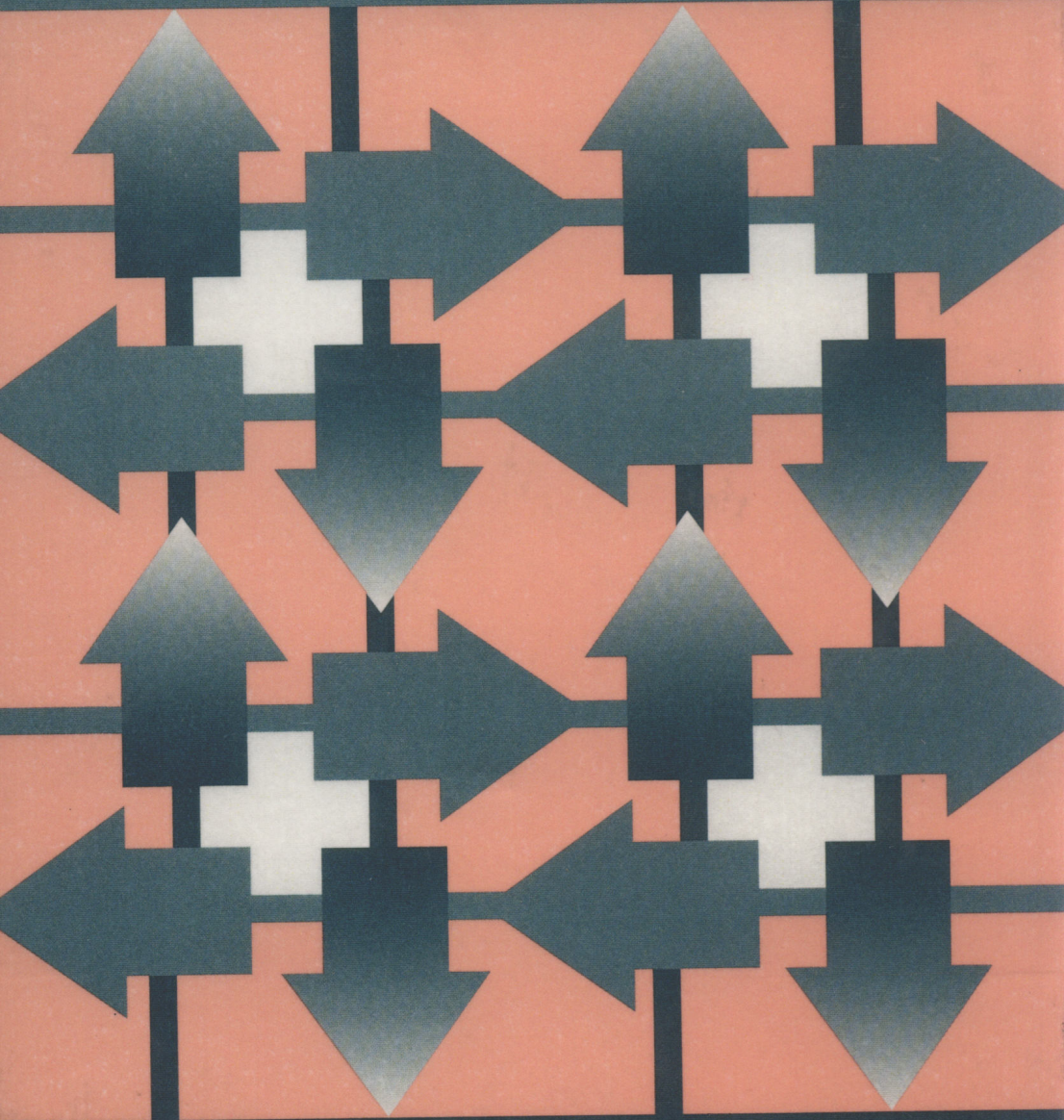


SCIENTIFIC COMPUTING

AN INTRODUCTION WITH

PARALLEL COMPUTING



Gene Golub / James M. Ortega

Scientific Computing
An Introduction with
Parallel Computing

Gene Golub

*Department of Computer Science
Stanford University
Stanford, California*

James M. Ortega

*School of Engineering & Applied Science
University of Virginia
Charlottesville, Virginia*

江苏工业学院图书馆
藏书章




ACADEMIC PRESS

Harcourt Brace & Company, Publishers

Boston San Diego New York

London Sydney Tokyo Toronto

This book is printed on acid-free paper. 

Copyright © 1993 by ACADEMIC PRESS

All Rights Reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission in writing from the publisher.

Requests for permission to make copies of any part of the work should be mailed to:
Permissions Department, Harcourt, Inc., 6277 Sea Harbor Drive,
Orlando, Florida 32887-6777

Academic Press

A Harcourt Science and Technology Company
525 B Street, Suite 1900, San Diego, California 92101-4495, USA
<http://www.academicpress.com>

Academic Press

Harcourt Place, 32 Jamestown Road, London NW1 7BY, UK
<http://www.academicpress.com>

International Standard Book Number: 0-12-289253-4

PRINTED IN THE UNITED STATES OF AMERICA

01 02 03 04 05 06 EB 9 8 7 6 5 4 3 2

*Dedicated to the memories
of our mothers*

Preface	ix
Chapter 1. The World of Scientific Computing	1
1.1 What Is Scientific Computing?	3
1.2 Mathematical Modeling	9
1.3 The Process of Numerical Solution	5
1.4 The Computational Environment	17
Chapter 2. Linear Algebra	25
2.1 Matrices and Vectors	25
2.2 Eigenvalues and Canonical Forms	38
2.3 Norms	70
Chapter 3. Parallel and Vector Computing	39
3.1 Parallel and Vector Computers	45
3.2 Basic Concepts of Parallel Computing	62
3.3 Matrix Multiplication	71
Chapter 4. Polynomial Approximation	91
4.1 Taylor Series, Interpolation, and Least Squares	91
4.2 Least Squares Approximation	105
4.3 Application to Root-Finding	117

Preface

This book is a modification of "Scientific Computing and Differential Equations: An Introduction to Numerical Methods." The intent of the present book is to introduce the basic ideas of vector and parallel computing as part of the introduction of basic numerical methods. We believe this will be useful for computer science students as well as those in science and engineering. It is clear that supercomputing will make an increasing impact on scientific and engineering endeavors in the years to come and supercomputing is necessarily based on vector and parallel computers. It would be useful for students to have access to such machines during this course, although that is not necessary. Much of the material on parallel and vector computing can still be studied profitably.

The book is meant to be used at the advanced undergraduate or beginning graduate level and it is assumed that such students will have a background in calculus, including some differential equations, and also some linear algebra. Linear algebra is the most important mathematical tool in scientific computing, both for the formulation of problems as well as for analysis of numerical methods for their solution. Even most students who have had a full undergraduate course in linear algebra will not have been exposed to some of the necessary material, such as norms and some aspects of eigenvalues, eigenvectors and canonical forms. Chapter 2 contains a review of such material. It is suggested that it be used primarily as a reference as the corresponding material arises later in the book.

Chapter 1 is an overview of scientific computing, especially the topics of mathematical modeling, the general process of numerical solution of problems, and the computational environment in which these solutions are obtained. Chapter 3 gives an introduction to some of the basic ideas in parallel and vector computing including a review of the architecture of such computers as well as some fundamental concepts. Some of these ideas and techniques are then illustrated by the simple but important problem of matrix-vector multiplication, a topic that arises repeatedly in the remainder of the book. Chapter 4 treats some elementary topics that are covered in any first course on numerical methods: Taylor series approximation, interpolation and splines,

least squares approximation, and finding solutions of nonlinear equations.

A large fraction of the book is devoted to the solution of linear systems of equations, since linear systems are at the center of many problems in scientific computing. Chapter 5 shows how linear systems arise by discretizing differential equations. An important aspect of these systems is the non-zero structure of the coefficient matrix. Thus, we may obtain tridiagonal matrices, banded matrices or very large sparse matrices, depending on the type of differential equation and the method of discretization.

Chapter 6 begins the solution of linear systems, concentrating on those systems for which direct methods such as Gaussian elimination are appropriate. There are discussions of rounding error and the effect of ill-conditioning as well as an introduction to other direct methods such as QR factorization. Chapter 7 shows how the basic direct methods must be reorganized to be efficient on parallel and vector computers.

For the very large sparse linear systems that arise from partial differential equations, direct methods are not always appropriate and Chapter 8 begins the study of iterative methods. The classical Jacobi, Gauss-Seidel and SOR methods, and their parallel and vector implementations, are treated leading up to their use in the multigrid method. Then, in Chapter 9, we consider conjugate gradient methods, for both symmetric and nonsymmetric systems. Included in this chapter are discussions of preconditioning, which utilizes some of the methods of Chapter 8, and parallel and vector implementations. This chapter ends with a nonlinear partial differential equation.

Many important topics have not been included; for example, computation of eigenvalues and eigenvectors and solution of linear or nonlinear programming problems. However, such areas also rely heavily on techniques for the solution of linear equations.

We believe that this book can be used successfully at different levels. For an introductory one semester course at the undergraduate level, concentration would be on Chapters 3, 4, 6 and parts of 5 and 7. For a first course at the graduate level for those who have had an undergraduate numerical methods course, much of Chapters 4, 5, and 6 can be covered rapidly in review and emphasis placed on the more advanced topics of Chapters 7, 8, and 9. Or, with some supplementary material from the instructor, the book could form the basis for a full year course.

We owe thanks to many colleagues and students for their comments on our previous book and on the draft of the current one. We are also indebted to Ms. Brenda Lynch for her expert LaTeXing of the manuscript.

Stanford, California
Charlottesville, Virginia

Table of Contents

Preface	ix
Chapter 1. The World of Scientific Computing	1
1.1 What Is Scientific Computing?	1
1.2 Mathematical Modeling	3
1.3 The Process of Numerical Solution	6
1.4 The Computational Environment	11
Chapter 2. Linear Algebra	15
2.1 Matrices and Vectors	15
2.2 Eigenvalues and Canonical Forms	28
2.3 Norms	39
Chapter 3 Parallel and Vector Computing	49
3.1 Parallel and Vector Computers	49
3.2 Basic Concepts of Parallel Computing	62
3.3 Matrix Multiplication	71
Chapter 4. Polynomial Approximation	91
4.1 Taylor Series, Interpolation and Splines	91
4.2 Least Squares Approximation	106
4.3 Application to Root-Finding	117

Chapter 5 Continuous Problems Solved Discretely	137
5.1 Numerical Integration	137
5.2 Initial Value Problems	150
5.3 Boundary Value Problems	173
5.4 Space and Time	191
5.5 The Curse of Dimensionality	202
Chapter 6. Direct Solution of Linear Equations	215
6.1 Gaussian Elimination	215
6.2 Errors in Gaussian Elimination	236
6.3 Other Factorizations	257
Chapter 7. Parallel Direct Methods	275
7.1 Basic Methods	275
7.2 Other Organizations of Factorization	294
7.3 Banded and Tridiagonal Systems	302
Chapter 8. Iterative Methods	321
8.1 Relaxation-Type Methods	321
8.2 Parallel and Vector Implementations	333
8.3 The Multigrid Method	353
Chapter 9. Conjugate Gradient-Type Methods	371
9.1 The Conjugate Gradient Method	371
9.2 Preconditioning	381
9.3 Nonsymmetric and Nonlinear Problems	397
Bibliography	413
Author Index	429
Subject Index	435

Chapter 1

The World of Scientific Computing

1.1 What Is Scientific Computing?

The many thousands of computers now installed in this country and abroad are used for a bewildering – and increasing – variety of tasks: accounting and inventory control for industry and government, airline and other reservation systems, limited translation of natural languages such as Russian to English, monitoring of process control, and on and on. One of the earliest – and still one of the largest – uses of computers was to solve problems in science and engineering and, more specifically, to obtain solutions of mathematical models that represent some physical situation. The techniques used to obtain such solutions are part of the general area called *scientific computing*, and the use of these techniques to elicit insight into scientific or engineering problems is called *computational science* (or *computational engineering*).

There is now hardly an area of science or engineering that does not use computers for modeling. Trajectories for earth satellites and for planetary missions are routinely computed. Engineers use computers to simulate the flow of air about an aircraft or other aerospace vehicle as it passes through the atmosphere, and to verify the structural integrity of aircraft. Such studies are of crucial importance to the aerospace industry in the design of safe and economical aircraft and spacecraft. Modeling new designs on a computer can save many millions of dollars compared to building a series of prototypes. Similar considerations apply to the design of automobiles and many other products, including new computers.

Civil engineers study the structural characteristics of large buildings, dams, and highways. Meteorologists use large amounts of computer time to predict

tomorrow's weather as well as to make much longer range predictions, including the possible change of the earth's climate. Astronomers and astrophysicists have modeled the evolution of stars, and much of our basic knowledge about such phenomena as red giants and pulsating stars has come from such calculations coupled with observations. Ecologists and biologists are increasingly using the computer in such diverse areas as population dynamics (including the study of natural predator and prey relationships), the flow of blood in the human body, and the dispersion of pollutants in the oceans and atmosphere.

The mathematical models of all of these problems are systems of differential equations, either ordinary or partial. Differential equations come in all "sizes and shapes," and even with the largest computers we are nowhere near being able to solve many of the problems posed by scientists and engineers. But there is more to scientific computing, and the scope of the field is changing rapidly. There are many other mathematical models, each with its own challenges. In operations research and economics, large linear or nonlinear optimization problems need to be solved. Data reduction – the condensation of a large number of measurements into usable statistics – has always been an important, if somewhat mundane, part of scientific computing. But now we have tools (such as earth satellites) that have increased our ability to make measurements faster than our ability to assimilate them; fresh insights are needed into ways to preserve and use this irreplaceable information. In more developed areas of engineering, what formerly were difficult problems to solve even once on a computer are today's routine problems that are being solved over and over with changes in design parameters. This has given rise to an increasing number of computer-aided design systems. Similar considerations apply in a variety of other areas.

Although this discussion begins to delimit the area that we call scientific computing, it is difficult to define it exactly, especially the boundaries and overlaps with other areas. We will accept as our working definition that *scientific computing is the collection of tools, techniques, and theories required to solve on a computer mathematical models of problems in science and engineering.*

A majority of these tools, techniques, and theories originally developed in mathematics, many of them having their genesis long before the advent of electronic computers. This set of mathematical theories and techniques is called numerical analysis (or numerical mathematics) and constitutes a major part of scientific computing. The development of the electronic computer, however, signaled a new era in the approach to the solution of scientific problems. Many of the numerical methods that had been developed for the purpose of hand calculation (including the use of desk calculators for the actual arithmetic) had to be revised and sometimes abandoned. Considerations that were irrelevant or unimportant for hand calculation now became of utmost importance for the efficient and correct use of a large computer system. Many of these considerations – programming languages, operating systems, management of large

quantities of data, correctness of programs – were subsumed under the discipline of computer science, on which scientific computing now depends heavily. But mathematics itself continues to play a major role in scientific computing: it provides the language of the mathematical models that are to be solved and information about the suitability of a model (Does it have a solution? Is the solution unique?), and it provides the theoretical foundation for the numerical methods and, increasingly, many of the tools from computer science.

In summary, then, scientific computing draws on mathematics and computer science to develop the best ways to use computer systems to solve problems from science and engineering. This relationship is depicted schematically in Figure 1.1.1. In the remainder of this chapter, we will go a little deeper into these various areas.

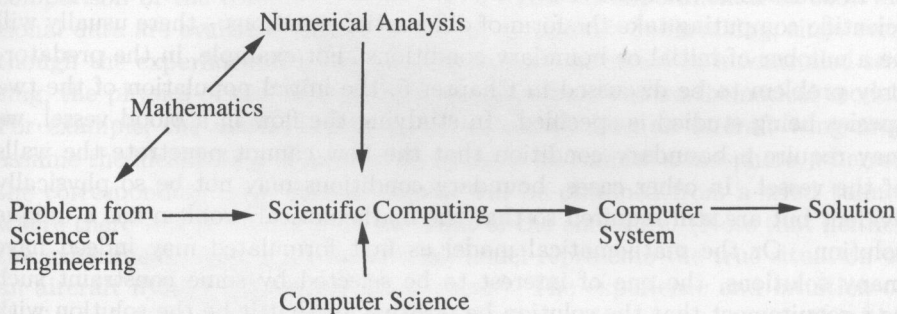


Figure 1.1.1: *Scientific Computing and Related Areas*

1.2 Mathematical Modeling

As was discussed in Section 1.1, we view scientific computing as the discipline that achieves a computer solution of mathematical models of problems from science and engineering. Hence, the first step in the overall solution process is the formulation of a suitable mathematical model of the problem at hand.

Modeling

The formulation of a mathematical model begins with a statement of the factors to be considered. In many physical problems, these factors concern the balance of forces and other conservation laws of physics. For example, in the formulation of a model of a trajectory problem the basic physical law is Newton's second law of motion, which requires that the forces acting on a body equal the rate of change of momentum of the body. This general law must then be specialized to the particular problem by enumerating and quantifying the

forces that will be of importance. For example, the gravitational attraction of Jupiter will exert a force on a rocket in Earth's atmosphere, but its effect will be so minute compared to the earth's gravitational force that it can usually be neglected. Other forces may also be small compared to the dominant ones but their effects not so easily dismissed, and the construction of the model will invariably be a compromise between retaining all factors that could likely have a bearing on the validity of the model and keeping the mathematical model sufficiently simple that it is solvable using the tools at hand. Classically, only very simple models of most phenomena were considered since the solutions had to be achieved by hand, either analytically or numerically. As the power of computers and numerical methods has developed, increasingly complicated models have become tractable.

In addition to the basic relations of the model – which in most situations in scientific computing take the form of differential equations – there usually will be a number of initial or boundary conditions. For example, in the predator-prey problem to be discussed in Chapter 5, the initial population of the two species being studied is specified. In studying the flow in a blood vessel, we may require a boundary condition that the flow cannot penetrate the walls of the vessel. In other cases, boundary conditions may not be so physically evident but are still required so that the mathematical problem has a unique solution. Or the mathematical model as first formulated may indeed have many solutions, the one of interest to be selected by some constraint such as a requirement that the solution be positive, or that it be the solution with minimum energy. In any case, it is usually assumed that the final mathematical model with all appropriate initial, boundary, and side conditions indeed has a unique solution. The next step, then, is to find this solution. For problems of current interest, such solutions rarely can be obtained in “closed form.” The solution must be approximated by some method, and the methods to be considered in this book are numerical methods suitable for a computer. In the next section we will consider the general steps to be taken to achieve a numerical solution, and the remainder of the book will be devoted to a detailed discussion of these steps for a number of different problems.

Validation

Once we are able to compute solutions of the model, the next step usually is called the *validation of the model*. By this we mean a verification that the solution we compute is sufficiently accurate to serve the purposes for which the model was constructed. There are two main sources of possible error. First, there invariably are errors in the numerical solution. The general nature of these errors will be discussed in the next section, and one of the major themes in the remainder of the book will be a better understanding of the source and control of these numerical errors. But there is also invariably an error in the model itself. As mentioned previously, this is a necessary aspect of modeling:

the modeler has attempted to take into account all the factors in the physical problem but then, in order to keep the model tractable, has neglected or approximated those factors that would seem to have a small effect on the solution. The question is whether neglecting these effects was justified. The first test of the validity of the model is whether the solution satisfies obvious physical and mathematical constraints. For example, if the problem is to compute a rocket trajectory where the expected maximum height is 100 kilometers and the computed solution shows heights of 200 kilometers, obviously some blunder has been committed. Or, it may be that we are solving a problem for which we know, mathematically, that the solution must be increasing but the computed solution is not increasing. Once such gross errors are eliminated – which is usually fairly easy – the next phase begins, which is, whenever possible, comparison of the computed results with whatever experimental or observational data are available. Many times this is a subtle undertaking, since even though the experimental results may have been obtained in a controlled setting, the physics of the experiment may differ from the mathematical model. For example, the mathematical model of airflow over an aircraft wing may assume the idealization of an aircraft flying in an infinite atmosphere, whereas the corresponding experimental results will be obtained from a wind tunnel where there will be effects from the walls of the enclosure. (Note that neither the experiment, nor the mathematical model represents the true situation of an aircraft flying in our finite atmosphere.) The experience and intuition of the investigator are required to make a human judgement as to whether the results from the mathematical model are corresponding sufficiently well with observational data.

At the outset of an investigation this is quite often not the case, and the model must be modified. This may mean that additional terms – which were thought negligible but may not be – are added to the model. Sometimes a complete revision of the model is required and the physical situation must be approached from an entirely different point of view. In any case, once the model is modified the cycle begins again: a new numerical solution, revalidation, additional modifications, and so on. This process is depicted schematically in Figure 1.2.1.

Once the model is deemed adequate from the validation and modification process, it is ready to be used for prediction. This, of course, was the whole purpose. We should now be able to answer the questions that gave rise to the modeling effort: How high will the rocket go? Will the wolves eat all the rabbits? Of course, we must always take the answers with a healthy skepticism. Our physical world is simply too complicated and our knowledge of it too meager for us to be able to predict the future perfectly. Nevertheless, we hope that our computer solutions will give increased insight into the problem being studied, be it a physical phenomenon or an engineering design.

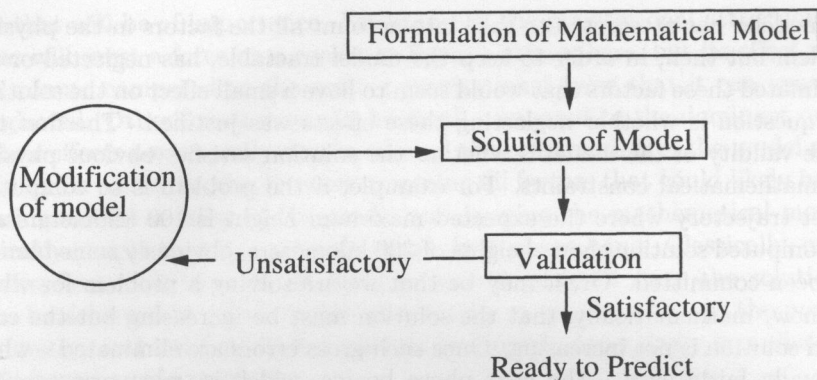


Figure 1.2.1: *The Mathematical Modeling and Solution Process*

1.3 The Process of Numerical Solution

We will discuss in this section the general considerations that arise in the computer solution of a mathematical model, and in the remainder of the book these matters will be discussed in more detail.

Once the mathematical model is given, our first thought typically is to try to obtain an explicit closed-form solution, but such a solution will usually only be possible for certain (perhaps drastic) simplifications of the problem. These simplified problems with known solutions may be of great utility in providing “check cases” for the more general problem.

After realizing that explicit solutions are not possible, we then turn to the task of developing a numerical method for the solution. Implicit in our thinking at the outset – and increasingly explicit as the development proceeds – will be the computing equipment as well as the software environment that is at our disposal. Our approach may be quite different for a microcomputer than for a very large computer. But certain general factors must be considered regardless of the computer to be used.

Rounding Errors

Perhaps the most important factor is that computers deal with a finite number of digits or characters. Because of this we cannot, in general, do arithmetic within the real number system as we do in pure mathematics. That is, the arithmetic done by a computer is restricted to finitely many digits, whereas the numerical representation of most real numbers requires infinitely many. For example, such fundamental constants as π and e require an infinite number of digits for a numerical representation and can *never* be entered exactly in a computer. Moreover, even if we could start with numbers that have

an exact numerical representation in the computer, the processes of arithmetic require that eventually we make certain errors. For example, the quotient of two numbers may require infinitely many digits for its numerical representation. Therefore, we resign ourselves at the outset to the fact that we cannot do arithmetic exactly on a computer. We shall make small errors, called *rounding errors*, on almost all arithmetic operations, and our task is to insure that these small errors do not accumulate to such an extent as to invalidate the computation.

Computers use the binary number system, and each machine allows a number of binary digits that can be carried in the usual arithmetic, called *single-precision* arithmetic, of the machine. On most scientific computers, this is the equivalent of between 7 and 14 decimal digits. *Higher-precision* arithmetic can also be carried out. On many machines *double-precision* arithmetic, which essentially doubles the number of digits that are carried, is part of the hardware; in this case, programs with double-precision arithmetic usually require only modest, if any, increases in execution time compared to single-precision. On the other hand, some machines implement double precision by software, which may require several times as much time as single precision. Precision higher than double is essentially always carried out by means of software and becomes increasingly inefficient as the precision increases. Higher-precision arithmetic is rarely used on practical problems, but it may be useful for generating "exact" solutions or other information for testing purposes.

Round-off errors can affect the final computed result in different ways. First, during a sequence of millions of operations, each subject to a small error, there is the danger that these small errors will accumulate so as to eliminate much of the accuracy of the computed result. If we round to the nearest digit, the individual errors will tend to cancel out, but the standard deviation of the accumulated error will tend to increase with the number of operations, leaving the possibility of a large final error. If chopping – that is, dropping the trailing digits rather than rounding – is used, there is a bias to errors in one direction, and the possibility of a large final error is increased.

In addition to the possible accumulation of errors over a large number of operations, there is the danger of *catastrophic cancellation*. Suppose that two numbers a and b are equal to within their last digit. Then the difference $c = a - b$ will have only one significant digit of accuracy *even though no round-off error will be made in the subtraction*. Future calculations with c may then limit the final result to one correct digit. Whenever possible, one tries to eliminate the possibility of catastrophic cancellation by rearranging the operations. Catastrophic cancellation is one way in which an algorithm can be *numerically unstable*, although in exact arithmetic it may be a correct algorithm. Indeed, it is possible for the results of a computation to be completely erroneous because of round-off error even though only a small number of arithmetic operations have been performed. Examples of this will be given later.

Detailed round-off error analyses have now been completed for a number of the simpler and more basic algorithms such as those that occur in the solution of linear systems of equations; some of these results will be described in more detail in Chapter 6. A particular type of analysis that has proved to be very powerful is *backward error analysis*. In this approach the round-off errors are shown to have the same effect as that caused by changes in the original problem data. When this analysis is possible, it can be stated that the error in the solution caused by round off is no worse than that caused by certain errors in the original model. The question of errors in the solution is then equivalent to the study of the sensitivity of the solution to perturbations in the model. If the solution is highly sensitive, the problem is said to be *ill-posed* or *ill-conditioned*, and numerical solutions are apt to be meaningless.

Discretization Error

Another way that the finiteness of computers manifests itself in causing errors in numerical computation is due to the need to replace "continuous" problems by "discrete" ones. Chapter 5 is devoted to showing how continuous problems such as differential equations are approximated by discrete problems. As a simple example, discussed further in Section 5.1, the integral of a continuous function requires knowledge of the integrand along the whole interval of integration, whereas a computer approximation to the integral can use values of the integrand at only finitely many points. Hence, even if the subsequent arithmetic were done exactly with no rounding errors, there would still be the error due to the discrete approximation to the integral. This type of error is usually called *discretization error* or *truncation error*, and it affects, except in trivial cases, all numerical solutions of differential equations and other "continuous" problems.

There is one more type of error, which is somewhat akin to discretization error. Many numerical methods are based on the idea of an *iterative process*. In such a process, a sequence of approximations to a solution is generated with the hope that the approximations will converge to the solution; in many cases mathematical proofs can be given that show convergence as the number of iterations tends to infinity. However, only finitely many such approximations can ever be generated on a computer, and, therefore, we must necessarily stop short of mathematical convergence. The error caused by such finite termination of an iterative process is sometimes called *convergence error*, although there is no generally accepted terminology here.

If we rule out trivial problems that are of no interest in scientific computing, we can summarize the situation with respect to computational errors as follows. Every calculation will be subject to rounding error. Whenever the mathematical model of the problem is a differential equation or other "continuous" problem, there also will be discretization error. And if an iterative method is used for the solution, there will be convergence error. These types of

errors and methods of analyzing and controlling them will be discussed more fully in concrete situations throughout the remainder of the book. But it is important to keep in mind that an acceptable error is very much dependent on the particular problem. Rarely is very high accuracy – say, 14 digits – needed in the final solution; indeed, for many problems arising in industry or other applications two or three digit accuracy is quite acceptable.

Efficiency

The other major consideration besides accuracy in the development of computer methods for the solution of mathematical models is *efficiency*. For most problems, such as solving a system of linear algebraic equations, there are many possible methods, some going back tens or even hundreds of years. Clearly, we would like to choose a method that minimizes the computing time yet retains suitable accuracy in the approximate solution. This turns out to be a difficult problem which involves a number of considerations. Although it is frequently possible to estimate the computing time of an algorithm by counting the required arithmetic operations, the amount of computation necessary to solve a problem to a given tolerance is still an open question except in a few cases. Even if one ignores the effects of round-off error, surprisingly little is known. In the past several years these questions have spawned the subject of *computational complexity*. However, even if such theoretical results were known, they would still give only approximations to the actual computing time, which depends on a number of factors involving the computer system. And these factors change as the result of new systems and architectures. Indeed, the design and analysis of numerical algorithms should provide incentives and directions for such changes.

We give a simple example of the way a very inefficient method can arise. Many elementary textbooks on matrix theory or linear algebra present Cramer's rule for solving systems of linear equations. This rule involves quotients of certain determinants, and the definition of a determinant is usually given as the sum of all possible products (some with minus signs) of elements of the matrix, one element from each row and each column. There are $n!$ such products for an $n \times n$ matrix. Now, if we proceed to carry out the computation of a determinant based on a straightforward implementation of this definition, it would require about $n!$ multiplications and additions. For n very small, say $n = 2$ or $n = 3$, this is a small amount of work. Suppose, however, that we have a 20×20 matrix, a very small size in current scientific computing. If we assume that each arithmetic operation requires 1 microsecond (10^{-6} second), then the time required for this calculation – even ignoring all overhead operations in the computer program – will exceed one million years! On the other hand, the Gaussian elimination method, which will be discussed in Chapter 6, will do the arithmetic operations for the solution of a 20×20 linear system in less than 0.005 second, again assuming 1 microsecond per operation. Although