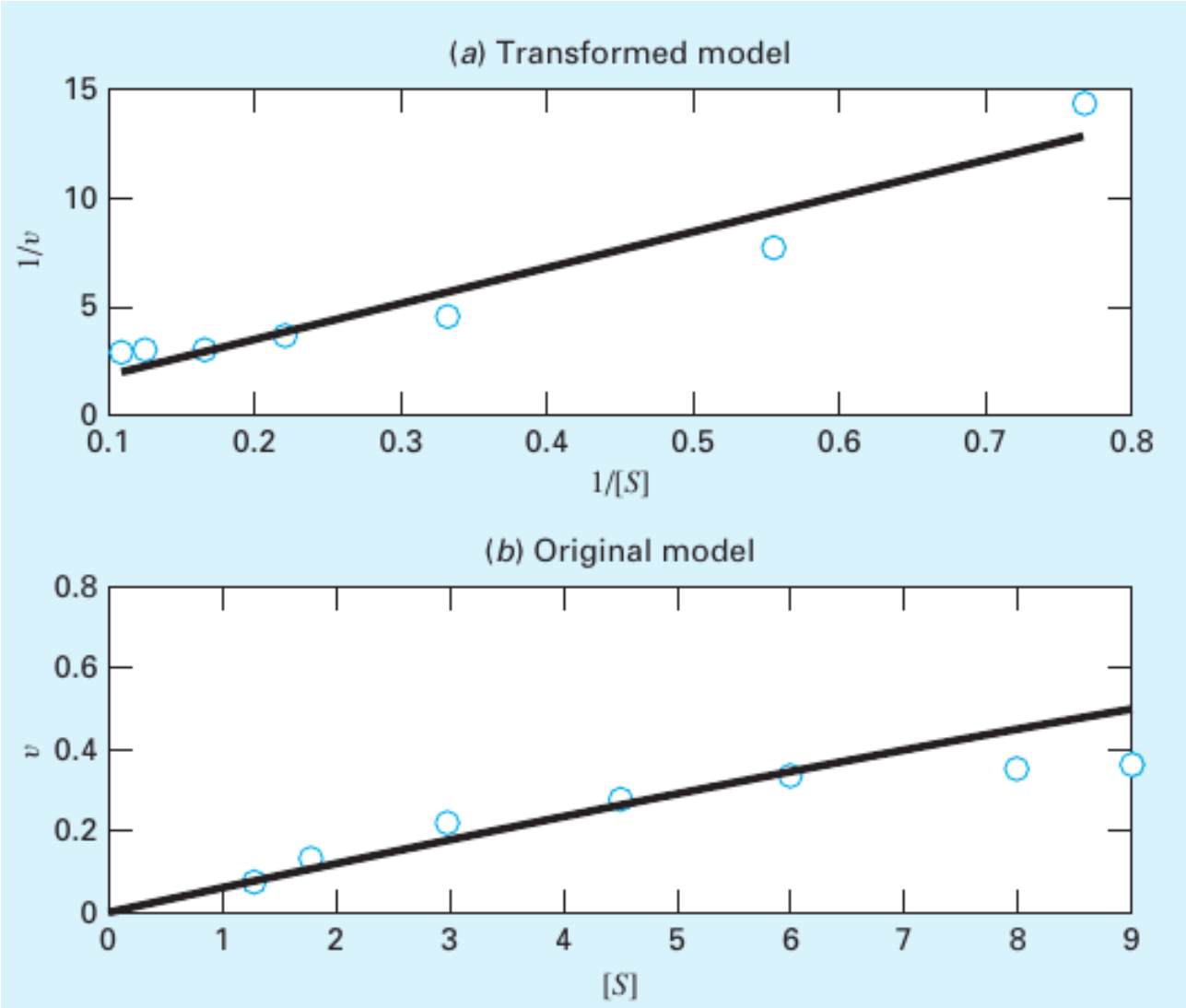


Figure 5.15: Plots of least-squares fit (line) of the Michaelis-Menten model along with data (points). The plot in (a) shows the transformed fit, and (b) shows how the fit looks when viewed in the untransformed, original form.

The model coefficients can then be calculated as

```
>> vm=1/a(2)
vm =
0.4083
>> ks=sqrt(vm*a(1))
ks =
2.8127
```

Substituting these values into Eq. (14.29) gives

$$v = \frac{0.4083[S]^2}{7.911 + [S]^2} \tag{5.8}$$

Although we know that a high r^2 does not guarantee of a good fit, the fact that it is very high (0.9929) is promising. In addition, the parameters values also seem consistent with the trends in the data; that is, the k_m is slightly greater than the highest observed velocity and the half-saturation rate is lower than the maximum substrate concentration (9).

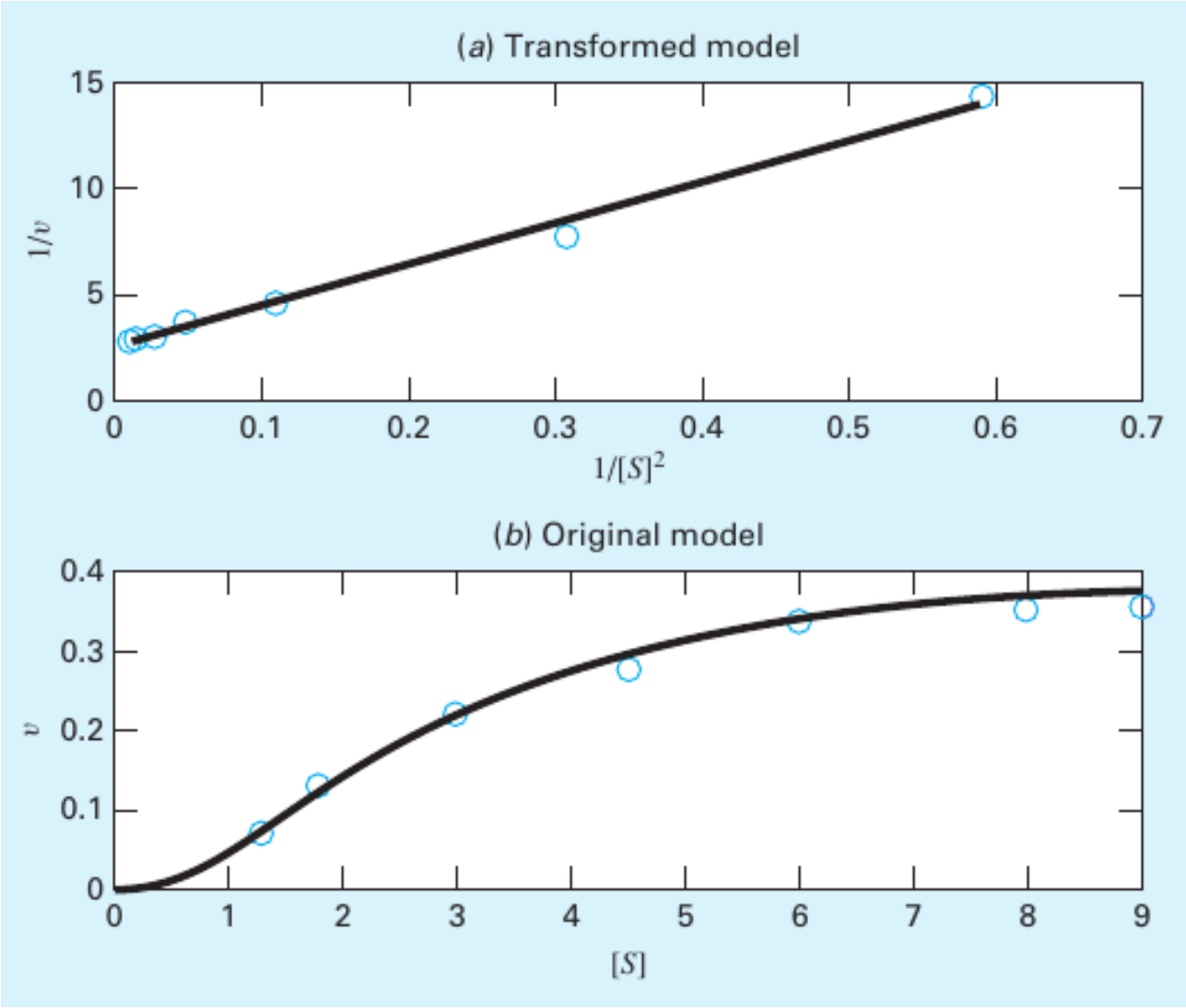


Figure 5.16: Plots of least-squares fit (line) of the second-order Michaelis-Menten model along with data (points). The plot in (a) shows the transformed fit, and (b) shows the untransformed, original form.

The adequacy of the fit can be assessed graphically. As in Fig. 14.18a, the transformed results appear linear. When the original equation is plotted along with the data in the untransformed version (Fig. 14.18b), the fit nicely follows the trend in the measurements. Beyond the graphs, the goodness of the fit is also reflected by the fact that the coefficient of determination for the untransformed case can be computed as $r^2 = 0.9896$.

Based on our analysis, we can conclude that the second-order model provides a good fit of this data set. This might suggest that we are dealing with an allosteric enzyme.

Beyond this specific result, there are a few other general conclusions that can be drawn from this case study. First, we should never solely rely on statistics such as r^2 as the sole basis of assessing goodness of fit. Second, regression equations should always be assessed graphically. And for cases where transformations are employed, a graph of the untransformed model and data should always be inspected.

Finally, although transformations may yield a decent fit of the transformed data, this does not always translate into an acceptable fit in the original format. The reason that this might occur is that minimizing squared residuals of transformed data is not the same as for the untransformed data. Linear regression assumes that the scatter of points around the best-fit line follows a Gaussian distribution, and that the standard deviation is the same at every value of the dependent variable. These assumptions are rarely true after transforming data.

As a consequence of the last conclusion, some analysts suggest that rather than using linear transformations, nonlinear regression should be employed to fit curvilinear data. In this approach, a best-fit curve is developed that directly minimizes the untransformed residuals. We will describe how this is done in Chap. 15.

Chapter 6

General Linear Least-Squares and Nonlinear Regression

CHAPTER OBJECTIVES This chapter takes the concept of fitting a straight line and extends it to (a) fitting a polynomial and (b) fitting a variable that is a linear function of two or more independent variables. We will then show how such applications can be generalized and applied to a broader group of problems. Finally, we will illustrate how optimization techniques can be used to implement nonlinear regression. Specific objectives and topics covered are

- âĂć Knowing how to implement polynomial regression.
- âĂć Knowing how to implement multiple linear regression.
- âĂć Understanding the formulation of the general linear least-squares model.
- âĂć Understanding how the general linear least-squares model can be solved with MATLAB using either the normal equations or left division.
- âĂć Understanding how to implement nonlinear regression with optimization techniques.

6.1. POLYNOMIAL REGRESSION

In Chap.14, a procedure was developed to derive the equation of a straight line using the least-squares criterion. Some data, although exhibiting a marked pattern such as seen in Fig. 15.1, are poorly represented by a straight line. For these cases, a curve would be better suited to fit the data. As discussed in Chap. 14, one method to accomplish this objective is to use transformations. Another alternative is to fit polynomials to the data using *polynomial regression*.

The least-squares procedure can be readily extended to fit the data to a higher-orderpolynomial. For example, suppose that we fit a second-order polynomial or quadratic:

$$y = a_0 + a_1 x + a_2 x^2 + e \tag{15.1}$$

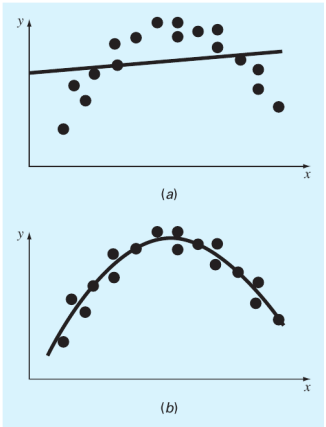


Figure 6.1: a) Data that are ill-suited for linear least-squares regression. (b) Indication that a parabola is preferable.

For this case the sum of the squares of the residuals is

$$S_r = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2 \tag{15.2}$$

To generate the least-squares fit, we take the derivative of Eq. (15.2) with respect to each of the unknown coefficients of the polynomial, as in

$$\frac{\partial S_r}{\partial a_0} = -2\sum (y_i - a_0 - a_1 x_i - a_2 x_i^2) \tag{6.1}$$

$$\frac{\partial S_r}{\partial a_1} = -2\sum x_i (y_i - a_0 - a_1 x_i - a_2 x_i^2) \tag{6.2}$$

$$\frac{\partial S_r}{\partial a_2} = -2\sum x_i^2 (y_i - a_0 - a_1 x_i - a_2 x_i^2) \tag{6.3}$$

These equations can be set equal to zero and rearranged to develop the following set of normal equations:

$$(n)a_0 + (\sum x_i)a_1 + (\sum x_i^2)a_2 = \sum y_i \tag{6.4}$$

$$(\sum x_i)a_0 + (\sum x_i^2)a_1 + (\sum x_i^3)a_2 = \sum x_i y_i \tag{6.5}$$

$$(\sum x_i^2)a_0 + (\sum x_i^3)a_1 + (\sum x_i^4)a_2 = \sum x_i^2 y_i \tag{6.6}$$

where all summations are from $i = 1$ through n. Note that the preceding three equations are linear and have three unknowns: a_0 , a_1, and a_2. The coefficients of the unknowns can be calculated directly from the observed data.

For this case, we see that the problem of determining a least-squares second-order polynomial is equivalent to solving a system of three simultaneous linear equations. The two-dimensional case can be easily extended to an mth-order polynomial as in

$$y = a_0 + a_1 x + a_2 x^2 + ... + a_m x^m + e \tag{6.7}$$

The foregoing analysis can be easily extended to this more general case. Thus, we can recognize that determining the coefficients of an mth-order polynomial is equivalent to solving a system of $m+1$ simultaneous linear equations. For this case, the standard error is formulated as

$$s_{y/x} = \sqrt{\frac{S_r}{n-(m+1)}} \tag{15.3}$$

This quantity is divided by $n-(m+1)$ because $(m+1)$ data-derived coefficients - $a_0, a_1, ..., a_m$ - were used to compute S_r; thus, we have lost $m+1$ degrees of freedom. In addition to the standard error, a coefficient of determination can also be computed for polynomial regression with Eq. (14.20).

6.2. MULTIPLE LINEAR REGRESSION

Another useful extension of linear regression is the case where y is a linear function of two or more independent variables. For example, y might be a linear function of x_1 and x_2, as in

$$y = a_0 + a_1 x_1 + a_2 x_2 + e \tag{6.8}$$

Such an equation is particularly useful when fitting experimental data where the variable being studied is often a function of two other variables. For this two-dimensional case, the regression "line" becomes a "plane" (Fig. 15.3).

As with the previous cases, the "best" values of the coefficients are determined by formulating the sum of the squares of the residuals:

$$S_r = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_{1,i} - a_2 x_{2,i})^2 \tag{15.4}$$

and differentiating with respect to each of the unknown coefficients:

$$\frac{\partial S_r}{\partial a_0} = -2 \sum (y_i - a_0 - a_1 x_{1,i} - a_2 x_{2,i}^2) \tag{6.9}$$

$$\frac{\partial S_r}{\partial a_1} = -2 \sum x_{1,i} (y_i - a_0 - a_1 x_{1,i} - a_2 x_{2,i}^2) \tag{6.10}$$

$$\frac{\partial S_r}{\partial a_2} = -2 \sum x_{2,i}^2 (y_i - a_0 - a_1 x_{1,i} - a_2 x_{2,i}^2) \tag{6.11}$$

The coefficients yielding the minimum sum of the squares of the residuals are obtained by setting the partial derivatives equal to zero and expressing the result in matrix form as

... (6.12)

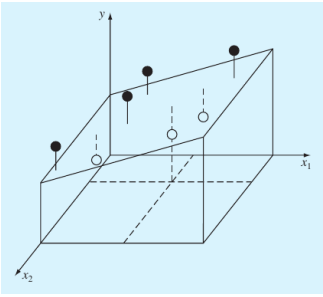


Figure 6.2: Graphical depiction of multiple linear regression where y is a linear function of x_1 and x_2.