NUMERICAL METHODS: THEORIES AND ALGORITHMS

BENCHAWAN WIWATANAPATAPHEE
Mahidol University • Thailand

YONG HONG WU
Curtin University of Technology • Australia

Physics Center Press Limited Partnership
Bangkok, Thailand

Published by
Physics Center Press Limited Partnership
Bangkok, Thailand

National Library of Thailand Cataloging in Publication Data

Benchawan Wiwatanapataphee, Yong Hong Wu

Numerical Methods: Theories and Algorithms. Nakhon Pathom : Physics center, 2008.
258 p.

1. Computer Algorithms. I. Title

005.1
ISBN : 978-974-434-897-5

Numerical Methods: Theories and Algorithms

Printed in Thailand by Physics Center Press Limited Partnership

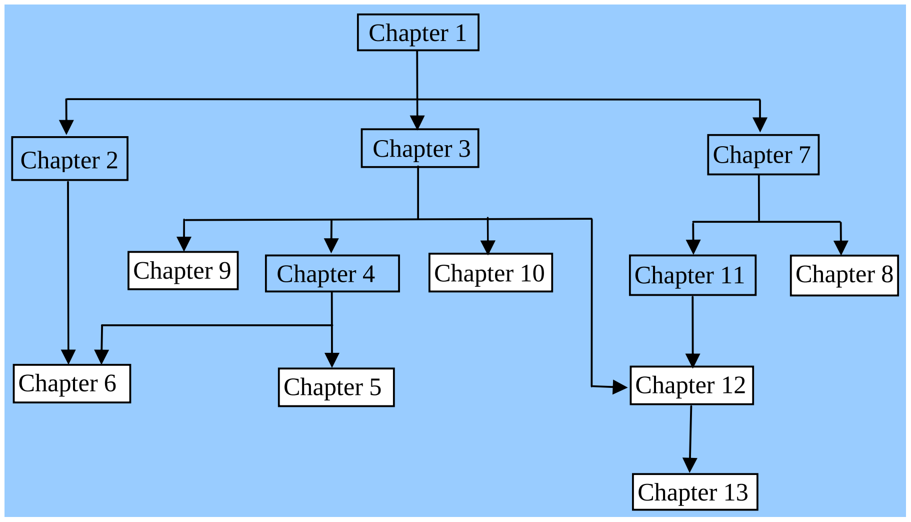
ISBN : 978-974-434-897-5

PREFACE

The study of real world problems often involves solving certain mathematical problems such as a set of linear equations or a set of ordinary differential equations. In most cases, analytical solutions cannot be obtained and thus, the only way to obtain any idea of the behaviour of a solution is to approximate the problem in such a way that numbers representing the solution can be produced. The sequence of numbers obtained is called a numerical solution and the process generating such a solution is called a numerical method.

This book introduces the fundamental numerical methods for solution of some mathematical problems often encountered in applied mathematical modelling, science and engineering studies. It focuses on the mathematical theories behind the numerical methods, the design of numerical algorithms for approximating the solution of the problems that cannot be solved exactly, the estimate of numerical errors, and convergence analysis. To cater for those who are interested in finding the solution without getting details on how it is obtained and how accurate the solution is, the book also shows, through many examples, how to use symbolic packages like Maple and MATLAB to solve most of the mathematical problems covered in each chapter.

The book can be used as a reference book for scientists and engineers who have interest in using mathematics to solve real world problems. It can also be used as a textbook for numerical analysis or scientific computing courses designed for students who have completed at least the first year of standard university calculus courses. The delivery of the materials covered in the book requires a minimum of 72 hours of lectures, which normally requires two standard semesters of studies. However, it is also possible for instructors to choose various topics from the book to construct a one semester course. The following diagram, showing chapter prerequisites, provides guidelines for the choice of chapters.



For example, one could choose to construct a one semester course consisting of chapters 1, 2, 3, 7, 8, 9 and 11.

Today, many programming languages, such as F95, C++, Maple and MATLAB, are available for implementation of numerical algorithms on computers. For large-scale numerical computation where efficiency is important, F95 is the best choice and C++ provides a second best choice. On the other hand, MATLAB and Maple both offer powerful graphic functions and are easy to use as these languages do not require high level of programming skills. For design of numerical computing programs using F95/C++/MATLAB, the reader is referred to our recently published book "Program Design using F95/C++/MATLAB"; while for numerical computing using Maple, the appendix in this book provides a quick reference.

The numerical algorithms covered in this book are written without referencing to any specific programming language, and so one could choose a proper programming language for implementation of numerical algorithms on computer based on the need and the purpose of study. For the programming exercises questions in each chapter, one could also choose to implement the associated algorithms using his/her chosen programming language instead of F95 as specified in some of the questions.

Yong Hong Wu & B. Wiwatanapataphee
October 2007

CONTENTS

CHAPTER 1

Preliminaries

In this chapter, we first review some basic theorems of calculus which are to be used in later chapters. Then we discuss the general procedure for solving real world problems mathematically and give general guidelines for algorithm design. Then, we introduce the concept of round off errors and analyse their generation and propagation in computation using floating point arithmetic. Finally, we introduce two kinds of error measures and error tests for checking convergence of numerical results.

1.1 Some Basic Theorems in Calculus

The Mean Value Theorem

If $f(x) \in C\,[a, b]$ and $f(x)$ is differentiable on (a, b) then there exists a number c in (a, b), as shown in Figure 1.1 below, such that

$$f'(c) = \frac{f(b) - f(a)}{b - a}$$

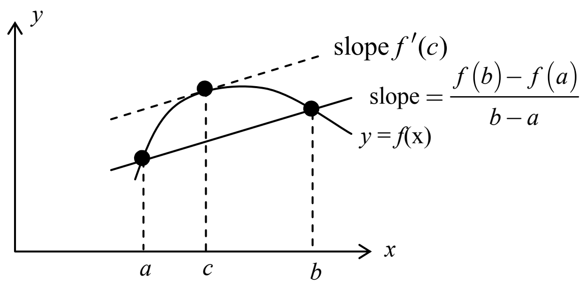


Figure 1.1 Sketch for the Mean value theorem

The Mean Value Theorem for Integrals

If $f(x) \in C[a, b]$, and $g(x)$ is integrable on $[a, b]$ and does not change sign on $[a, b]$, then there exists a number $c \in (a, b)$ such that

$$\int_a^b f(x)\,g(x)\,dx = f(c)\int_a^b g(x)dx\,.$$

The Intermediate Value Theorem

Suppose $f(x) \in C[a, b]$ and L be any number between $f(a)$ and $f(b)$ where $f(a) \neq f(b)$. Then there exists a number $c \in [a, b]$, ass shown in Figure 1.2, such that $f(c)=L$

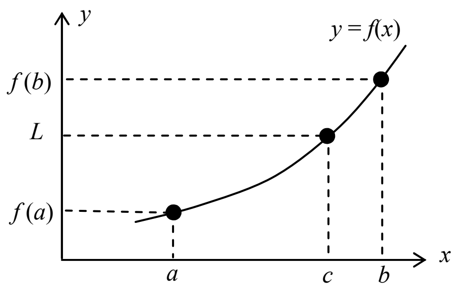


Figure 1.2 Sketch for the intermediate value theorem

The Extreme Value Theorem

If $f(x) \in C[a, b]$, then $f(x)$ attains an absolute maximum value $f(c_1)$ and an absolute minimum value $f(c_2)$ at some number c_1 and c_2 in $[a, b]$. If $f(x)$ is differentiable in (a, b), then c_1 and c_2 occur either at the end points of $[a, b]$ or at the critical points where f' is zero.

Taylor Theorem

Suppose that $f(x) \in C^n[a, b]$ and $f^{(n+1)}$ exists on $[a, b]$. Let $x_0 \in [a, b]$. Then for any $x \in [a, b]$, there exists a $\xi(x)$ between x_0 and x such that

$$f(x) = P_n(x) + R_n(x)$$

where $P_n(x)$ and $R_n(x)$ are called the nth Taylor polynomial for $f(x)$ about x_0 and the truncation error respectively, and are defined by

$$P_n(x) = f(x_0) + f'(x_0)(x - x_0) + \ldots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

$$= \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i ,$$

$$R_n(x) = \frac{f^{(n+1)}\xi(x)}{(n+1)!}(x - x_0)^{n+1}.$$

Taylor Theorem in 2-Variables

Suppose that $f(x,y)$ and all its partial derivatives of order less than or equal to $n+1$ are continuous on some region D and let $(x_0, y_0) \in D$. Then for any $(x,y) = (x_0 + h, y_0 + k) \in D$, there exist $\xi \in [x_0, x_0 + h]$ and $\eta \in [y_0, y_0 + k]$ such that

$$f(x,y) = P_n(x,y) + R_n(\xi,\eta)$$

where $P_n(x,y)$ and $R_n(\xi,\eta)$ are called the nth Taylor polynomial for $f(x,y)$ about (x_0, y_0) and the truncation error respectively, and are defined by

$$P_n(x,\ y) = \sum_{i=0}^{\infty} \frac{1}{i!}\left((x-x_0)\frac{\partial}{\partial x} + (y-y_0)\frac{\partial}{\partial y} \right)^i f(x_0,\ y_0) = \sum_{i=0}^{\infty} \frac{1}{i!}\left(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y} \right)^i f(x_0,\ y_0)$$

$$E_n(\xi,\eta) = \frac{1}{(n+1)!}\left(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y} \right)^{n+1} f^{(n+1)}(\xi,\ \eta)\,.$$

1.2 Procedure for Solving Mathematical Problems Numerically

The following figure summarizes the procedure for solving mathematical problems numerically

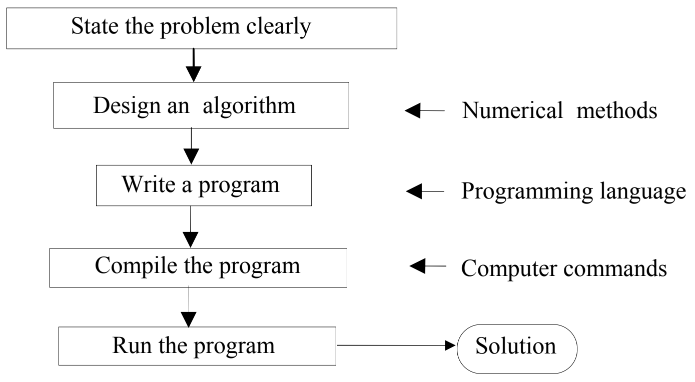


Figure 1.4 Procedure for solving mathematical problems

Definition of Algorithm: *An algorithm is a sequence of steps that describe how to obtain the solution to a given problem.*

Obviously, algorithm design is the most important step in the process of problem solving, while numerical methods provide support to the design of algorithms. Numerical methods and detailed algorithms for solving various mathematical problems will be given in chapter 2 to 13, while the general guidelines for algorithm design are given in the following.

Use the top to down design technique

In this technique, we first use the *Decomposition technique* to break the problem into a series of smaller problems, then use the *Stepwise refinement technique* to describe each smaller problem in greater detail.

The advantage of decomposition is that we can initially think of the overall steps required without getting lost in the details. Details are introduced only as we begin the refinement of our algorithm.

Use flowchart and pseudocode to assist in the design of algorithms

Two tools can be used to assist us in designing and presenting an algorithm, namely

Pseudocode	- *show the steps in a series of English - like language.*
Flowchart	- *show the steps in graphic form.*

Use structured algorithm

To improve the readability, we should use a set of standard forms (structures) to describe an algorithm. There are three kinds of standard algorithm structures

- Sequence structures
- Selection structures
- Repetition structures

An algorithm formed by a standard structure is called a *structured algorithm*. When a structured algorithm is converted to a computer program, the computer program is called a *structured program*.

Example 1.1: Given a set of experiment data x_i, compute the average value.

Algorithm Design

Decomposition: To compute an average, we need to sum and count the values. Thus our decomposition is

- Read data values and keep a sum and count
- Divide the sum by the number of data (count) to get the average
- Print the average.

Refinement:

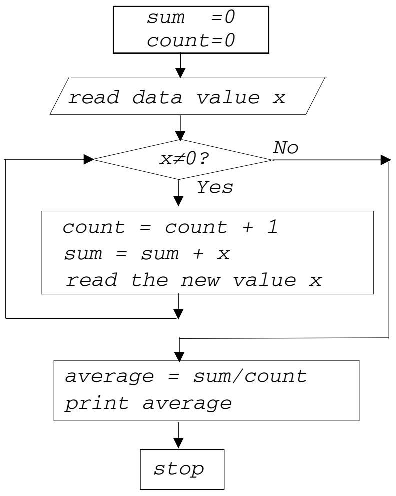
- Step 1 can be refined by setting a count and a sum to zero following by a loop to read the values and update the count and the sum
- The loop should be structured into a while loop, thus we must determine a condition necessary to keep us in loop. For example, if all data values are non-zero, we can put zero at the end of the data to indicate the end of data. So the loop is 'while the data is non zero do ....' .

Algorithm Presentation

Pseudocodes

```
Set  Sum = 0
     Count = 0
Read data value x
while ($x \neq 0$) do
     Set Sum = Sum + x
     Count = Count + 1
     Read next data value x
Average = Sum/ Count
Print the Average
```

Flow Chart



1.3 Round – off Errors and Their Propagation in Computation

Numerical results are influenced by many types of errors. Some sources of errors are difficult to influence; others can be reduced or even eliminated by, for example, rewriting formulas or making other changes in the computational sequence. The major sources of errors are listed as follows:

- *Error in given Input data* such as errors in the input of irrational numbers and data from experiments;
- *Truncation errors* such as errors due to breaking off an infinite series after a finite number of terms;
- *Simplification in the mathematical model;*
- *Human errors and Machine errors*;
- *Round - off errors*

In this section, we introduce the concept of round of errors and then analyse their generation and propagation in computation using floating point arithmetic.

Concept of Round -Off Errors

The arithmetic in a machine involves numbers with only a finite number of digits and so errors automatically occur when values are combined through an arithmetic operation in computers.

Storage structure of numbers in computers

Numbers are stored in the computer as a sequence of binary (base 2) digits, or bits. To simplify the analysis of the effects of round-off errors, throughout this chapter a hypothetical decimal computer will be used, in which numbers are represented in the *normalised decimal floating point form:*

$$\pm 0.d_1d_2\ldots d_k \times 10^n, \qquad \text{where } 0 \le d_i \le 9 \quad \text{for } i = 1, 2, \ldots, k \tag{1.1}$$

The sequence of digits $d_1d_2\ldots$ is known as the **MANTISSA** and the power, n, as the **EXPONENT**.

As an example, 25/4 has the form $+0.625\times10^1$.

The finite size of the computer implies that there is a maximum number (say k) of digits with which a value can be represented; that is, the mantissa must contain only k digits**.** The value of k will vary for different computers and compilers.

Representation of real numbers in computers

Any real number, x, can be represented in the form

$$x = \pm 0.d_1d_2\ldots d_kd_{k+1}\ldots\times10^n.$$

However, this number can only be represented by its floating point form (1.1) in computers. The floating point form, denoted by *fl(x)*, is obtained by terminating the mantissa of x after k digits. There are two common ways of doing this.

- One is known as **Chopping or Truncating**: the digits $d_{k+1}, d_{k+2},\ldots$ are chopped from the mantissa to give

$$fl(x) = \pm 0.d_1d_2\ldots d_k \times 10^n.$$

- The second method is known as **Rounding**. In this case,

  if $d_{k+1} \ge 5$, we add one to d_k to obtain $fl(x)$, that is we round up.
  If $d_{k+1} < 5$, we merely chop off all but the first k digits.

  For example, if $k = 4$ and rounding is used, 1.12356 are represented as $+0.1124\times10^1$
  while -7.22321 are represented as -0.7223×10^1

Round-Off Error

The term "**Round-Off Error**" is used to describe the difference between the exact representation of a number and the chopped or rounded machine representation.

Overflowed and Underflow

Beside limitation on the number of digits for the mantissa, there are also limitations on the size of the exponent n; $\boldsymbol{n}$ must satisfy the inequality $-m \le n \le M$ where M and m are positive integers, which may differ for different computers.

- If $n > M$, the number has become too large to be represented in the computer and is said to have **overflowed** and the execution of program is stopped with an error message.
- If $n < -m$, **Underflow** occurs. In this case some computers reset the value of the number to zero and continue the calculation; others give an error message.

Generation and Propagation of Round-off Errors

What happens to round-off errors when numbers are combined using floating arithmetic?

Denote $fl(x)$ and $fl(y)$ as the floating point representations of x and y ,

$\oplus$, Θ, $\otimes$ and $\varnothing$ as the machine operations of addition, subtraction, multiplication and division.

Then we can define the machine operations, for example, $x \oplus y = fl[fl(x) + fl(y)]$ and we are interested in the difference between $x + y$ and $x \oplus y$.

Before we can analyse the round-off error generated in machine operations, we need to know the following point: when an arithmetic is performed in a k-digit computer, it is performed in registers that can hold $2k$ digits (twice as many digits as the floating point representation of number). Then this $2k$-digit number is rounded to give a k-digit representation.

In the following, we consider each of the basic machine operations in turn.

Addition $x \oplus y = fl(fl(x) + fl(y))$

The error in $x \oplus y$ has two sources:

- the error in representing x and y by $fl(x)$ and $fl(y)$, and
- the error in representing the sum $fl(x) + fl(y)$ by $fl[fl(x) + fl(y)]$.

For example, if $x = 4/3$ and $y = 2/9$ then on a four-digit machine,

$fl(x) = 0.1333 \times 10^1$, (the error in representing x by $fl(x)$ is $x\text{-}fl(x)$=0.00033333)

$fl(y) = 0.2222 \times 10^0$.

Thus, $fl(x) + fl(y) = 0.1333 \times 10^1 + 0.2222 \times 10^0$.

As numbers that are to be added together must have the same exponent and so this sum is rewritten as

$$fl(x)+fl(y)=0.1333\times10^1+0.02222\times10^1=0.15552\times10^1$$

from which

$$x\oplus y=fl[fl(x)+fl(y)]=0.1555\times10^1$$

and an additional error 0.00002×10^1 in representing $fl(x)+fl(y)$ has occurred.

The effect of this kind of errors can be more noticeable when a sequence of numbers is being added. The order in which the numbers are added may affect the round-off error and, as a consequence, the sum.

For example, in a four-digit computer, for $x=0.1333\times10^1$ and y=0.3000×10^{-4}

$x\oplus y\oplus y\oplus y\oplus y=0.1333\times10^1=x$, (Exercise: derive this result)

but $y\oplus y\oplus y\oplus y\oplus x=0.1334\times10^1$, which is the correct rounded result.

Therefore, Ideally, when summing numbers of the same sign, they should be added in *order of increasing magnitude.*

Another way of avoiding the accumulation of errors is to accumulate the sum, product by product, in a variable with more digits of precision. Eg, to calculate the sum of a sequence of products

$$z=x_1y_1+x_2y_2+\ldots+x_ny_n,$$

we can use

```
INTEGER,PARAMETER :: d kind= SELECTED REAL KIND(p=14)
REAL (KIND=d kind):: z
REAL X(100),Y(100)
        Z=0.0
        DO 1, I = 1, N
             Z = Z + X(I)*Y(I)
     1  CONTINUE
```

Subtraction $x\Theta y=fl[fl(x)-fl(y)]$

A problem that can occur with this computation is illustrated as follows.

For $x=3/11$ and $y=8/29$, $fl(x)=0.2727\times10^0$ and $fl(y)=0.2759\times10^0$ and so

$$x\Theta y=fl[fl(x)-fl(y)]=fl(-0.0032\times10^0)=-0.3200\times10^{-2}.$$

But $fl[x-y]=fl(-1/319)=-0.3134\times10^{-2}$.

The problem is that the zeros in the first expression do not represent useful information; they are there to fill in the floating-point representation of the result of the computation. This lost of precision is called **Cancellation Error** and can be an important source of further error in numerical calculation. Thus, **calculations should be organised to avoid** (if possible) **subtraction of two close data.**

Multiplication $x \otimes y = fl\left[fl(x) \times fl(y)\right]$

This operation presents few problems apart from possible overflow or underflow. For example, the product $(0.2324 \times 10^M) \otimes (0.3245 \times 10^M)$ cannot be represented on a machine with maximum exponent *M*.

Division $x \oslash y = fl\left[fl(x) / fl(y)\right]$

This operation also presents few problems apart from *division by a number close to zero***.** The effect of this is to *magnify the error* that resulted from representing *x* in floating-point form.

For example, for $x = 5/9$ and $y = 1/9876$,

$$x \oslash y = 0.5484 \times 10^4$$

But $fl(x/y) = 0.5486 \times 10^4$.

The earlier rounding errors in *x* and *y* have been magnified.

1.4 ERROR MEASURES & ERROR TESTS

Two measures are usually used to quantify numerical errors.

Absolute Error

Definition: Let *x** be an approximation to *x*, then the absolute error in *x** is defined as $|x - x^*|$. Thus the floating-point representation of *x* has absolute error $\left|x - fl(x)\right|$.

Definition: Two numbers *x* and *x** are said to agree to *n* decimal places if *n* is the largest non-negative integer for which, $\left|x - x^*\right| < \frac{1}{2} \times 10^{-n}$ (i.e., $< 0.\overbrace{00......0}^{n-1}5$ or less than half unit of the nth decimal place).

For example, for $x = 23.496$ and $x^* = 23.494$, as the absolute error $|x - x^*| = 0.002$ is less than 1/2 unit of the 2*nd* decimal place, i.e.

$$\left|x - x^*\right| < \frac{1}{2} \times 10^{-2},$$

*x** and *x* are said to agree to 2 decimal places.

Relative Error

Definition: Let x^* be an approximation to x, then the relative error in x^* is defined as $\frac{|x-x^*|}{|x|}$ provided that x is not zero. Thus, the floating-point representation of x has relative error

$$\frac{|x-fl(x)|}{|x|}.$$

Definition: Two numbers x and x^* are said to agree to n significant digits (or figures) if n is the largest non-negative integer for which the relative error

$$\frac{|x-x^*|}{x} < 5\times 10^{-n} \qquad \text{(i.e., } < 0.\overbrace{00\ldots\ldots0}^{n-1}5\text{)}.$$

Example 1.2: Find x^* to approximate 1000 to four significant digits.

Solution By definition, x^* must satisfy

$$\frac{|x^*-1000|}{|1000|} < 5\times 10^{-4} \quad \Rightarrow \quad 999.5 < x^* < 1000.5.$$

ERROR TEST

In scientific computing, we usually need to check for the convergence of a sequence of iterations, for example, to determine whether x_{n+1} and x_n are close. We can use two different kinds of tests.

Absolute Error Test

$$|x_{n+1}-x_n| \le Tolerance$$

for some small positive value of tolerance specified beforehand.

Relative Error Test

$$\frac{|x_{n+1}-x_n|}{|x_n|} \le Tolerance$$

which can also be implemented in the form

$$|x_{n+1}-x_n| \le |x_n| \cdot Tolerance$$

to avoid problems that may arise if x_n is close to zero. However, the new form of the relative error test may cause another problem $|x_n|\cdot Tolerance$ is very small, possibly too small to be represented on the computer. In this case we shall use a number ε_m, usually called machine epsilon that is a measure of the precision we can obtain on a particular computer, and then replace $|x_n|\cdot Tolerance$ by a small multiple (say 5) of ε_m. The value of ε_m varies for computer to computer.

EXERCISE 1

Q1. 1 Answer the following questions:

(a) What are the essential parts of a computer system?
(b) What do we mean by computer memory?
(c) What are the differences between internal memory and external memory?
(d) What do we mean by software and hardware?
(e) What is the definition of algorithm? Briefly describe the top-down technique for the design of algorithms.
(f) How to present an algorithm?
(g) Why do we need to compile a C++ program before we can run the program?

Q1. 2 Consider the following values of p and p*. What is (i) the absolute error, (ii) the relative error in approximating p by p*?

(a) $p = \pi$, $p^* = 3.1$; (b) $p = 1/3$, $p^* = 0.333$; (c) $p = \pi/1000$, $p^* = 0.0031$.

(Ans: (a) 4.16×10^{-2}, 1.33×10^{-2}; (b) 3.34×10^{-4}, 1.00×10^{-3}; (c) 4.16×10^{-5}, 1.33×10^{-2})

Q1. 3 The number p* approximates p = 2.7182 to four significant digits. Find the largest interval in which p* can lie.

(Ans: 2.7168409, 2.7195591)

Programming

Q1. 4 To find the root of an equation of the form

$$x - f(x) = 0,$$

we can start from an initial guess x_o of the root , then improve the estimate by using the following iterations

$$x_{n+1} = f(x_n),$$

until the difference between x_{n+1} and x_n is sufficiently small. Now consider the following equation

$$x - 6^{-x} = 0, \qquad 0.3 \le x \le 0.6$$

(a) Write a program to find the root of the equation accurate to 3 decimal places. Use $x_0 = 0.45$.
(b) Modify the program in (a) such that the solution is accurate to 3 significant digits.

CHAPTER 2

Solution of Nonlinear Equations

Solution of many real world problems involves solving an equation of the form $f(x) = 0$, or in other words locating zeros of $f(x)$ or finding roots of $f(x)$. To solve $f(x) = 0$, we usually need to find an initial approximation to the root and then refine the approximation step by step. In this chapter, we present various methods for solving nonlinear equations.

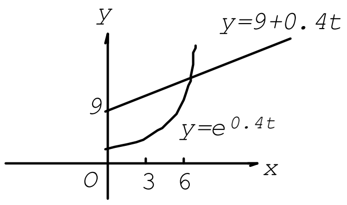
2.1 Graphical Solution

For a general nonlinear equation $f(x) = 0$, the real roots of the equation are the x values of the intersections of the graph $f(x)$ with the x-axis. We can thus determine the real roots by sketching the graph of $f(x)$ and finding the points where it crosses the x-axis.

Sometimes it is more convenient to rearrange the equation as $f_1(x) = f_2(x)$ and draw the graphs of f_1(x) and $f_2(x)$. In this case, the real roots of the original equation are the x values of the intersections of y $= f_1(x)$ with $y = f_2(x)$.

Example 2.1 Find the positive real roots of the equation

$$e^{0.4t} - 0.4t - 9 = 0.$$



Solution

By rearranging the equation, we have

$$e^{0.4t} = 9 + 0.4t.$$

The graph $y = e^{0.4t}$ intersects $y = 9 + 0.4t$ at $t \approx 6$

$\therefore \quad \exists$ a positive real root $t \approx 6$

If the nonlinear equation is a polynomial equation of degree n, then it always has exactly n roots, some of which may be multiple roots or complex roots.

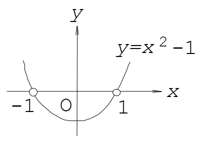
For a ***real single root*** x_0,	the graph	$y = f(x)$ crosses the x- axis at x_0;
For a ***repeated real root*** $x_1 = x_2$,		$y = f(x)$ touches the x- axis at x_1;
For a ***complex root***,		$y = f(x)$ may not cross or touch the x axis.

For example, $x^2 - 1 = 0 \Rightarrow x = \pm 1$:

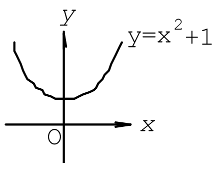
$y = f(x)$ crosses the x- axis at $x = 1$ and $x = -1$

$x^2 + 1 = 0 \Rightarrow x = \pm i$:

$y = f(x)$ does not cross or touch the x-axis



Real single roots x=-1, 1



Complex roots x=-i, i

2.2 Incremental – Search Method

This method is one of the simplest and most reliable methods for finding real roots of an equation, located within a selected region $[a, b]$.

The method is based on the fact that a single real root is the x value of the intersection of the curve $y = f(x)$ with the x-axis. Thus, if $f(x)$ is a continuous function and changes its sign from x_{i-1} to x_i, then the graph $y = f(x)$ must have crossed the x- axis at least once in the interval $[x_{i-1}, x_i]$ and hence $[x_{i-1}, x_i]$ contains at least one real single root.

Algorithm:

Step 1. Select a search region $[a, b]$;

Step 2. Divide the region into N sub-intervals with nodes $x_0=a, x_1, x_2, ..., x_n= b$;

Step 3. Proceed from left to right, if $f(x_{i-1}).f(x_i)<0$, then the ith interval $[x_{i-1}, x_i]$ contains at least one real root.

Note: *The process can be used in a subinterval to refine the result.*

2.3 Bisection Method

The problem to solve here is: for a given interval $[a, b]$ which is assumed to contain a unique real root of the equation $f(x) = 0$, find an interval containing the root but with a much smaller size.

The bisection method is an iterative process for finding the root of $f(x) = 0$. In iteration one, the initial interval $[a, b]$ is halved into two sub-intervals one of which will contain the root.

We keep the sub-interval containing the root as the new initial interval and repeat the process until the size of the interval containing the root becomes sufficiently small. Then we take the midpoint of the final sub-interval as the estimate of the root.

The key point in the algorithm is that when an interval $[a, b]$ is divided into two sub-intervals $[a, p]$ and $[p, b]$, how to determine which of the sub-intervals contains the root. As $[a, b]$ contains a unique real root, the root will be either at p or in one of the sub-intervals. If $f(p) = 0$, p is the root; otherwise $f(p)$ will have different sign with either $f(a)$ or $f(b)$ but not both. If $f(p)$ has different sign with $f(a)$, then (a, p) contains the root or otherwise (p, b) contains the root.

Thus, stating with an initial interval $[a, b]$ with $f(a)f(b)<0$ and containing only one real root, the basic steps used to find the root include

Step 1, divide $[a, b]$ into two subintervals $[a, p]$ and $[p, b]$ where $p = (a+b)/2$ is the midpoint of $[a, b]$. If $f(p)=0$, then $x = p$ is the solution, or otherwise go to step 2.

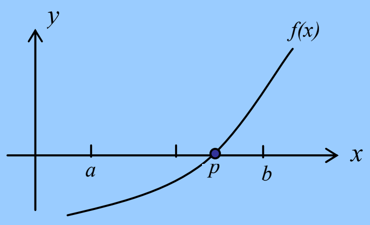
Step 2, if $f(a).f(p)<0$, then $[a, p]$ contains the root and thus the new smaller interval for further analysis is $[a, b]$ with $b= p$; otherwise $[p, b]$ contains the root and the new smaller interval is $[a, b]$ with $a= p$.

Step 3, repeat steps 1 and 2 with the new smaller interval until $f(p)=0$ or $\frac{|b-a|}{2} < Tol$.

Algorithm:

To find a solution to $f(x)=0$ given $f(x) \in C[a,b]$ where $f(a)$ and $f(b)$ have opposite sign.

```
Input a, b, Tol and MaxNit.
Set i=1
While (i<MaxNit ) do
     Set p=(a+b)/2
     If  f(p) = 0 or (b – a)/2<Tol   then
          output p; (procedure completed successfully)
          stop
     else
          Set i = i+1
          If {f(a).f(p)<0 }   then
               set b = p
          else
               set a = p
 Output ('method failed after MaxNit iterations, MaxNit=', MaxNit)
```



Remarks :

1) The sequence generated by the method always converges to the solution since we keep the solution x inside the interval located at each step.
2) The sequence $\{p_n\}$ generated to approximate the solution has the error bound

$$|p_n - p| \le \frac{b-a}{2^n}.$$

3) Order of convergence:

As $e_n = |p_n - p| \le |b_n - a_n|, \quad e_{n+1} = |p_{n+1} - p| \le \frac{|b_n - a_n|}{2} = \frac{e_n}{2}$,

the bisection method has linear convergence rate.

2.4 Successive Substitution (fixed – point iteration)

To find a root of $f(x) = 0$, we manipulate the equation into the form

$$x = g(x). \tag{2.1}$$

Any solution of this equation is called a *fixed point* of $g(x)$. As (2.1) is a rearrangement of $f(x) = 0$, any fixed point of $g(x)$ is guaranteed to be a root of $f(x) = 0$.

With the new equation, we can set up an iteration, namely *fixed point iteration*

$$x_{n+1} = g(x_n), \qquad n = 0,\ 1,\ 2,\ \ldots$$

to generate a sequence $\{x_1, x_2, \ldots\}$ beginning with an initial guess x_o. Our hope is that $x_n \to \alpha$, which satisfies $\alpha = g(\alpha)$, i.e., a fixed point of $g(x)$ or a root of $f(x) = 0$.

Usually, for a given equation $f(x) = 0$, there are many different ways of rearranging $f(x) = 0$ in the form of (2.1). However, only some of these are likely to give rise to successful iterations and yield the fixed point, as illustrated in the following example.

Example 2.2: Find a root of $x^2 - 2x - 8 = 0$.

Solution Consider $x^2 - 2x - 8 = 0$. We can have the following three rearrangements

$$\text{(a)}\ \ x = \sqrt{2x+8} =: g_1(x); \quad \text{(b)}\ \ x = \frac{2x+8}{x} =: g_2(x); \quad \text{(c)}\ \ x = \frac{x^2-8}{2} =: g_3(x).$$

Numerical results for the corresponding iterations with x_o=5 are given in the following table.

n	$x_{n+1} = g_1(x_n)$	$x_{n+1} = g_2(x_n)$	$x_{n+1} = g_3(x_n)$
0	5	5	5
1	4.243	3.600	8.5
2	4.060	4.222	32.125
3	4.015	3.895	512.008
5	4.001	3.973	
6	4.000	4.013	
10		4.001	
11		4.000	

Remarks:

Obviously, the sequence converges for schemes (a) and (b) but divergences for (c). This highlights the need for a mathematical analysis of the method.

Key problems to be analyzed

- How to choose x_0 and $g(x)$ such that the sequence x_o, x_1, x_2 .... converges to the fixed point of $g(x)$ with high convergence rate ?
- How to estimate the rate of convergence and error?

In the following, we will study these problems regarding the solution of $x = g(x)$.

1) Sufficient Conditions for the Existence, Uniqueness and Convergence

Theorem 2.1

1) If $g(x) \in C[a, b]$ and $a \le g(x) \le b \ \ \forall x \in [a, b]$, then $\exists$ a solution in $[a, b]$.

2) If also $g'(x)$ exists on (a, b) and $|g'(x)| \le k < 1 \ \ \forall x \in (a, b)$, then

- $g(x)$ has a unique fixed point in $[a,b]$.
- for any $x_o \in [a,b]$, the sequence generated by $x_n = g(x_{n-1})$ $(n \ge 1)$ converges to the solution

Proof:

Existence

If $g(a) = a$ or $g(b) = b$, then there exists a fixed point either in a or in b and thus the existence is proved;

If not, then it must be true that $g(a) > a$ and $g(b) < b$. Define $h(x) = g(x) - x$, then $h(x) \in C[a,b]$ with property $h(a) > 0$ and $h(b) < 0$. So by the intermediate value theorem, there exists $c \in [a,b]$ such that $h(c)=0$

$$\therefore \quad g(c) = c \ ,$$

which means that $x = c$ is a fixed point of $g(x)$, and thus the existence is proved.

Uniqueness

Let $p \ne q$ both be fixed points of $g(x)$, then

$$|p-q| = |g(p) - g(q)| \overset{\text{using Mean Value Theorem}}{=} |g'(c)||p-q| \le k|p-q| < |p-q| \quad (\text{as } k<1)$$

which is a contradiction and hence $p = q$.

Convergence

Since g maps $[a, b]$ into $[a, b]$, the sequence $\{x_n\}_{n=0}^{n=\infty}$ is defined and $x_n \in [a,b]$.

Denote x^* as the fixed point of $g(x)$, i.e. $x^* = g(x^*)$, then

$$|x_n - x^*| = |g(x_{n-1}) - g(x^*)| \overset{\text{using M.V.T.}}{=} |g'(\xi)||x_{n-1} - x^*| \le k|x_{n-1} - x^*| \quad (\text{where } \xi \in [\text{a,b}]) .$$

Applying the above inequality inductively gives

$$|x_n - x^*| \le k|x_{n-1} - x^*| \le k^2|x_{n-2} - x^*| \le \ldots \le k^n|x_0 - x^*|.$$

Then

$$\lim_{n\to\infty} |x_n - x^*| = |x_0 - x^*| \lim_{n\to\infty} k^n = 0, \quad \text{as } k < 1.$$

Therefore

$$x_n \to x^*, \quad \text{and so convergence is proved.} \qquad \square$$

Corollary

If $g(x)$ satisfies the hypotheses of the convergence theorem, the bound of error involved in using x_n to approximate x^* is given by

$$|x_n - x^*| \le k^n \max\{x_0 - a,\ \text{b} - x_0\}; \qquad |x_n - x^*| \le \frac{k^n}{1-k}|x_1 - x_0| \qquad \text{for all } n > 1.$$

Example 2.3 Given equation $x - \pi/2 - 1/3\sin x = 0$

a) Show that the equation has a unique root in $[0, \pi]$.

b) Show that for any $x0$ in $[0, \pi]$, the fixed point iteration

$x_{i+1} = g(x_i)$ with $g(x) = \frac{\pi}{2} + \frac{1}{3}\sin x$ will generate a sequence converging to the root.

c) Perform two iterations.

d) Estimate the number of iterations necessary to obtain approximation accurate to within 10^{-4}

Solution

a) Choose $g(x) = \pi/2 + 1/3 \sin x$. Then the following conditions hold

1) $g(x) \in C[a, b]$

2) $\underset{x\in[0,\pi]}{Max}\ g(x) = \pi/2 + 1/3 < \pi;$ $\quad \underset{x\in[0,\pi]}{Min}\ g(x) = \pi/2 + 0 = \pi/2 > 0.$

The above results imply that $0 < g(x) < \pi \quad \forall x \in [0, \pi]$.

3) $|g'(x)| = \frac{1}{3}|\cos(x)| \le \frac{1}{3} = k_0 \qquad \forall x \in [0, \pi]$

Therefore, there exists a unique solution in $[0, \pi]$.

b) As $g(x)$ satisfies the conditions (1) – (3) in (a), for any $x_0 \in [0, \pi]$ the given fixed point iteration will generate a sequence converging to the root.

c) Choose $x_0 = \pi$, then

$$x_1 = \pi/2 + 1/3 \sin \pi = \pi/2;$$

$$x_2 = \pi/2 + 1/2 \sin(\pi/2) = \pi/2 + 1/3$$

d) As

$$|x_n - x^*| \le k^n \max(\pi - 0, \pi - \pi) = \pi k^n .$$

To satisfy $|x_n - x^*| < 10^{-4}$, we choose n such that $\pi k^n < 10^{-4}$. Therefore

$$n \ln k < \ln \tfrac{10^{-4}}{\pi} \qquad \Rightarrow \qquad n > -\ln \tfrac{10^{-4}}{\pi} \Big/ \ln \tfrac{1}{3} \approx 10$$

2) Convergence (Stopping) Criteria

In iterative methods, we start with an initial estimate of the root and then generate a new estimate to improve the previous one. The process is repeated continuously until the estimate is sufficiently close to the true value of the root. However, it is usually difficult to determine the difference between an estimate and the true value. Hence one of the key problems in iteration is deciding when it is time to stop, or what the convergence criteria for the problem are.

Essentially there are three possible criteria that we might use to terminate an iterative search for a root, all of which depend upon some value becoming less than some small number *Tolerance*. Suppose that the iterative method successively generates the values x_0 , x_1 , x_2 , ... Then the ***convergence criteria*** are

1) $f(x_n) < Tol$; 2) $|x_n - x_{n-1}| < Tol$; 3) $\dfrac{|x_n - x_{n-1}|}{|x_n|} < Tol$ or combination of them.

Algorithm:

```
Input initial approximation x0, Tolerance Tol & maximum number of iteration MaxNit.

Set iter = 0
While (iter < MaxNit ) do
      Calculate new approximation  $x = g(x_0)$
       If (the result is convergent) then
            output the solution x ; (procedure completed successfully)
               stop
      else
               set:  $x_0 = x$
               set:  iter = iter+1
      end if
Output ('method failed after MaxNitr iterations, MaxNit=', MaxNit)
```

3) Error Analysis and Rate of Convergence

Definition: If a sequence $\{x_n\}$ converges to x and $\lim_{n\to\infty} \frac{|x_{n+1}-x|}{|x_n - x|^\alpha} = \lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^\alpha} = \lambda$,

Then we say the convergence rate of the sequence is of order α:

α -- the **order of convergence**, $\alpha = 1$ gives linear convergence;
$\alpha = 2$ represents quadratic convergence.
A sequence with higher order of convergence converges more rapidly.

λ -- **asymptotic error constant**,
which affects the speed of convergence but is not as important as α. A sequence with lower value of λ converges more rapidly.

Error Analysis (Linear convergence rate)

Assume that $g(x)$ satisfies the convergence condition of a fixed-point iteration.

Let $x_n = x + e_n$,

then $x_{n+1} = x + e_{n+1} = g(x_n) = g(x+e_n) = g(x) + e_n g'(x) + O(e_n^2)$.

As $x = g(x)$, the above equation becomes

$$e_{n+1} \approx e_n g'(x) \quad \text{for all } |e_n| << 1.$$

Hence

$$\frac{e_{n+1}}{e_n} = g'(x) = k.$$

Thus if $g'(x) = k \neq 0$, the fixed-point iteration has linear convergence and the value of k determines the rate of convergence. The smaller the value of k, the faster the convergence rate. So for fixed point iterations, ideally **choose *g*(*x*) such that** $|g'(x)| < 1$ **and as small as possible**.

Condition for Quadratic Convergence

Let s be a solution of $x = g(x)$. Suppose $g'(s) = 0$ and $g''(s)$ is continuous and strictly bounded by M on an open interval containing s, then $\exists$ a $\delta > 0$ such that for $x_0 \in [s-\delta, s+\delta]$, the sequence defined by $x_{n+1} = g(x_n)$ converges at least quadratically to s. Moreover for a sufficiently large value of n,

$$|x_{n+1} - s| < \frac{M}{2}(x_n - s)^2.$$

Proof

$$g(x) = g(s) + g'(s)(x-s) + g''(\xi)(x-s)^2/2, \quad \xi \in [s, x].$$

$$g'(s) = 0 \quad \Rightarrow \quad g(x) = s + \frac{g''(\xi)}{2}(x-s)^2$$

For $x = x_n$:
$$x_{n+1} = g(x_n) = s + \frac{g''(\xi)}{2}(x_n - s)^2$$

$$\therefore \quad \lim_{n\to\infty} \frac{|x_{n+1} - s|}{|x_n - s|^2} = \lim_{n\to\infty} \left|\frac{g''(\xi)}{2}\right| = \frac{|g''(s)|}{2}$$

Thus, if $g(x)$ satisfies the convergence condition and $g'(s) = 0$, then the convergence rate is at least of order two.

2.5 Newton – Raphson Method

The N-R method is one of the most powerful and well-known numerical methods for solving a root-finding problem.

The method starts with an initial estimate x_0 and refines the approximation step by step.

Graphically, the solution of $f(x) = 0$ is the intersection of $y = f(x)$ with the x-axis (the s as shown). To get an estimate of s from the point $[x_0, f(x_0)]$ on the curve $y = f(x)$, we draw a straight line tangent and find its intersection with the x-axis, x_1, and use this as the new approximation of s. By repeating this process, we can gradually approach s.

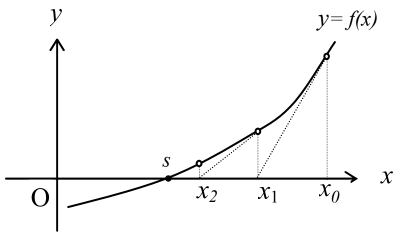
The equation of the tangent line through point $[x_o, f(x_o)]$ is $\dfrac{y - f(x_0)}{x - x_0} = f'(x_0)$

As at $y = 0$, $x = x_1$, we can thus obtain x_1 by letting $y = 0$ and solve the above equation for x. So

$$y = 0 \quad \Rightarrow \quad x = x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Denote x_1 by x_{n+1} and x_0 by x_n, we can rewrite the above formula as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad \text{for} \quad n=0,1,2,\ldots$$



Example 2.4: Solve $x^3 - 4x - 9 = 0$ using Newton method.

Solution Firstly, use the bisection method to get an initial guess to the root.

As $f(2) < 0$, $f(3) > 0$, there exists a root in $(2, 3)$.

Letting $x_0 = 2.5$ and using Newton's method, we have

$f(x) = x^3 - 4x - 9$

$f'(x) = 3x^2 - 4$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 2.5 - \frac{2.5^3 - 4*2.5 - 9}{3*2.5^2 - 4} = 2.7288$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 2.7067$$

$x_3 = 2.7065$

Algorithm

To find a solution to $f(x) = 0$ given an initial approximation x0.

```
Input x0, Tol and MaxNit.
Set i =1
While (i < MaxNit) do
    Set x = x0 – f(x0)/ f'(x0)
    If ( |x–x0| < Tol ) then
        output x; (procedure completed successfully)
        stop
    else
        Set i = i+1
        x0 = x
Output ('method failed after MaxNit iterations, MaxNit =', MaxNit)
```

Remarks: The N-R method can be used to find *complex roots* when complex variables are used and when the initial guess is a complex value.

Convergence Theorem:

Let $f(x) \in C^2[a, b]$ and $s \in [a, b]$ is such that $f(s) = 0$ and $f'(s) \neq 0$. Then if the initial guess x_0 is chosen sufficiently close to s, Newton's method will generate a sequence $\{x_n\}_{n=1}^{\infty}$ converging to s.

Proof. (exercise)

Hint: Consider Newton's method as a functional iteration scheme

$$x_{n+1} = g(x_n) \text{ with } g(x) = x - \frac{f(x)}{f'(x)}.$$

Then show that there exists a neighbourhood of s ($[s - \delta, s + \delta]$) such that g satisfies the convergence conditions for the fixed point iteration.

Error Analysis

Definition of Simple Roots and Multiple Roots

f(x) has a ***simple root*** at s if $f(x)=(x-s)q(x)$ where $q(s)\neq 0$.

f(x) has ***multiple root*** of multiplicity m if $f(x)=(x-s)^m q(x)$ where $q(s)\neq 0$.

Notes: Newton's method is not so good at finding multiple roots since $f'(s)=0$..

Convergence Rate

Denote s and e_n as exact solution and the error at the nth iteration, then

$$x_n = s + e_n$$

$$x_{n+1} = s + e_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = s + e_n - \frac{f(s+e_n)}{f'(s+e_n)}$$

$$e_{n+1} = e_n - \frac{f(s)+f'(s)e_n + \frac{1}{2}f''(\xi_1)e_n^2}{f'(s)+f''(\xi_2)e_n} = \frac{\left(f''(\xi_2)-\frac{1}{2}f''(\xi_1)\right)e_n^2}{f'(s)+f''(\xi_2)e_n},$$

where both ξ_1 and ξ_2 are in between s and (x_n+e_n).

Thus, $$\lim_{n\to\infty} e_{n+1} = \begin{cases} \dfrac{f''(s)}{2f'(s)}e_n^2 & \text{if } f'(s)\neq 0 \quad \rightarrow \quad \text{quadratic convergence} \\ \dfrac{1}{2}e_n & \text{if } f'(s)=0 \quad \rightarrow \quad \text{linear convergence} \end{cases}$$

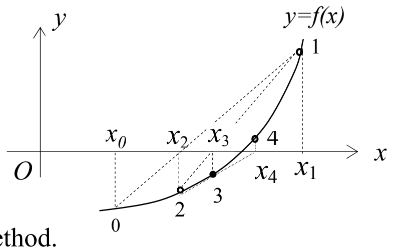
Limitation

- Newton’s method usually requires a good initial guess for the root. If the guess is not good enough, the algorithm may miss the root entirely, finding instead one of the other roots or not finding a root at all. Thus usually we use a globally convergence scheme to find an initial estimate then use Newton's scheme to accelerate the convergence rate.
- Problems may also arise for certain kinds of equations. Places where $f'(x)$ is zero or close to zero at peaks or inflection points in the curve can cause difficulties.
- The method requires evaluating the derivative of the function that is usually far more difficult.

2.6 The Secant Method

To circumvent the problem of derivative evaluation in Newton's method, we approximate the derivative by

$$f'(x_n) = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$



Using this approximation in Newton's formula gives

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

The technique using this formula is called the Secant method.

It can be proved that the convergence rate of this method is slower than Newton's method in terms of iterations.

2.7 Accelerating Convergence- Aitken's Δ^2 process*

The Aitken's Δ^2 process can be used to 'speed up' convergence of any linearly convergent process.

Suppose $\{x_n\}$ is a linearly convergent sequence with limit s and asymptotic error constant $\lambda<1$, that is

$$\lim_{n\to\infty} \frac{|x_{n+1} - s|}{|x_n - s|} = \lambda < 1.$$

Assume also that the signs of $x_n - s$, $x_{n+1} - s$ and $x_{n+2} - s$ agree and that n is sufficiently large that

$$\frac{x_{n+1} - s}{x_n - s} \approx \frac{x_{n+2} - s}{x_{n+1} - s}.$$

Solving the above equation for s yields

$$s \approx \frac{x_{n+2}x_n - x_{n+1}^2}{x_{n+2} - 2x_{n+1} + x_n} = x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n} = x_n - \frac{(\Delta x_n)^2}{\Delta^2 x_n} =: \hat{x}_n. \qquad (2.2)$$

Theorem 2.2:

Let $\{x_n\}$ be a sequence that converges linearly to the limit s with asymptotic constant $\lambda<1$ and $x_n - s \neq 0$ for all $n \geq 0$. Then the sequence $\{\hat{x}_n\}_{n=1}^{\infty}$ converges to s faster than $\{x_n\}_{n=1}^{\infty}$ in the sense that $\lim\limits_{n\to\infty} \dfrac{\hat{x}_n - s}{x_n - s} = 0$.

Proof

Hint: Let $\delta_n = (x_{n+1} - s)/(x_n - s) - \lambda$ and show that $\lim\limits_{n\to\infty} \delta_n = 0$.

Then express $(\hat{x}_n - s)/(x_n - s)$ in terms of δ_n, δ_{n+1} and λ.

The direct procedure for constructing $\{\hat{x}_n\}_{n=0}^{\infty}$ is :

$$\begin{tikzcd}
x_0, & x_1 \arrow[d] & , x_2, & x_3, & x_4 \arrow[d] & , x_5, & x_6, & x_7 \arrow[d] & , x_8, & x_{3n}, & x_{3n+1} \arrow[d] & , x_{3n+2} \\
 & \hat{x}_0, \arrow[urr, dashed] & & & \hat{x}_1, \arrow[urr, dashed] & & & \hat{x}_2, \arrow[urr, dashed] & & & \hat{x}_n &
\end{tikzcd}$$

In practice, for all values of n, the $\hat{x}_{n-1}$ calculated from the Δ^2 process is usually a more accurate approximation to s than is the value x_{3n} obtained from $g(x_{3n-1})$. Hence, a modified Aitken's Δ^2 method, namely the Steffensen's method, is usually used . In this method, we construct in order

$$x_o, \left\{\left(x_{3n+1}, x_{3n+2}, \hat{x}_n\right)\right\}_{n=0}^{\infty},$$

where $x_n = g(x_{n-1})$ and $\hat{x}_n$ is as defined in (2.2).

Algorithm

To find a solution of an equation using the fixed-point iteration $x = g(x)$ and the Δ^2 process given an initial approximation x0.

```
Input x0, Tol and Max_iter.
Set I=1
while ($i <$ max_iter) Do
        x1 = g(x0)
        x2 = g(x1)
        x = x0 – (x1– x0)$^2$/(x2 – 2x1+x0)
        if ($|x - x0| < Tol$) then
              output (‘solution is: x =‘, x)
              stop
        else
              $i = i + 1$
              set x0 = x
output (‘method failed after Max_iter iterations, max_iter =‘, max_iter)
```

2.8 Solution of Nonlinear Equations using Maple/MATLAB

Solution of Nonlinear Equations using Maple

The Maple **`fsolve`** function can be used to solve a nonlinear equation $f(x)=0$ numerically. The Syntax is

```
fsolve(f(x), variable, [option])
```

where *variable* specifies the variable of the equation (eg x), and the optional arguments may include

complex	:	find roots over the complex plane
interval	:	specify the range where solutions are to be found, eg. x=-2..-1;
var = value	:	with this, you can specify starting values for the variables, eg. x=2.0 (if this is to be used, then cannot include the *variable* argument)
maxsols = n	:	find only the n least roots.
avoid = s	:	avoid certain values when searching for roots, eg. avoid={x=0, x=2}

Example 1. solve $e^{-x}+2x-2=0$ on [-2, -1] .

```
> x_sol=fsolve(exp(-x)+2*x-2,x,x=-2..-1);
```

yields $\boxed{x\_sol = -1.678346990}$

Example 2. solve $x^3-1.58x^2-4.04x+2.29=0$.

```
> x_sol=fsolve(x^3-2*x^2+x-2, x=2, complex);
```

yields $\boxed{x\_sol = (-1.000000000*I, 1.000000000*I, 2.)}$

Solution of Nonlinear Equations using MATLAB

In Matlab, an equation $f(x)=0$ can be solved by using the "**`fsolve()`**" function as below

```
>> xsol = fsolve(@(x) f(x), x0)
```

where `x0` is the initial guess of the solution.

Example Solve $e^{-x}+2x-2=0$ for x with initial guess $x0$=1.

```
>> xsol = fsolve(@(x) (exp(-x)+2*x-2), 1)
```

yields

```
xsol  =
        0.7680
```

SUMMARY

In this chapter, we discuss several methods for finding roots of nonlinear equations: graphical solution method, incremental search method, bisection method, fixed point iterations, Newton's method, secant method and etc. The graphic method is useful for finding an initial guess. The incremental search method is useful for finding intervals that contain roots. The bisection method and the secant method begin with an interval that contains a root and then determine a smaller subinterval that contains the root. The fixed-point iterations and Newton's method start with an initial guess and refine the estimate through an iteration process.

EXERCISE 2

Q2.1 Show, by using the graphical method, that the following equations have exactly one root and that in each case the root lies in the interval [0.5, 1].

$(a)\ x-\cos x=0;\quad (b)\, x^2+\ln x=0;\quad (c)\, x\mathrm{e}^x-1=0$

Q2.2. Consider the following equation

$$x^3-1.58x^2-4.04x+2.29=0.$$

(a) How many roots exist for this equation?
(b) Starting at x = –10, use the incremental search method to find all intervals of size 1.0 in the interval [–10, 10] that contain roots for this equation.
(c) Explain why you might find fewer than three small intervals containing roots for cubic equations over a given interval
(d) Explain why you should be concerned if you find more than 3 small intervals containing roots for a cubic equation over a given interval.

Q2. 3 The interval [2, 3] contains a root of the equation in Q9. 2.
(a) Use two iterations of the bisection method to approximate the root in the interval (mark the subinterval discarded in each iteration).
(b) Use the bisection method to approximate a root x in the interval such that $|f(x)<0.1|$
(Ans: (a) 2.625)

Q2. 4 Consider the nonlinear equation $\mathrm{e}^{-x}+2x-2=0$ (sections2.1 and 2. 2)
(a) show that the equation has roots in both intervals [–2, –1] and [0, 0.8].
(b) determine which of the following iteration schemes will give a sequence converging to the solution in [0, 0.8] for any $x_0\in[0,\ 0.8]$ (reasons should be given)

$(i)\ x_{n+1}=1-\frac{1}{2}e^{-x_n}$ (ii) $x_{n+1}=-\ln 2-\ln(1-x_n)$ (answer (i))

(c) perform 3 iterations using the scheme determined in (b).

Q2.5 For equation $x = \frac{5}{x^2} + 2$, determine an interval [*a, b*] on which fixed point iteration will converge. Perform two iterations and estimate the number of iterations necessary to obtain the approximation accurate to within 10^{-5}
(Ans: [2.5, 3.0], 2.69065)

Q2.6 a) Use the convergence theorem for fixed point iteration to show that the sequence defined by

$$x_n = \frac{1}{2}x_{n-1} + \frac{1}{x_{n-1}}, \quad \text{for } n \geq 1, \quad \text{converges to } \sqrt{2} \text{ whenever } x_0 > \sqrt{2}.$$

b) Use the fact that $0 < \left(x_0 - \sqrt{2}\right)^2$ whenever $x_0 \neq \sqrt{2}$ to show that if $0 < x_0 < \sqrt{2}$, then $x_1 > \sqrt{2}$.

c) Use the result in parts (a) and (b) to show that the sequence in (a) converges to $\sqrt{2}$, whenever $x_0 > 0$.

Q2. 7 Use the Newton's -Raphson method to solve equation $x^3 - 1.58x^2 - 4.04x + 2.29 = 0$.
Perform two iterations with $x_0 = 3$
(Ans: 2.781)

Programming

Q2. 8 Write the program using the bisection method in the following aspects and then use the program to solve the equation in Q2. 4.

1) Declare all real variables using an appropriate kind type value such that they can be used to store values with at least 10 digits of precision;
2) Change the block if construct in the main program unit to a case construct;
3) Print results using formatted output statements;
4) Assign the value of the root computed to a variable with name EQ_root;
5) Stop iteration if $f(x\_mid) = 0$ or $|x\_right - x\_left| < tolerance$.

Q2.9 Write a subroutine ITERAT to implement the fixed-point iteration method. Then write a main program which calls ITERAT to find roots accurate to within 10^{-5} for the equations in Q2. 4 and Q2. 5.

Requirement: 1) Define the fixed point iteration function g(x) using a function subprogram
FUNCTION G(X)
2) Implement the fixed point iteration method using a subroutine
SUBROUTINE ITERAT(G, X0, Tol, Max_iter, X, err)

where

G	= Fixed point iteration function
*X*0	= Before entry, must be set to the initial guess of the root x0.
Tol	= Input. Before entry, must be set to the value of Tolerance
Max_iter	= Input. Before entry, must be set to the maximum number of iterations to be used
X	= *Output. On exit, X contains the value of the root.*
err	= *Output. On exit,* *err*=0 indicates that the procedure completed successfully; *err* =1 indicates that the method failed after Niter iterations.

3) Write a main program to read $x0$, *tol* & *Max_iter*, to call the subroutine and to print the result or error message.

Q2.10 Write a subroutine NEWTON to implement the Newton-Raphson method for solving $f(x) = 0$. Then write a main program which calls the subroutine to find the root accurate to within 10^{-5} for the equation in **Q**2. 7.
Requirement: as in **Q**2. 9.

Q2.11 Modify the program Write a subroutine BISEC to implement the Bisection method for solving $f(x) = 0$. Then write a main program which calls the subroutine to find roots accurate to the within 10^{-5} for the equation in **Q**2.5.

Requirement:

1) Define f(x) using a function subprogram or a statement function
2) Implement the Bisection method using a subroutine
   SUBROUTINE BISET(a, b, Tol, Nite, X, Ierr)
   where a, b, Tol, Niter = *Input, as defined in section* 2.3.
   X, Ierr = *Output, as defined in* Q2.7.
3) Write a main prog. to read *a, b, tol* and *Niter*, to call the subroutine and to print the result.

Q2.12 Solve **Q**2.2 and **Q**2.5 using Maple and MATLAB.

Chapter 3 Direct Methods for Systems of Linear Equations

A general linear system of n equations in n unknowns can be written in the form

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots\dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots\dots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots\dots + a_{nn}x_n &= b_n \end{aligned} \tag{3.1}$$

or in matrix form $Ax=b$.

At present, many techniques are available for the solution of linear systems of equations. These techniques are classified into direct methods and iterative methods. In this chapter, we study various direct methods for solving linear systems. The rest of the chapter is organised as follows.

Section 3.1	gives the preliminary for the solution of linear systems.
Section 3.2	introduces the Gaussian elimination method.
Section 3.3	presents the LU factorization method and analyses its connection with the Gaussian elimination method
Section 3.4	first shows the need of performing row interchanges and then introduces the scaled maximum column pivoting technique
Section 3.5	introduces the permuted LU method and analyses its connection with the Gaussian elimination method with row interchange.
Section 3.6	presents various efficient techniques for the solution of various special types of linear systems including positive definite systems and tridiagonal systems.
Section 3.7	describes how to find the inverse of a matrix by solving linear systems
Section 3.8	describes how to solve linear systems using Maple/MATLAB functions

3.1 Preliminary

Existence and Uniqueness of Solution

For a system of n equations in terms of n unknowns $Ax= b$, we can construct an augmented matrix $(A|b)$ to characterise the linear system. Based on the ranks of A and $(A|b)$, there are three possibilities of solutions

• A unique solution for x	$\leftarrow$	if Rank (A) = Rank $(A|b) = n$
• Infinitely many solutions for x	$\leftarrow$	if Rank (A) = Rank $(A|b) < n$
• No solution for x (inconsistent)	$\leftarrow$	if Rank (A) < Rank $(A|b)$

Elementary Row Operations

The following three elementary row operations can be used to simplify the augmented matrix (i.e., the linear system)

- Multiplying a row by a non-zero scalar
- Adding a multiple of one row to a different row
- Interchanging rows

Application of Row Operations in Solving linear Systems

If an augmented matrix $C' = [C \mid d]$ is obtained from $A' = [A \mid b]$ by a finite number of elementary row operations, then C' is row equivalent to A' and the solution of $Cx = d$ is identical to the solution of $Ax = b$. Hence, given a linear system, one could use row operations to reduce the system into a simpler system and solve as such. The Gaussian elimination method presented in the next section is based on this idea.

3.2 Gaussian Elimination

Solving a linear system $Ax = b$ by Gaussian elimination includes two phases: eliminating process and backward substitution process.

Elimination Process

The elimination process reduces the system by row operations to an equivalent simpler system $Ux = y$ in which U is an upper triangular matrix. This process requires (n –1) steps.

Step 1 (Assume $a_{11} \neq 0$). Eliminate the 1st unknown from equations (2 – N) (i.e., *set the* 1*st column below the diagonal to zero*). This can be achieved by subtracting suitable multiples of the first row (equation) from the other rows (equations), namely

$$R_i \longleftarrow R_i - m_{i1} R_1 .$$

The above rule is to be applied to every element of the ith row. Thus we have,

$$\left.\begin{aligned} a_{ij}^{(2)} &= a_{ij}^{(1)} - m_{i1} a_{1j}^{(1)} \\ b_i^{(2)} &= b_i^{(1)} - m_{i1} b_1^{(1)} \end{aligned}\right\} \quad (j = 1, 2,\ 3, \ldots,\ n)$$

For $j = 1$, we have

$$a_{i1}^{(2)} = a_{i1}^{(1)} - m_{i1} a_{11}^{(1)}, \qquad i = 1, 2, \ldots, n.$$

Thus to set $a_{i1}^{(2)} = 0$, we only need to choose

$$m_{i1} = a_{i1}^{(1)} / a_{11}^{(1)} .$$

During the process, the first equation is called pivotal equation and its coefficient at the diagonal ($a_{11}^{(1)}$) is called the pivot. After this step, the augmented matrix becomes

$$\left[\begin{array}{cccc|c} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(1)} \\ 0 & \vdots & & \vdots & \vdots \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} & b_n^{(2)} \end{array}\right]$$

Step *k* After (k–1) steps of the elimination process, all the elements below the diagonal in columns 1 to (k–1) have been set to zero. In the kth step, assume that the k^{th} pivot $a_{kk}^{(k)} \neq 0$, we deal with column k to set the elements below the diagonal in this column to zero (i.e., eliminate the kth unknown from equations (k + 1) to n). This can be achieved by performing the following row operations for $i = k+1$ to n

$$R_i \longleftarrow R_i - m_{ik} R_k \quad \text{with} \quad m_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)} .$$

Applying the above row operation rule to every column of the ith row yields

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)}, \qquad b_i^{(k+1)} = b_i^{(k)} - m_{ik} b_k^{(k)}, \qquad (j = k\,,\, n) \tag{3.2}$$

Obviously from the above formulae, we have

$$a_{ik}^{(k+1)} = a_{ik}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kk}^{(k)} = 0\,,$$

and so the elements below the diagonal in column k will all be set to zero.

After (n –1) steps, the system becomes

$$\begin{bmatrix} a_{11}^{(1)} & \cdots & \cdots & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & \cdots & a_{2n}^{(2)} \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & a_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ \vdots \\ \vdots \\ \vdots \\ b_n^{(n)} \end{bmatrix}$$

or $\quad A^{(n)} x = b^{(n)} \quad$ or $\quad Ux = y.$

In summary, we have the following recurrence formulae for the elimination process

Recurrence formulae for the Gaussian elimination process

```
for $k$=1, 2 ,…, $n$–1
    for $i$=$k$+1,$n$
        $m_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)}$
        $b_i^{(k+1)} = b_i^{(k)} - m_{ik} b_k^{(k)}$
        for $j = k + 1,\ n$
            $a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)}$
```

Exercise. Write a F95/C++/Maple/Matlab program segment to implement the above elimination process. Store the multiples m_{ij} in the lower triangle of A.

Backward Substitution

The backward substitution solves the new equivalent system $Ux = y$, i.e

$$\begin{pmatrix} u_{11} & & \cdots & & u_{1n} \\ & \ddots & & & \vdots \\ & 0 & u_{kk} & \cdots & u_{kn} \\ \vdots & & & \ddots & \vdots \\ 0 & \cdots & & 0 & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_k \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_k \\ \vdots \\ y_n \end{pmatrix}$$

From the nth equation, we have $$u_{nn}x_n = y_n . \qquad (3.3)$$

From the kth equation, we have $$u_{kk}x_k + \sum_{j=k+1}^{n} u_{kj}x_j = y_k . \qquad (3.4)$$

Thus from (3.3) and (3.4), we have the following recurrence formulae

Recurrence formulae for the backward substitution

$$x_n = \frac{y_n}{u_{nn}}$$

$$x_k = \frac{1}{u_{kk}}[y_k - \sum_{j=k+1}^{n} u_{kj}x_j], \quad (k = n-1, n-2, \ldots\ldots 1).$$

Exercise. Write a F95/C++/Maple/Matlab program segment to implement the above backward substitution process.

Operation Count

The number of operations in each step and consequently the total number of operations required are summarized in the following tables

	Elimination for U			forward subs. for y		backward subs. for x		
Step	$\pm$	$\times$	$\div$	$\times$	$\pm$	$\div$	$\times$	$\pm$
1	$(n-1)^2$	$(n-1)^2$	$n-1$	$n-1$	$n-1$	1	0	0
2	$(n-2)^2$	$(n-2)^2$	$n-2$	$n-2$	$n-2$	1	1	1
k	$(n-k)^2$	$(n-k)^2$	$n-k$	$n-k$	$n-k$	1	$k-1$	$k-1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n-1$	1	1	1	1	1	1	$n-2$	$n-2$
n	0	0	0	0	0	1	$n-1$	$n-1$
Total	$\sum_{i=1}^{n-1} i^2$	$\sum_{i=1}^{n-1} i^2$	$\sum_{i=1}^{n-1} i$	$\sum_{i=1}^{n-1} i$	$\sum_{i=1}^{n-1} i$	n	$\sum_{i=1}^{n-1} i$	$\sum_{i=1}^{n-1} i$

Therefore, for the elimination process

the number of $*/\div$: $\sum_{i=1}^{n-1} i^2 + \sum_{i=1}^{n-1} i = \frac{n(n-1)(2n-1)}{6} + \frac{n(n-1)}{2} = \frac{n^3}{3} - \frac{n}{2}$,

the number of $+/-$: $\sum_{i=1}^{n-1} i^2 = \frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$;

while for the substitution process

the number of $*/\div$: $2\sum_{i=1}^{n-1} i + n = n^2$,

the number of $+/-$: $2\sum_{i=1}^{n-1} i = n^2 - n$.

So if n is large, the elimination process requires about $n^3/3$ operations of $*/\div$,
the substitution process requires about n^2 operations of $*/\div$.

The elimination process described in this section includes computations of the upper triangular matrix $U(A^{(n)})$ and the right hand side vector $y(b^{(n)})$. It will be shown in section 3 that the determination of y is in fact through a forward substitution process, namely solving $Ly = b$ where L is a lower triangular matrix.

Example 3.1 Solve $\begin{cases} x_1 + 2x_2 + x_3 = 0 \\ 2x_1 + 2x_2 + 3x_3 = 3 \\ -x_1 - 3x_2 = 2 \end{cases}$ using Gaussian elimination method.

Solution

$$[A|b] = \left[\begin{array}{ccc|c} 1 & 2 & 1 & 0 \\ 2 & 2 & 3 & 3 \\ -1 & -3 & 0 & 2 \end{array}\right] = \xrightarrow[\substack{R_2 \leftarrow R_2 - 2R_1 \\ R_3 \leftarrow R_3 + R_2}]{\substack{m_{21}=a_{21}/a_{11}=2/1=2 \\ m_{31}=a_{31}/a_{11}=-1/1=-1}} \left[\begin{array}{ccc|c} 1 & 2 & 1 & 0 \\ 0 & -2 & 1 & 3 \\ 0 & -1 & 1 & 2 \end{array}\right]$$

$$= \xrightarrow[R_3 \leftarrow R_3 - \frac{1}{2}R_2]{m_{32}=a_{32}/a_{22}=-1/(-2)=1/2} \left[\begin{array}{ccc|c} 1 & 2 & 1 & 0 \\ 0 & -2 & 1 & 3 \\ 0 & 0 & 1/2 & 1/2 \end{array}\right]$$

$$\therefore \quad x_3 = 1$$
$$-2x_2 + x_3 = 3 \quad \Rightarrow \quad x_2 = \tfrac{3-x_3}{-2} = \tfrac{3-1}{-2} = -1$$
$$x_1 + 2x_2 + x_3 = 0 \quad \Rightarrow \quad x_1 = -2x_2 - x_3 = -2(-1) - 1 = 1 .$$

3.3 LU Factorization and Its Connection with Gaussian Elimination

Theorem 3.1 (Factorization Theorem)

If the Gaussian elimination procedure can be performed on the linear system $Ax=b$ without row interchanges, then A can be factored into the product of a lower triangular matrix L and an upper triangular U, i.e., $A = LU$.

Proof (Hint). To prove $A=LU$, let

$$L = \begin{bmatrix} 1 & & & \text{O} \\ m_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ m_{n1} & m_{n2} & \cdots & 1 \end{bmatrix}, \quad U = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ & & \ddots & \vdots \\ \text{O} & & & a_{nn}^{(n)} \end{bmatrix}$$

where m_{ij} and $a_{ij}^{(k)}$ are as defined in section 3. 2. Then show that $(LU)_{ij} = a_{ij}$

Corollary:

$$\det A \;=\; a_{11}^{(1)} a_{22}^{(2)} \ldots a_{nn}^{(n)}\,.$$

Proof $\det A = \det L \det U$.
Since L and U are triangular, their determinants are the product of their diagonal elements.

Exercise. Write a F95/C++/Maple/Matlab program segment to read a matrix A and then find its LU factorization using the Gaussian elimination.

LU Method and Its Connection with the Gaussian Elimination Method

If $A = LU$, then $Ax = b$ becomes $LUx = b$.
Put $Ux = y$, then $Ly = b$.

Thus, the procedure for solving $Ax = b$ by the LU method is:

(1) Compute L, U (by Gaussian elimination process)

(2) Solve $Ly = b$ for y (by forward substitution to yield $y_1 \to y_2 \to \ldots \to y_n$)

(3) Solve $Ux = y$ for x (by backward substitution to yield $x_n \to x_{n-1} \to \ldots \to x_1$)

Advantages of *LU* Method

- More economic if we need to solve many systems with the same coefficient matrix *A* but different right hand side, as we only need to evaluate *L* and *U* for one time. Once *L* and *U* are saved, only the forward and backward substitutions are needed to solve each system.
- Storage space may be economized. If *A* is not required after factorization, we can store *L* and *U* in *A*.

Form of LU Factorization

From the result of Gaussian elimination process, we can obtain one form of *LU* factorization. Is it the unique form of the factorization? No ! Why ?

Consider the defining equations for *L* and *U*:

$$\begin{bmatrix} a_{11} & \cdots & \cdots & a_{1n} \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & & & O \\ \vdots & \ddots & & \\ \vdots & & \ddots & \\ l_{n1} & \cdots & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & \cdots & \cdots & u_{1n} \\ & \ddots & & \vdots \\ & & \ddots & \vdots \\ O & & & u_{nn} \end{bmatrix}.$$

As we have n^2 equations in terms of $n(n+1)$ unknowns (both *L* and *U* have $n(n+1)/2$ non-zero elements), the form of *LU* is not unique. To have a unique solution, we need *n* more additional equations on the entries of *L* and *U*. Different choices of additional equations will lead to different forms of *LU* factorization.

eg.	$l_{ii} = 1 \quad (i=1, n)$	$\rightarrow$	Doolittle's method
	$U_{ii} = 1$	$\rightarrow$	Crout's method
	$l_{ii} = U_{ii}$	$\rightarrow$	Choleski's method

Compact Scheme for LU Decomposition

As an example, we consider here the **Crout's Method.** The defining equations for the elements of *L* and *U* for this scheme are as follows

$$\begin{pmatrix} l_{11} & & & & & 0 & \\ \vdots & \ddots & & & & & \\ l_{k1} & & l_{kk} & & & & \\ \vdots & & \vdots & \ddots & & & \\ l_{i1} & \cdots & l_{ik} & & l_{ii} & & \\ \vdots & & \vdots & & & \ddots & \\ l_{n1} & & l_{nk} & & \cdots & & l_{nn} \end{pmatrix} \begin{pmatrix} 1 & \cdots & u_{1k} & \cdots & u_{1j} & \cdots & u_{1n} \\ & \ddots & \vdots & & & & \\ & & 1(u_{kk}) & \cdots & u_{kj} & \cdots & u_{kn} \\ & & & \ddots & & & \vdots \\ & & & & 1(u_{jj}) & & \\ & 0 & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} = \begin{pmatrix} a_{11} & \ddots & & \cdots & & a_{1n} \\ & & a_{kk} & \cdots & \boxed{a_{kj}} & \\ \vdots & & \vdots & & & \vdots \\ & & \boxed{a_{ik}} & & \ddots & \\ a_{n1} & & & \cdots & & a_{nn} \end{pmatrix}$$

The above system consists of $n \times n$ equations in terms of $n \times n$ unknowns including $n(n+1)/2$ non-zero entries of *L* and $n(n-1)/2$ non-zero entries of *U*, and hence can be used to completely determine *L* and *U*.

To determine the L and U, we multiply rows of L with columns of U and each multiplication will result in one equation. If we calculate the elements of L and U one by one in a systematic order, namely column 1 of L and row 1 of U, and then column 2 of L and row 2 of U and so on, then each multiplication will result in one equation involving only one new unknown, and so recurrence formulae can be derived for calculating l_{ij} and u_{ij}, as detailed below.

Assume that the first (k-1) columns of L and (k-1) rows of U have been determined, then

- multiplying the ith row of L with the kth column of U yields

  $$\begin{pmatrix} l_{i1} & l_{i2} & \cdots & l_{i(k-1)} & l_{ik} & \cdots & l_{ii} & 0\cdots \end{pmatrix} \cdot \begin{pmatrix} u_{1k} & u_{2k} & \cdots & u_{(k-1)k} & 1 & 0 & \cdots & \end{pmatrix} = a_{ik}$$

  that gives

  $$\sum_{p=1}^{k-1} l_{ip}u_{pk} + l_{ik} = a_{ik}$$

  and so we obtain the recurrence formulae for calculating the unknown elements in column k of L:

  $$l_{ik} = a_{ik} - \sum_{p=1}^{k-1} l_{ip}u_{pk} \qquad (i = k \text{ to } n);$$

- multiplying the kth row of L with the jth column of U yields

  $$\begin{pmatrix} l_{k1} & l_{k2} & \cdots & l_{kk} & 0 & \cdots \end{pmatrix} \cdot \begin{pmatrix} u_{1j} & u_{2j} & \cdots & u_{kj} & u_{(k+1)j} & \cdots \end{pmatrix} = a_{kj}$$

  that gives

  $$\sum_{p=1}^{k-1} l_{kp}u_{pj} + l_{kk}u_{kj} = a_{kj}$$

  and so we obtain the recurrence formulae for calculating the unknown elements in row k of U:

  $$u_{kj} = \frac{1}{l_{kk}}\left(a_{kj} - \sum_{p=1}^{k-1} l_{kp}u_{pj}\right) \qquad (j = k\text{+1 to } n).$$

In summary, we have the following compact scheme for the LU factorization.

Compact Scheme for *LU* Factorization (Crout's method)

for k=1, 2,..., n :

$$l_{ik} = a_{ik} - \sum_{p=1}^{k-1} l_{ip}u_{pk} \qquad (i = k \text{ to } n)$$

$$u_{kj} = \frac{1}{l_{kk}}\left(a_{kj} - \sum_{p=1}^{k-1} l_{kp}u_{pj}\right) \qquad (j = k+1 \text{ to } n)$$

Example 3. 2 Solve $\begin{bmatrix} 3 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 5 \\ -1 \end{bmatrix}$ using the *LU* factorization method.

Solution

LU factorisation

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Calculate 1st column of L (ith row of L * 1st column of U $\Rightarrow l_{i1}$)

$$l_{11} = 3, \quad l_{21} = 0, \quad l_{31} = 1$$

1st row of U (1st row of L * jth column of U $\Rightarrow u_{1j}$)

$$l_{11}\, u_{12} = 0, \qquad \Rightarrow \qquad u_{12} = 0$$
$$l_{11}\, u_{13} = 1, \qquad \Rightarrow \qquad u_{13} = 1/3$$

2nd column of L (ith row of L * 2nd column of U $\Rightarrow l_{i2}$)

$$l_{21}\, u_{12} + l_{22} = 2, \qquad \Rightarrow \qquad l_{22} = 2 - 0 = 2\,,$$
$$l_{31}\, u_{12} + l_{32} = 0, \qquad \Rightarrow \qquad l_{32} = 1(0) = 0$$

2nd row of U

$$l_{21}\, u_{13} + l_{22}\, u_{23} = 1 \qquad \Rightarrow \qquad u_{23} = (1 - 0*1/3)/2 = 1/2$$

3rd column of L

$$l_{31}u_{13} + l_{32}u_{23} + l_{33} = 0 \quad \Rightarrow \qquad l_{33} = -1 * 1/3 - 0 = -1/3$$

Therefore, the linear system becomes $LU\mathbf{x} = b$ as follows

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & -1/3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1/3 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 5 \\ -1 \end{bmatrix}.$$

Substitution

Solve $Ly = b$

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & -1/3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 5 \\ -1 \end{bmatrix}$$

$$\Rightarrow \quad y_1 = -2/3, \; y_2 = 5/2\,, y_3 = -3(-1 + 2/3) = 1$$

Solve $Ux = y$

$$\begin{bmatrix} 1 & 0 & 1/3 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2/3 \\ 5/2 \\ +1 \end{bmatrix}$$

$$\Rightarrow \quad x_3 = +1, \quad x_2 = 5/2 - 1/2 = 2, \quad x_1 = -2/3 - 1/3 = -1$$

Therefore,

$$x = \begin{bmatrix} -1 \\ 2 \\ +1 \end{bmatrix}.$$

Exercise. Solve the linear system using Doolittle's method.

3.4 Pivoting and Scaling

Pivoting

At each step (say k) of the Gaussian elimination process, we need to use a multiplier

$$m_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)}.$$

If $a_{kk}^{(k)}$ is small in magnitude compared to $a_{ik}^{(k)}$, m_{ik} will have magnitude much larger than one and thus

- when computing $a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik}a_{kj}^{(k)}$ $(j = k+1, n)$, a rounding error introduced in the computation of one of the terms $a_{kj}^{(k)}$ will be multiplied by m_{ik} compounding the original error.

- When performing the backward substitution for the solution x_k, any rounding error in the numerator will be amplified when dividing by $a_{kk}^{(k)}$.

To avoid the above problem, pivoting is performed by selecting a larger element for the pivot.

Partial (maximum column) Pivoting

In the Gaussian elimination process at stage k, determine the smallest $p > k$, such that

$$\left|a_{pk}^{(k)}\right| = \max_{k \le i \le n} \left|a_{ik}^{(k)}\right|$$

and perform $\left(E_k \leftrightarrow E_p\right)$ where $E_k =$ the kth equation, then proceed with step k of the elimination process. Thus, all of the multipliers m_{ik} will now satisfy $\left|m_{ik}\right| \le 1$.

Exercise. Write a F95/C++/Maple/Matlab program segment to implement the elimination process with maximum column pivoting.

Example 3.3 Solving $\begin{bmatrix} 0.003 & 59.11 \\ 5.291 & -6.130 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 59.17 \\ 46.78 \end{bmatrix}$ using Gaussian elimination with partial pivoting (with 4 digit arithmetic).

Solution

$E_1 \leftrightarrow E_2$:

$$\left[\begin{array}{cc|c} 5.291 & -6.130 & 46.78 \\ 0.003 & 59.14 & 59.17 \end{array}\right] \xrightarrow[R_2 \leftarrow R_2 - 0.000567 R_1]{m_{12}=0.003/5.291=0.000567} \left[\begin{array}{cc|c} 5.291 & -6.130 & 46.78 \\ 0 & 59.14 & 59.14 \end{array}\right]$$

Therefore, $\begin{cases} x_1 = 10.00 \\ x_2 = 1.000 \end{cases}$

Check. Exact solution $x_1 = 10,\ x_2 = 1$
Gaussian elimination without pivoting $x_1 = -10,\ x_2 = 1.001$

Complete Pivoting

At stage k, determine the smallest $p, q > k$, such that

$$\left|a_{pq}^{(k)}\right| = \max_{k \le i, j \le n} \left|a_{ij}^{(k)}\right|$$

and perform $\left(E_k \leftrightarrow E_p,\ C_k \leftrightarrow C_q\right)$, then proceed with step k of the elimination process.

Scaling and Scaled-column Pivoting

If the elements of A vary greatly in size, the pivoting strategy described above may fail. To deal with this problem, two methods may be used.

a) **Scaling matrix A** so that the elements vary less, usually by multiplying the rows and columns by suitable constants. This process will generally change the choice of pivot elements when pivoting is used with Gaussian elimination.

b) **Scaled - Column Pivoting Technique**

The first step in this procedure is to define for each row a scale factor s_i by

$$s_i = \max_{1 \le j \le n} \left|a_{ij}\right|$$

If for some i we have $s_i = 0$, then the system has no unique solution, since all entries in the ith row are zero. If all s_i are not equal to zero, we continue the Gaussian elimination process using matrix A. But we choose the pivot element in step k by determining the smaller $p > k$, such that

$$\frac{\left|a_{pk}^{(k)}\right|}{s_p} = \max_{k\le i\le n} \frac{\left|a_{ik}^{(k)}\right|}{s_i}$$

replacing the definition in partial pivoting. This process is to select the pivotal equation from the available ($n - k$) candidates as the one that has the absolutely largest coefficient of x_k relative to the size of the equation.

Example 3. 4 Solve $\begin{bmatrix} 30.00 & 591400 \\ 5.291 & -6.130 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 591700 \\ 46.78 \end{bmatrix}$

using scaled column pivoting (the system is obtained by multiplying the 1st equation in example 3. 3 by 10^4)

Solution

$$s_1 = \max(|30|, |591400|) = 591400, \qquad s_2 = 6.130$$

$$\frac{|a_{11}|}{s_1} = \frac{30}{591400} = 0.5073*10^{-4}, \qquad \frac{|a_{21}|}{s_2} = 0.863$$

$$\frac{|a_{11}|}{s_1} < \frac{|a_{21}|}{s_2}$$

Therefore, $E_1 \leftrightarrow E_2$

$$\left[\begin{array}{cc|c} 5.291 & -6.130 & 48.78 \\ 30.00 & 591410 & 591700 \end{array}\right] \quad \Rightarrow \quad x = \begin{pmatrix} 10 \\ 1 \end{pmatrix},$$

while the result obtained by Gaussian elimination with partial pivoting is $x = \begin{pmatrix} -10 \\ 1 \end{pmatrix}$.

3.5 Permuted LU Factorization

If the matrix A is such that a linear system $Ax = b$ can be solved using Gaussian elimination that does not require row interchanges, then there exists a direct LU factorization of A and the system can be solved by the LU method presented in 3.3. In the following, we will show that if row interchanges are required to control the rounding error resulting from the use of finite-digit arithmetic, there also exists a LU method, namely the permuted LU factorization method corresponding to the Gaussian elimination with pivoting.

We begin the discussion with the introduction of a class of matrices that are used to rearrange, or permute, rows of a given matrix.

Permutation Matrices

A permutation matrix P is a square matrix having the same form as the identity matrix except that the order of the rows is different:

$$P = \begin{Bmatrix} \mathbf{e}_{k1} \\ \mathbf{e}_{k2} \\ \vdots \\ \mathbf{e}_{kn} \end{Bmatrix} \tag{3.5}$$

where $\mathbf{e}_{ki}$ denotes the $k$ith row of the $n \times n$ identity matrix.

For example, if $k1 = 2$, $k2 = 3$, $k3 = 1$, then

$$P = \begin{Bmatrix} \mathbf{e}_2 \\ \mathbf{e}_3 \\ \mathbf{e}_1 \end{Bmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Property of Permutation Matrix

(i) A permutation matrix is an orthogonal matrix, i.e., $A^T A = I$, and the inverse of the permutation matrix is the same as its transpose ($A^{-1} = A^T$).

(ii) Left multiplying a matrix by a permutation matrix P has the effect of interchanging (permuting) the rows of the matrix. More specifically, let k_1, k_2, ……, k_n be a permutation of the integers 1, 2,……, n, and define P as in (3.5), then

$$PA = \begin{bmatrix} a_{k_1,1} & a_{k_1,2} & \cdots & a_{k_1,n} \\ a_{k_2,1} & a_{k_2,2} & \cdots & a_{k_2,n} \\ \vdots & \vdots & & \vdots \\ a_{k_n,1} & a_{k_n,2} & \cdots & a_{k_n,n} \end{bmatrix} = \{a_{k_i,j}\}$$

i.e, the ith row of the new system is the k_ith row of the original system.

Example 3.5 Let $P = \begin{Bmatrix} \mathbf{e}_1 \\ \mathbf{e}_3 \\ \mathbf{e}_2 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & 1 & 0 \end{bmatrix}$, $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$.

Then $PA = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$.

That is, left multiplying A by P has the effect of interchange row 2 and row 3.

Maximum Column Pivoting & Permuted *LU* Factorization

Application of the elimination phase of the Gaussian elimination with maximum column pivoting is equivalent to the LU factorization of a permuted version of the coefficient matrix. Thus the Gaussian elimination with maximum column pivoting algorithm can be used to find the permuted LU factorization.

Given $Ax = b$, by premultiplying P, we have

$$PAx = Pb.$$

Let $PA = LU$, then

$$LUx = Pb \qquad \Rightarrow \qquad \begin{cases} Ly = Pb, \\ Ux = y. \end{cases}$$

Suppose after row interchanges, the order of equations to be processed is $k1, k2, \ldots, kn$ where ki denotes the kith equation of the original system, then

$$P = \begin{Bmatrix} \mathbf{e}_{k1} \\ \mathbf{e}_{k2} \\ \vdots \\ \mathbf{e}_{kn} \end{Bmatrix} \text{ and } Pb = \begin{Bmatrix} b_{k1} \\ b_{k2} \\ \vdots \\ b_{kn} \end{Bmatrix}.$$

Thus to determine Pb in solving $Ly = Pb$, we only need to create an array (permutation vector) p(1:n) to store the values $k1, k2, \ldots, kn$ and hence

$$Pb = \begin{Bmatrix} b_{k1} \\ b_{k2} \\ \vdots \\ b_{kn} \end{Bmatrix} = \begin{Bmatrix} b(p(1)) \\ b(p(2)) \\ \vdots \\ b(p(n)) \end{Bmatrix}.$$

Program Construction (Doolittle's method)

1) Obtain the permuted LU matrices by the Gaussian elimination with maximum column pivoting. A permutation vector p is also produced to indicate the order in which the original equations are to be processed.

2) Forward substitution $\quad Ly = Pb \quad \Rightarrow \quad y_i = b[p(i)] - \sum_{j=1}^{i-1} l_{ij} y_j$

3) Backward substitution $\quad Ux = y \quad \Rightarrow \quad x_i = \dfrac{1}{u_{ii}}\left(y_i - \sum_{j=i+1}^{n} u_{ij} y_j \right)$

Notes: Numerical algorithms for the above permuted LU factorization and substitutions are given in Exercise Q3.6.

3.6 Special Types of Matrices

For strictly diagonally dominant and positive definite matrices, Gaussian elimination can be performed without row interchanges.

Strictly Diagonally Dominant Matrix (S.D.D)

Definition: A is (strictly) diagonally dominant iff $|a_{ii}| \underset{(>)}{\ge} \sum_{j \ne i} |a_{ij}|$ for all i.

Theorem 3.2: If A is a strictly diagonally dominant matrix, then A is nonsingular.

Theorem 3.3: If A is diagonally dominant and nonsingular, then Gaussian elimination can be performed on any linear system $Ax = b$ to obtain its unique solution without row or column interchanges.

Positive Definite Matrices

Definition: A symmetric n x n matrix A is positive definite iff $x^tAx > 0$ for every $n-D$ column vector $x \neq 0$.

Properties: If A is an $n \times n$ positive definite matrix, then
(a) A is nonsingular
(b) $a_{ii} > 0$ for all i
(c) A is symmetric, $A = A^t$

Theorem 3.4: The n x n symmetric matrix A is positive definite iff Gaussian elimination without row interchanges can be performed on the linear system $Ax = b$ with all pivot elements positive. Moreover, the computations are stable with respect to the growth of rounding error.

The Cholesky Factorization and Solution of Ax = b.

If A is symmetric,

$$A = LU = A^t = U^t L^t \quad \Rightarrow \quad L = U^t,\ U = L^t,\ l_{ii} = u_{ii}.$$

Thus, If A is positive definite, A can be factorized in the form of LL^t where L is a lower triangular matrix with nonzero diagonal entries.

Formulation

(a) LL^t **factorization**

From A $= LL^t$ for $i = 1, 2, \ldots, n$ and $j \leq i$, by multiplying the ith row of L and the jth column of L^t, we have

$$a_{ij} = \sum_{k=1}^{j} l_{ik} l_{jk} = \sum_{k=1}^{j-1} l_{ik} l_{jk} + l_{ij} l_{jj} \quad . \tag{3.6}$$

From the above, for i=j=1, we have

$$l_{11} = \sqrt{a_{11}}\,.$$

Now suppose row 1, row 2,…, row i-1 of L have been determined, we can derive, from (3.6), the following recurrence formulae to determine the ith row of L.

$$l_{ij} = \frac{1}{l_{jj}} \left[a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right], \qquad \text{for } j\text{=1 to } i\text{-1},$$

$$l_{ii}^{\,2} = \ a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 \,.$$

b) Solving $Ax= b$

As $A = LL^t$, $Ax = b \rightarrow LL^t x = b \rightarrow \begin{cases} Ly = b \\ L^t x = y \end{cases}$

To solve $Ly=b$, multiplying the ith row of L with y yields

$$(l_{i1}, l_{i2}, \ldots, l_{i,i-1}, l_{ii}, 0, \ldots, 0)\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{pmatrix} = b_i .$$

Therefore, $y_i = \dfrac{1}{l_{ii}}\left[b_i - \sum_{j=1}^{i-1} l_{ij} y_j \right]$ $(i = 1,\ 2, \ldots,\ n)$.

Remark: From the above, we can determine y_1, then $y_2, \ldots, y_n$

To solve $L^t x = y$, multiplying the ith row of L^t with x yields

$$(0, \ldots, 0, l_{ii}, l_{(i+1)i}, \ldots, l_{ni})\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} = y_i .$$

Therefore, $x_i = \dfrac{1}{l_{ii}}\left[y_i - \sum_{j=i+1}^{n} l_{ji} x_j \right]$ $(i = n,\ n-1, \ldots,\ 1)$.

Remark: From the above, we can determine x_n , then $x_{n-1}, \ldots, x_1$.

Algorithm (exercises)

Operation Count

Number of $* / \div$ operations: $\frac{n^3}{6} + \frac{n^2}{2} - \frac{2}{3}n$;

Number of $+ / -$ operations: $\frac{n^3}{6} - \frac{n}{6}$;

Number of $\sqrt{\ }$ operations: n.

LDL^t Decomposition and Solution of Ax = b

- The square roots in the LL^t decomposition can be avoided by using a slight modification, i.e., find a diagonal matrix D and a new lower triangular matrix L with one's on the diagonal such that $A = LDL^t$.
- This method applies for not only the positive definite matrices but also certain symmetric matrices.

Formulation

(a) ***LDL^t* decomposition**

From A = LDL^t , for i = 1, 2, …, n and $j \le i$,

$$a_{ij} = \sum_{k=1}^{j} l_{ik} d_k l_{jk} = \sum_{k=1}^{j-1} l_{ik} d_k l_{jk} + d_j l_{ij} l_{jj} \ ,$$

thus we have for

$$i = 1 \ , \ j = 1, \qquad \rightarrow \quad d_1 = a_{11}$$

$$i = 2, 3, \ldots, n \quad \begin{cases} j = 1, \ 2, \ldots, \ i-1 & \rightarrow \quad l_{ij} = \dfrac{1}{d_j}\left[a_{ij} - \displaystyle\sum_{k=1}^{j-1} l_{ik} d_k l_{jk}\right] \\ j = i & \rightarrow \quad d_i = \left[a_{ii} - \displaystyle\sum_{k=1}^{i-1} d_k l_{il}^2\right] \end{cases}$$

b) Solving $Ax = b$

As $A = LDL^t, \quad Ax = b \quad \rightarrow \quad LDL^t x = b \quad \rightarrow \quad \begin{cases} Ly = b \\ L^t x = D^{-1} y \end{cases}$

Hence, we can derive

$$y_i = b_i - \sum_{j=1}^{i-1} l_{ij} y_j \qquad (i = 1, \ 2, \ldots, \ n)$$

$$x_i = \frac{y_i}{d_i} - \sum_{j=i+1}^{n} l_{ji} x_j \qquad (i = n, \ n-1, \ldots, \ 1)$$

Algorithm_ (exercise)

Operation Count

Number of * / ÷ operations: $\dfrac{n^3}{6} + n^2 - \dfrac{7}{6} n$; number of + / – operations: $\dfrac{n^3}{6} - \dfrac{n}{6}$.

Tridiagonal Matrix (Band matrix, band width = 3) and Tridiagonal Systems of equations

Definition: $A_T = \left[a_{ij}\right]$ is tridiagonal if $a_{ij} = 0$ for $|i-j| > 1$.

Factorization: Provided no row interchanges are required for A_T, we can factor A_T as

$$\begin{bmatrix} d_1 & c_1 & & & & \\ a_2 & d_2 & c_2 & & \text{O} & \\ & a_3 & d_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & \text{O} & & a_{n-1} & d_{n-1} & c_{n-1} \\ & & & & a_n & d_n \end{bmatrix} =$$

$$\begin{bmatrix} \alpha_1 & & & & & \\ a_2 & \alpha_2 & & & \text{O} & \\ & a_3 & \alpha_3 & & & \\ & & \ddots & \ddots & & \\ & \text{O} & & a_{n-1} & \alpha_{n-1} & \\ & & & & a_n & \alpha_n \end{bmatrix} \begin{bmatrix} 1 & \beta_1 & & & & \\ & 1 & \beta_2 & & \text{O} & \\ & & 1 & \beta_3 & & \\ & & & \ddots & \ddots & \\ & \text{O} & & & 1 & \beta_{n-1} \\ & & & & & 1 \end{bmatrix}$$

We multiply the L and U matrices to obtain a way to compute α_i and β_i recursively. From the above equations, we have

$$\begin{aligned} & d_1 = \alpha_1, \; c_1 = \alpha_1\beta_1 \\ & d_i = a_i\beta_{i-1} + \alpha_i \qquad (i = 2,\ 3, \ldots,\ n) \\ & c_i = \alpha_i\beta_i \qquad (i = 2,\ 3, \ldots,\ n-1) \end{aligned}$$

which give rise to the following recurrence formulae,

$$\begin{aligned} & \alpha_1 = d_1, \quad \beta_1 = c_1 / \alpha_1 \\ & \left.\begin{aligned} \alpha_i &= d_i - a_i\beta_{i-1} \\ \beta_i &= c_i / \alpha_i \end{aligned}\right\} \qquad (i = 2,\ 3, \ldots,\ n-1) \\ & \alpha_n = d_n - a_n\beta_{n-1} \end{aligned}$$

Substitution

$$A_T x = b \quad \Rightarrow \quad LUx = b$$

Forward substitution $\quad Ly = b \quad \Rightarrow \quad y_1 = b_1 / \alpha_1$

$$y_i = (b_i - a_i y_{i-1}) / \alpha_i \qquad (i = 2,\ 3, \ldots,\ n)$$

Backward substitution $\quad Ux = y \quad \Rightarrow \quad x_n = y_n$

$$x_i = y_i - \beta_i x_{i+1} \qquad (i = n-1,\ n-2, \ldots,\ 1)$$

3.7 Gauss-Jordan Elimination and Inverse of Square Matrices

Gauss-Jordan Elimination

This process is much the same as regular elimination including the possible use of pivoting and scaling. It differs in eliminating the unknowns in equations below the diagonal as well as above it. The Gauss-Jordan procedure converts $[A|b]$ to $[I|b^{(n)}]$, so that at the completion of the elimination, $x= b^{(n)}$. Solving $Ax= b$ by this technique requires about $n^3/2$ multiplications and divisions, this is about 50% more than the regular Gaussian elimination.

Matrix Inverse

Denote x_i the *ith* columnof A^{-1}, then $A^{-1} = \{x_1, x_2,..., x_n\}$
e_i the *ith* column of I, then $I = \{e_1, e_2,..., e_n\}$.

As $AA^{-1}= I$, we can obtain A^{-1} by solving the following n linear systems

$$Ax_i=e_i$$

which require $n^3/3 - n/3$ (* / ÷) operations for LU decomposition, $n*n^2$ (* / ÷) operations for substitution. So calculating A^{-1} requires about $4n^3/3$ (* / ÷) operations. If we utilize $A^{-1}=U^{-1}L^{-1}$ to calculate A^{-1} we only need n^3 operations.

Note

To choose a method for solving a linear system, we generally consider the following two points.

(a) Use special techniques for special types of matrices.

(b) For general matrices, use triangular factorization with scaled-column pivoting.

3.8 Solution of Linear Systems using Maple/MATLAB

Solution of Linear Systems using Maple

By the permuted LU factorization method, $Ax=b$ is solved (in Maple) by the following steps

(a) Factor A into PLU so that the system becomes $PLUx=b$
where P is a permutation matrix ;

(b) Solve $PLy=b$ (by letting $Ux=y$), i.e, solve $Ly = P^T b$ (as $P^{-1} \equiv P^T$);

(c) Solve $Ux=y$

(**Note**: in Maple, pivoting is done only when a leading entry is zero)

The following Maple functions can be used to perform the above work

(*P,L,U*):=LUDecomposition(*A*):	factor ***A*** into *PLU*	& store the results in *P,L* and *U;*
***y*:=ForwardSubstitute(*L, Ptb*) :**	solve $Ly = P^T b$	& store the solution in vector *y*
***x*:=BackwordSubstite (*U, y*) :**	solve *Ux=y*	& store the solution in vector *x*

As the above functions are in the package ***LinearAlgebra***, the following statement should be declared before using the functions.

```
> with(LinearAlgebra):
```

Example. Solve $\begin{bmatrix} 0 & 2 & 1 & 1 \\ 0 & 0 & 3 & 2 \\ 1 & -2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 7 \\ 1 \\ -6 \\ 7 \end{bmatrix}$

```
> with(LinearAlgebra):
>
> A:=<<0 | 2 | 1 | 1 >, < 0| 0 | 3 | 2 >, <1 | -2 | 1 | 0 > ,<2 | 1 | 0 | 1  >>:
> b:=<<7>, <1>, <-6>, <7>>:
> (P,L,U):=LUDecomposition(A, method=GaussianElimination, output=['P','L','U']):
> Ptb:=Transpose(P).b:
> y:=ForwardSubstitute(L,Ptb):
> x:=BackwardSubstitute(U,y);
```

which yields the solution $x := \begin{bmatrix} 1 \\ 3 \\ -1 \\ 2 \end{bmatrix}$

Notes: *The arguments "method=..." and "Output=..." are optional. By default, the method used is the Gaussian elimination with partial pivoting method and the outputs are P, L and U.*

Solving linear Systems using MATLAB

In MATLAB, a non-singular system *Ax=b* can be solved directly by

```
x=A\b
```

or by the ***permuted LU method*** via the following 3 steps

(a) Factor *PA* into *LU* by *[L,U,P]=lu(A)* so that the system becomes *LUx=Pb*

(b) Solve *Ly=Pb* by *y=L\(P*b)*

(c) Solve *Ux=y* by *x=U\y*

Example. Solve $\begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ -1 \\ 2 \end{bmatrix}$

```
>> A=[1  1  0;  0  0  1;  3  0  1];
>> b =[3;  -1; 2];
>> [L, U, P] =Lu(A);
>> y=L\(P*b);
>> x=U\y
```

which yields the solution $x = [1 \quad 2 \quad -1]^T$

EXERCISE 3

Q3. 1 Given $A = \begin{pmatrix} 1 & -1 & 0 \\ -2 & 4 & -2 \\ 0 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ -1 \\ 1.5 \end{pmatrix}.$

a) Solve Ax = b by Gaussian elimination without row interchanges
b) Construct the LU factorization of A from the results of Gaussian elimination process in (a).
c) Solve Ax = b by first factorizing A into LU with u_{ii}= 1.

Q3. 2 Solve the following linear systems using
a) Gaussian elimination (GE.), (2- digit rounding arithmetic) without pivoting,
b) GE. with maximal column pivoting (2- digits),
c) GE. with scaled-column pivoting (2- digits)
d) Exact arithmetic and determine which part, (a), (b), or (c) is the most accurate.

$$\begin{cases} 10^{-2}x + y = 1 \\ x + y = 2 \end{cases} \qquad \begin{cases} x_1 + 2x_2 + 3x_3 = 1 \\ 2x_1 + 3x_2 + 4x_3 = -1 \\ 3x_1 + 4x_2 + 6x_3 = 2 \end{cases}$$

Q3. 3 Using the compact scheme (l_{ii}= 1), factor matrix A, solve $Ax = b$ and calculate det A.

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \\ 1 & 16 & 81 & 256 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 10 \\ 44 \\ 190 \end{pmatrix}.$$

Q3. 4 Determine which of the following matrices are
(i) symmetric, (ii) singular, (iii) strictly diagonally dominant, (iv) positive definite.

a) $\begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$, (b) $\begin{bmatrix} 2 & 1 & 0 \\ 0 & 3 & 2 \\ 1 & 2 & -4 \end{bmatrix}$, (c) $\begin{bmatrix} -2 & 1 \\ 1 & -3 \end{bmatrix}$, (d) $\begin{bmatrix} -1 & 2 \\ 1 & 2 \end{bmatrix}$,

Q3. 5 Find a factorization of the form $A = LDL^T$ for matrix

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

Programming

Q3. 6 The attached subroutines *LUFACT* and *SUBST* are for solving $Ax = b$. The routine *LUFACT* computes a *LU* factorization of A with scaled-column pivoting $PA=LU$. The routine *SUBST* calculates x by forward and backward substitutions: $Ly= Pb$, $Ux= y$. Document and modify the program such that (1) all real values are stored with at least 12 digits of precision; (2) whole array operations are used whenever possible; (3) use assumed-shape arrays in procedures. Then write a main program which reads entries of A and b from a data file and calls the subroutine to solve the following system

3.3330x1 + 15920 x2 –10.333 x3 =15913
2.2220x1 + 16.710x2+9.6120 x3 = 28.544
1.5611x1 + 5.1791x2 +1.6852x3 = 8.4254

Algorithm for Subroutine LUFACT

This algorithm uses the Gaussian elimination process with scaled-column pivoting to find the permuted *LU* factorization of *A* (namely, to find *P* and the *LU* factorization of *PA*) where
U is the upper triangular matrix obtained from the elimination process;
L is the lower triangular matrix which is the collection of the multiples m_{ij}

```
Step 1 Set $s(i) = \underset{1\le j\le n}{Max}|a_{ij}|$, (determine the size of each equation).
Step 2 For $k = 1$ to $N-1$ do step 3 to step 6  (set 1st, ...., $(n-1)$th column below diagonal to zero)
Step 3      Find the (smaller) $P \ge k$ such that $\left|\frac{a_{pk}}{s_p}\right| = \max_{k\le i\le n}\left|\frac{a_{ik}}{s_i}\right|$
                    (select pivot element for the step)
Step 4      If $a_{pk} = 0$ then
                write '(IERR=1, $A$ is singular)' then return.
Step 5      Else
                $E_k \leftrightarrow E_p$ , $P_k \leftrightarrow P_p$ , (row interchange)
                    ($P_k$ records the order in which the equations are to be processed)
Step 6          For $i = k+1$ to $n$   (do usual Gauss elimination process for the $k$th step)
                    Set   $m_{ik} = \frac{a_{ik}}{a_{kk}}$      $(\Rightarrow a(I,k))$
                    Set   $a_{ij} = a_{ij} - m_{ik}a_{kj}$      $(j = k+1, ......, n)$
Step 7  If $a_{nn} = 0$, return "(IERR=1: $A$ is singular)'
Return
```

Algorithm for SUBST

```
For $i = 1$ to $n$ do
    $y_i = b(p_i) - \sum_{j=1}^{i-1} a_{ij} y_j$        Forward substitution    (as $Pb = b(\text{pi})$)
For $i = n$ to 1 by $-1$ do
    $x_i = \frac{1}{a_{ii}}\left(y_i - \sum_{j=i+1}^{n} a_{ij} x_j\right)$        Backward substitution
End
```

Q3.7 Based on the subroutines *LUFACT* and *SUBST*, write subroutines *LU*1 and *SUB*1, respectively for computing the *LU* factorization with partial pivoting (without scaling) and for finding solution of $Ax = b$ by forward and backward substitution. Then, write a main program to call the subroutines to solve the linear system in Q3. 6.

Q3.8 Based on subroutines *LUFACT* and *SUBST*, write subroutines *LU2* and *SUB2*, respectively for computing the *LU* factorization without pivoting and for finding solution of $Ax = b$ by forward and backward substitution. Then, write a main program to call the subroutines to solve the linear system in Q3. 6.

Q3.9 Write a subroutine for LDL^T factorization of a square matrix A, and a separate subroutine for solving equations $Ax= b$ by forward and backward substitutions using the LDL^T factorization. Then write a main program to call the subroutines to solve the following linear system.

$$\begin{aligned} 4x1 + x2 - x3 &= 7,\\ x1 + 3x2 - x3 &= 8,\\ -x1 - x2 + 5x3 + 2x4 &= -4,\\ 2x3 + 4x4 &= 6. \end{aligned}$$

Algorithm for factorizing an $n \times n$ matrix A into LDL^{T} decomposition

```
Input   $n$ and matrix $A$
Output  $L$
Set   $d_1 = a_{11}$
For  $i = 2$ to $N$ do
    set  $l_{ij} = \frac{1}{d_j}\left[a_{ij} - \sum_{k=1}^{j-1} l_{ik} d_k l_{jk}\right]$      $j = 1, 2, \ldots, i-1$
    set  $d_i = \left[a_{ii} - \sum_{k=1}^{i-1} d_k l_{ik}^2\right]$
Return
```

Q3.10 Write a subroutine STRIDEQ for solving Tridingonal linear systems, and then write a main program to call the subroutine to solve

$$\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} x = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \qquad \text{(Answer: } x = (1,\ 1,\ 1,\ 1)^T \text{)}$$

Subroutine header: **SUBROUTINE STRIDEQ (N, A, D, C, B, X)**

where **N**	=	Input. Number of equations.
A(N)=		Input. Before entry, must contain the sub-diagonal element of $A_T(a_i)$.
D(N)=		Input. Before entry, must contain the main diagonal element of $A_T(d_i)$.
C(N)=		Input. Before entry, must contain the super-diagonal element of $A_T(c_i)$.
B(N)=		Input. Before entry, must contain the element of the right hand side (b_i).
X(N)=		Output. On exit, contain the solution of the tridiagonal system.

Algorithm for solving Tridiagonal Systems $A_r x = b$

```
Set $\alpha_1 = d_1 \qquad \beta_1 = \frac{c_1}{\alpha_1}$
For $i = 2$ to $n-1$
    set $\alpha_i = d_i - a_i\beta_{i-1}$
        $\beta_i = \frac{c_i}{\alpha_i}$
Set $\alpha_n = d_n - a_n\beta_{n-1}$
    $y_1 = \frac{b_1}{\alpha_1}$
For $i = 2$ to $n$ do
    set $y_i = \frac{1}{\alpha_i}\left[b_i - a_i y_{i-1}\right]$
Set $x_n = y_n$
For $i = n-1$ to 1 by $-1$ do
    set $x_i = y_i - \beta_i x_{i+1}$
Output $(x_1, x_2, \ldots, x_n)$
```

Q3.11 Solve the linear systems in Q3.1 and Q3.6 using Maple.

Q3.12 Solve the linear systems in Q3.1 and Q3.6 using MATLAB.

CHAPTER 4

Norms and Error Analysis for Linear Systems

In this chapter, we first introduce the concepts of vector norms, matrix norms and spectral radius of matrices in sections 4.1-4.3. Then in section 4.4, we use the notions of vector norms and matrix norms as well as spectral radius to estimate the prior and posterior errors for the solutions of linear systems. Finally, in section 4.5, we introduce the residual correction method for solving linear systems utilizing the *LU* factorization method.

4.1 Norms of Vectors

Solving a system of linear equations using computer can only yield, because of round off errors, approximate solution $\tilde{\mathbf{x}}$. To analyse whether an approximate solution vector is close enough to the solution of the system, we need a means for measuring the distance between two *n*-dimensional vectors $\tilde{\mathbf{x}}$ and **x**, or in the other words, the size of an *n*-dimensional vector $\mathbf{e} = \tilde{\mathbf{x}} - \mathbf{x}$ where

$$\tilde{\mathbf{x}} = [\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n]^t, \quad \mathbf{x} = [x_1, x_2, \ldots, x_n]^t.$$

One of the ways, which we are familiar with, is to use the Euclidean norm

$$d = \|\tilde{\mathbf{x}} - \mathbf{x}\| = \sqrt{(\tilde{x}_1 - x_1)^2 + (\tilde{x}_2 - x_2)^2 + \cdots + (\tilde{x}_n - x_n)^2}$$

which represents the distance between two points $\tilde{\mathbf{x}}$ and $\mathbf{x}$ in $\mathbb{R}^n$. But there are many situations, in which it is more convenient to measure the size of a vector in other ways. Thus we introduce a general concept of the norm of a vector.

Definition of Vector Norms

Let $$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = [x_1,\ x_2,\ \ldots.,\ x_n]^t \qquad \in \mathbb{R}^n$$

A norm of a vector on $\mathbb{R}^n$ is a function, $\|\cdot\|$, from $\mathbb{R}^n$ into $\mathbb{R}$ with the following properties.

(i) $\|\mathbf{x}\| \ge 0$ for all $\mathbf{x} \in \mathbb{R}^n$

(ii) $\|\mathbf{x}\| = 0$ if and only if (iff) $\mathbf{x} = 0$

(iii) $\|\alpha\mathbf{x}\| = |\alpha|\,\|\mathbf{x}\|$ for all $\alpha \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^n$ (4.1)

(iv) $\|\mathbf{x} + \mathbf{y}\| \le \|\mathbf{x}\| + \|\mathbf{y}\|$ for all $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^n$ (Δ inequality)

Lebesgue Norms (l_p norms)

Let $x = (x_1, x_2, ..., x_n)^T \in \mathbb{R}^n$, then

$$\|\mathbf{x}\|_p = \left(\sum_{j=1}^{n} |x_i|^p \right)^{1/p}$$

defines a family of vector norms, called Lebesque norms or l_p norms.

For p=1, we have the sum (l_1) norm $\|\mathbf{x}\|_1 = \sum_{j=1}^{n} |x_i|$. (4.2)

For p =2, we have the Euclidean (l_2) norm $\|\mathbf{x}\|_2 = \left(\sum_{j=1}^{n} x_i^2 \right)^{1/2}$. (4.3)

For $p = \infty$, we have the Chebyshev (l_∞) norm $\|\mathbf{x}\|_\infty = \max_{1 \le i \le n} |x_i|$. (4.4)

Remarks To show that a particular function is a norm, one must prove that properties (i) to (iv) hold for the function. In the following, we give an example.

Example 4.1 Prove that $\|\cdot\|_\infty$ is a vector norm.

Proof Let $\mathbf{x} = [x_1, x_2, \cdots, x_n]^t$, $\mathbf{y} = [y_1, y_2, \cdots, y_n]^t$.

Then properties (i)-(iii) hold immediately and we now consider property (iv). From the definition of $\|\cdot\|_\infty$, we have

$$\begin{aligned} \|\mathbf{x} + \mathbf{y}\|_\infty &= \max_{1 \le i \le n} |x_\text{i} + y_\text{i}| \\ &\le \max_{1 \le i \le n} \{|x_\text{i}| + |y_\text{i}|\} \\ &\le \max_{1 \le i \le n} |x_\text{i}| + \max_{1 \le i \le n} |y_\text{i}| \\ &= \|\mathbf{x}\|_\infty + \|\mathbf{y}\|_\infty \ . \end{aligned}$$

Hence, property (iv) holds. □

Example 4.2 Given $\mathbf{x} = [-1,\ 2,\ 1]$. Find the l_p norms for p = 1, 2 and ∞ .

Solution

$$\|\mathbf{x}\|_1 = |-1| + |2| + |1| = 4$$

$$\|\mathbf{x}\|_2 = \left[(-1)^2 + 2^2 + 1^2 \right]^{1/2} = \sqrt{6}$$

$$\|\mathbf{x}\|_\infty = \max\left(|-1|, |2|, |1| \right) = 2 \ .$$

Exercise Show that $\|\mathbf{x}\|_2 = \left\{ \sum_{j=1}^{n} x_i^2 \right\}^{\frac{1}{2}}$ is a norm. The Cauchy- Buniakowsky- Schwarz inequality, as given below, could be useful.

$$\sum_{i=1}^{n} |x_i \, y_i| \le \left\{\sum_{i=1}^{n} x_i^2\right\}^{\frac{1}{2}} \left\{\sum_{i=1}^{n} y_i^2\right\}^{\frac{1}{2}} \tag{4.5}$$

for any $(x_1, x_1, \cdots x_n)^T, (y_1, y_1, \cdots y_n)^T \in \mathbb{R}^n$.

Distance Between Two Vectors

If $\mathbf{x}$ and $\mathbf{y}$ are two vectors, then the 'distance' between $\mathbf{x}$ and $\mathbf{y}$ is given by the vector norm $\|\mathbf{x}-\mathbf{y}\|$

Example 4.3 Given $\mathbf{x} = (-1, 2, 1)^T$, $\mathbf{y} = (1, 1, 3)^T$. Find the l_1, l_2 and l_∞ distances.

Solution l_1 distance: $\|\mathbf{x}-\mathbf{y}\|_1 = |1-1| + |-2-1| + |1-3| = 5$

l_2 distance: $\|\mathbf{x}-\mathbf{y}\|_2 = \sqrt{(1-1)^2 + (-2-1)^2 + (1-3)^2} = \sqrt{13}$

l_∞ distance: $\|\mathbf{x}-\mathbf{y}\|_\infty = max\{|1-1|, |-2-1|, |1-3|\} = 3$.

Sequence of Vectors and Convergence

Definition A sequence of vectors $\{\mathbf{x}^{(k)}\}_{k=1}^{\infty}$ in $\mathbb{R}^n$ is said to converge to $\mathbf{x}$ with respect to the norm $\|\cdot\|$ if, for any given $\varepsilon > 0$, there exists an integer $N(\varepsilon)$ such that $\|\mathbf{x}^{(k)} - \mathbf{x}\| < \varepsilon$ for all $k \ge N(\varepsilon)$.

Theorem 4.1 The sequence of vectors $\{\mathbf{x}^{(k)}\}_{k=1}^{\infty}$ converges to **x** in $\mathbb{R}^n$ with respect to $\|\cdot\|_\infty$ if and only if $\lim_{k\to\infty} x_i^{(k)} = x_i$ for each i =1, 2,…, *n*.

Proof

(1°) ($\Rightarrow$ Necessary condition : if $\{\mathbf{x}^{(k)}\}_{k=1}^{\infty}$ converges, $\lim_{k\to\infty} x_i^{(k)} = x_i$)

If $\mathbf{x}^{(k)} \to \mathbf{x}$ with respect to $\|\cdot\|_\infty$, then given $\varepsilon > 0$, $\exists$ an integer $N(\varepsilon)$ such that

$$\|\mathbf{x}^{(k)} - \mathbf{x}\|_\infty = \max_i |x_i^{(k)} - x_i| < \varepsilon \quad \text{for all} \quad k \ge N(\varepsilon),$$

which implies that

$$|x_i^{(k)} - x_i| < \varepsilon \text{ for each } i \text{ when } k \ge N(\varepsilon).$$

$\Rightarrow$ $\lim_{n\to\infty} x_i^{(k)} = x_i$ for each *i*.

(2°) ($\Leftarrow$ Sufficient condition : if $\lim_{n\to\infty} x_i^{(k)} = x_i$, $\{\mathbf{x}^{(k)}\}_{k=1}^{\infty}$ converges)

If $\lim_{k\to\infty} x_i^{(k)} = x_i$ for every *i*=1, 2, …, *n*, then for any given $\varepsilon > 0$, $\exists$ an integer $N_i(\varepsilon)$ such that $|x_i^{(k)} - x_i| < \varepsilon$ whenever $k \ge N_i(\varepsilon)$.

Define $N(\varepsilon) = \max_i \{N_i(\varepsilon)\}$, then we have

$$\max_i |x_i^{(k)} - x_i| < \varepsilon \qquad \text{for all } k \geq N(\varepsilon).$$

Thus, $\|\mathbf{x}^{(k)} - \mathbf{x}\|_\infty < \varepsilon$ for all $k \geq N(\varepsilon)$.

This implies that $\{\mathbf{x}^{(k)}\}_{k=1}^{\infty}$ converges to $\mathbf{x}$. □

Example 4.4 Let $\mathbf{x}^{(k)} \in \mathbb{R}^4$ be defined by $\mathbf{x}^{(k)} = \{x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, x_4^{(k)}\}^T = (1,\ 2+\frac{1}{k},\ \frac{3}{k^2},\ e^{-k}\sin k)^T$.

Find the vector $\mathbf{x}$ that the above sequence converges to.

Solution

$$\lim 1 = 1, \qquad \lim_{k\to\infty}\left(2+\frac{1}{k}\right) = 2$$

$$\lim_{k\to\infty}\left(\frac{3}{k^2}\right) = 0, \qquad \lim_{k\to\infty}\left(e^{-k}\sin k\right) = 0.$$

Therefore $\{\mathbf{x}^{(k)}\} \to (1,\ 2,\ 0,\ 0)^T$ with respect to $\|\cdot\|_\infty$.

Equivalence of Vector Norms with Respect to Convergence

The following theorem shows the equivalence of $\|\cdot\|_\infty$ and $\|\cdot\|_2$ with respect to convergence.

Theorem 4.2 For each $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \sqrt{n}\,\|\mathbf{x}\|_\infty$.

Proof let x_m be the element with maximum magnitude of the vector $\mathbf{x}$, then

$$\|\mathbf{x}\|_\infty = \max_{1\leq i\leq n} |x_i| = |x_m|$$

Hence, $\|\mathbf{x}\|_\infty^2 = |x_m|^2 = x_m^2 \leq \sum_{i=1}^{n} x_i^2 \leq n x_m^2 = n\|\mathbf{x}\|_\infty^2$

Thus $\|\mathbf{x}\|_\infty^2 \leq \|\mathbf{x}\|_2^2 \leq n\|\mathbf{x}\|_\infty^2$

then $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \sqrt{n}\,\|\mathbf{x}\|_\infty$. □

Remark 1 In general, all norms on $\mathbb{R}^n$ are equivalent with respect to convergence, i.e. if $\|\cdot\|$ and $\|.\|'$ are any two norms, then

$$\mathbf{x}^{(k)} \to \mathbf{x} \text{ with respect to } \|\cdot\| \iff \mathbf{x}^{(k)} \to \mathbf{x} \text{ with respect to } \|\cdot\|'.$$

Remark 2 (Application of the convergence equivalence theorem) Sometimes it is a lot easier to show convergence with respect to $\|\cdot\|_\infty$ than with respect to $\|\cdot\|_2$. But by the equivalence theorem, the former prove the later (and vice-versa).

Example 4.5 In example 4.4, we find that

$$\mathbf{x}^{(k)} = \left(1,\ 2+\frac{1}{k},\ \frac{3}{k},\ e^{-k}\sin k\right)^t \text{ converges to } \mathbf{x} = (1,\ 2,\ 0,\ 0)^t \text{ with respect to } \|\cdot\|_\infty .$$

So by the above theorem, we can conclude that $\mathbf{x}^{(k)}$ also converges to $\mathbf{x}$ with respect to $\|\cdot\|_2$.

4.2 Norms of Matrices

Definition of matrix norm

A ***matrix norm*** on the set of all n x n matrices is a real-valued function, denoted by $\|\cdot\|$ and defined on the set that satisfies for all n x n matrices A and B and real numbers α :

(i) $\|A\| \geq 0$

(ii) $\|A\| = 0$ iff $A = 0$

(iii) $\|\alpha A\| = |\alpha|\|A\|$

(iv) $\|A+B\| \leq \|A\| + \|B\|$

(v) $\|AB\| \leq \|A\|\,\|B\|$ (4.6)

Remark From (iv), $\|A-B\| \geq \|A\| - \|B\|$

Proof : $\|A\| = \|A-B+B\| \leq \|A-B\| + \|B\|$

$\Rightarrow \|A-B\| \geq \|A\| - \|B\|$.

Natural Matrix Norms

Matrix norms can be obtained in various ways. Here we present one of the methods based on the concept of vector norms.

Theorem 4.3 (definition of natural matrix norm) If $\|\cdot\|$ is any vector norm on $\mathbb{R}^n$, then

$$\|A\| = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|$$

defines a matrix norm on the set of real n x n matrices, and is called the natural matrix norm associated with the vector norm.

Remark Associated with the l_∞, l_2 and l_1 vector norms, we have the following l_∞ and l_2 matrix norms:

$$\|A\|_\infty = \max_{\|\mathbf{x}\|_\infty=1} \|A\mathbf{x}\|_\infty , \tag{4.7}$$

$$\|A\|_2 = \max_{\|\mathbf{x}\|_2=1} \|A\mathbf{x}\|_2 , \tag{4.8}$$

$$\|A\|_1 = \max_{\|\mathbf{x}\|_1=1} \|A\mathbf{x}\|_1 . \tag{4.9}$$

Corollary $\|A\|$ can also be calculated as

$$\|A\| = \max_{\mathbf{x}\neq\mathbf{0}} \left(\frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} \right) \tag{4.10}$$

and

$$\|A\mathbf{x}\| \le \|A\| \, \|\mathbf{x}\| . \tag{4.11}$$

Proof Suppose $\mathbf{x} \neq 0$, and let $\mathbf{y} = \dfrac{\mathbf{x}}{\|\mathbf{x}\|}$, then $\|\mathbf{y}\| = 1$.

Thus
$$\|A\| = \max_{\|\mathbf{y}\|=1} \|A\mathbf{y}\|$$
$$= \max_{\mathbf{x}\neq 0} \left\| A \frac{\mathbf{x}}{\|\mathbf{x}\|} \right\| = \max_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}$$
$$\Rightarrow \quad \|A\| \ge \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}$$
$$\Rightarrow \quad \|A\mathbf{x}\| \le \|A\| \|\mathbf{x}\| .$$
□

Evaluation of $\|A\|_\infty$ and $\|A\|_1$

The l_∞ norm of a matrix has an interesting representation with respect to the entries of the matrix.

Theorem 4.4 If $A = (a_{ij})$ is an $n \times n$ matrix, then

$$\|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}| \quad \text{(maximum absolute value row sum)}$$

Theorem 4.5 If $A = (a_{ij})$ is an $n \times n$ matrix, then

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} | a_{ij} | \quad \text{(maximum absolute value column sum)}$$

Proof of Theorem 4.4

Let $\mu = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|$ and $\mathbf{x}$ be such that $\|\mathbf{x}\|_\infty = 1$.

Then
$$\|A\mathbf{x}\|_\infty = \max_{1\le i\le n} |(A\mathbf{x})_i| = \max_i \left| \sum_{j=1}^{n} a_{ij} x_j \right|$$
$$\le \max_i \sum_{j=1}^{n} |a_{ij}| |x_j| \le \max_i \sum_{j=1}^{n} |a_{ij}| \left(\max_j |x_j| \right)$$
$$= \mu \|\mathbf{x}\|_\infty = \mu .$$

So $\|A\mathbf{x}\|_\infty \le \mu$ for all $\mathbf{x}$

$$\therefore \qquad \|A\|_\infty = \max_{\|\mathbf{x}\|_\infty=1} \|A\mathbf{x}\|_\infty \le \mu \tag{4.12}$$

On the other hand, suppose at row p

$$\sum_{j=1}^{n} |a_{pj}| = \mu \quad \text{(max row sum)}$$

and $\mathbf{x}$ is those with

$$x_j = \begin{cases} 1 \text{ if } a_{pj} \ge 0 \\ -1 \text{ if } a_{pj} < 0. \end{cases}$$

Then $\quad \|\mathbf{x}\|_\infty = 1$ and $a_{pj}x_j = |a_{pj}|$ for $j = 1, 2, \ldots, n.$

$$\|A\mathbf{x}\|_\infty = \max_i \left|\sum_{j=1}^{n} a_{ij}x_j\right| \ge \left|\sum_{j=1}^{n} a_{pj}x_j\right| = \sum_{j=1}^{n} |a_{pj}| = \mu$$

$$\|A\|_\infty = \max_{\|\mathbf{x}\|_\infty=1} \|A\mathbf{x}\|_\infty \ge \mu \tag{4.13}$$

Equations (4.12) and (4.13) prove the theorem. □

Exercise. Prove Theorem 4.5.

Example 4.5 Given $A = \begin{pmatrix} 1 & 2 & -1 \\ 0 & 3 & -1 \\ 6 & -1 & 1 \end{pmatrix}$, find $\|A\|_\infty$ and $\|A\|_1$.

Solution

$$\|A\|_\infty = \max\{|1| + |2| + |-1|,\, 0+3+1,\, 6+1+1\} = 8,$$
$$\|A\|_1 = \max\{1+0+6,\ 2+3+|-1|,\ |-1| + |-1| + 1\} = 7.$$

Example 4.6 Given $A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$, find $\|A\|_2$.

Solution

$$A\mathbf{x} = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2x_1 - x_2 \\ -x_1 + 2x_2 \end{bmatrix}$$

$$\therefore \|A\|_2 = \max_{\sqrt{x_1^2+x_2^2}=1} \left[\sqrt{(2x_1 - x_2)^2 + (-x_1 + 2x_2)^2}\right]$$
$$= \max_{x_1^2+x_2^2=1} \left[5(x_1^2 + x_2^2) - 8x_1x_2\right]^{\frac{1}{2}}$$
$$= \max_{x_1^2+x_2^2=1} (5 - 8x_1x_2)^{\frac{1}{2}}\ .$$

Let $F = 5 - 8x_1x_2$ then $\|A\|_2 = \sqrt{F_{\max}}$ subject to $x_1^2 + x_2^2 = 1$.

From the constraint, we have

$$x_2 = \pm\sqrt{1-x_1^2} \quad \text{and so} \quad F = 5 \pm 8x_1\sqrt{1-x_1^2}\,.$$

To find the maximum of F, let

$$\frac{\partial F}{\partial x_1} = \pm 8\left(x_1 \frac{-x_1}{\sqrt{1-x_1^2}} + \sqrt{1-x_1^2} \right) = 0$$

Hence,

$$x_1 = \sqrt{\frac{1}{2}},\ x_2 = \sqrt{\frac{1}{2}}$$

$$F_{\max} = 5 + 8\sqrt{\frac{1}{2}}\sqrt{\frac{1}{2}} = 9 \quad \therefore \quad \|A\|_2 = \sqrt{9} = 3.$$

Remark: The l_2 norm of a matrix can also be determined without referring directly to the definition. To develop the technique for the evaluation of l_2 norm, we firstly introduce the notions of eigenvalues, eigenvectors, and spectral radius of matrices in the following section.

4.3 Eigenvalues (λ_i) and Spectral Radius $\rho(A)$

Eigenvalue and Eigenvectors

Definition: Let A be an $n \times n$ real matrix. If there exists $\mathbf{x} \neq 0$ such that

$$A\mathbf{x} = \lambda\mathbf{x}\,, \tag{4.14}$$

then λ is called the eigenvalue of A and $\mathbf{x}$ is the associated eigenvector.

From (4.14), we have

$$(A - \lambda I)\mathbf{x} = \mathbf{0}\,. \tag{4.15}$$

For λ to be an eigenvalue, (4.15) must have non-trivial solution $\mathbf{x} \neq 0$, which requires

$$P(\mathbf{x}) = \det(A - \lambda I) = 0. \tag{4.16}$$

The left hand side of the above equation is called the characteristic polynomial of A, and the eigenvalues of A, λ, are the roots of the characteristic equation $P(\mathbf{x}) = 0$. Hence, to find the eigenvalues of A, we just need to solve the characteristic equation.

As λ is an eigenvalue of A, $\det(A - \lambda I) = 0$, and hence (4.15) has a non-zero solution, $\mathbf{x}$, which is called an eigenvector of A corresponding to the eigenvalue λ.

Example 4.6 Given $A = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{pmatrix}$, find all eigenvalues and all eigenvectors of A.

Solution

1° Characteristic polynomial:

$$P(\lambda) = \det(A - \lambda I) = \begin{vmatrix} 1-\lambda & 0 & 2 \\ 0 & 1-\lambda & -1 \\ -1 & 1 & 1-\lambda \end{vmatrix}$$

$$= (1-\lambda)^3 + 2(1-\lambda) + (1-\lambda)$$

$$= (1-\lambda)(\lambda^2 - 2\lambda + 4)$$

2° The eigenvalues of A are the solutions of $P(\lambda) = 0$. Thus,

$$\lambda_1 = 1, \ \lambda_{2,3} = 1 \pm \sqrt{3}\, i\,.$$

3° An eigenvector $\mathbf{x}$ of A associated with λ_1 is the solution of the system

$$(A - \lambda_1 I)\mathbf{x} = 0\,,$$

i.e.

$$\begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

The solution of the above system is $x_3 = 0, \ x_2 = x_1$.

The choice of $x_1 = 1$ produces the eigenvector $(1, 1, 0)^T$.

Similarly, we have the following two eigenvectors corresponding respectively to $\lambda = 1 + \sqrt{3}\, i$ and $\lambda = 1 - \sqrt{3}\, i$:

$$\left(-\frac{2\sqrt{3}}{3} i, \ \frac{\sqrt{3}}{3} i, \ 1 \right)^T, \quad \left(\frac{2\sqrt{3}}{3} i, -\frac{\sqrt{3}}{3} i, \ 1 \right)^T.$$

Spectral Radius of Matrix

Definition. The **spectral Radius** $\rho(A)$ of a matrix A is defined by $\rho(A) = \max |\lambda_i|$ where λ_i are the eigenvalues of A.

Note: For complex eigenvalue $\lambda = \alpha + \beta i$, $|\lambda| = \sqrt{\alpha^2 + \beta^2}$. For example, for the matrix in the above example $\rho(A) = \max\left\{ |1|, \left|1 + \sqrt{3} i\right|, \left|1 - \sqrt{3} i\right| \right\} = \max\{1, 2, 2\} = 2$.

Properties of Eigenvalues and Eigenvectors of Symmetric and Non-Negative Definite Matrices

Definition (Symmetric Matrix): An $n \times n$ matrix A is said to be symmetric if $A = A^T$.

An $n \times n$ symmetric matrix has the following properties:

- All eigenvalues are real numbers.
- There are n eigenvectors that form an orthonormal set $\{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n\}$ such that

$$< \mathbf{v}_i, \mathbf{v}_j >= \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

where $< \mathbf{v}_i, \mathbf{v}_j >$ denotes inner product of the vectors $\mathbf{v}_i$ and $\mathbf{v}_j$.

Definition (Non-Negative definite matrix) : An $n \times n$ symmetric matrix is non-negative definite if $\mathbf{x}^T A\mathbf{x} \geq 0$ for every n-dimensional column vector $\mathbf{x} \neq 0$.

Theorem 4.6 A symmetric matrix is non-negative definite if and only if all the eigenvalues are non-negative.

Proof

1° Prove the necessary condition: $(\mathbf{x}^T A\mathbf{x} \geq 0 \rightarrow \lambda_i \geq 0)$.

Let λ be an eigenvalue associated with eigenvector $\mathbf{x}$. Then

$$A\mathbf{x} = \lambda \mathbf{x}$$

and hence

$$\mathbf{x}^T A\mathbf{x} = \lambda \mathbf{x}^T \mathbf{x} = \lambda \|\mathbf{x}\|_2^2,$$

Which implies that if A is non-negative definite $(\mathbf{x}^T A\mathbf{x} \geq 0)$, then $\lambda \geq 0$.

2° Prove the sufficient condition: $(\lambda_i \geq 0 \rightarrow \mathbf{x}^T A\mathbf{x} \geq 0)$.

Given $\lambda_i \geq 0$ and let $\mathbf{x} \neq 0$ be any vector. Since A is symmetric, its set of eigenvectors can be made into an orthonormal set $\mathbf{u}_i$ that spans $\mathbb{R}^n$, i.e.,

$$\mathbf{x} = \sum_1^n \alpha_i \mathbf{u}_i \qquad \text{for some} \quad \alpha_i$$

As $A\mathbf{u}_i = \lambda \mathbf{u}_i$, we have

$$\mathbf{x}^T A\mathbf{x} = \left(\sum_j \alpha_j u_j^T\right)\left(\sum_i \alpha_i \lambda_i u_i\right)$$

$$= \sum_i \alpha_i \lambda_i \left(\sum_j \alpha_j u_j^T u_i\right)$$

$$= \sum_i \alpha_i \lambda_i \alpha_i = \sum_i \alpha_i^2 \lambda_i \geq 0.$$

□

Evaluation of $\|A\|_2$ Using Spectral Radius of A^TA

Theorem 4.7 If A is a real $n\times n$ matrix, then $\|A\|_2 = \sqrt{\rho(A^TA)}$.

Proof

$$\begin{aligned}\|A\|_2^2 &= \max_{\|\mathbf{x}\|_2=1}\|A\mathbf{x}\|_2^2 = \max_{\|\mathbf{x}\|_2=1}\left(A\mathbf{x}\right)^T\left(A\mathbf{x}\right)\\ &= \max_{\|\mathbf{x}\|_2=1}\left(\mathbf{x}\right)^T\left(A^TA\mathbf{x}\right) = \max_{\|\mathbf{x}\|_2=1}\left\langle \mathbf{x},\, A^TA\mathbf{x}\right\rangle.\end{aligned} \tag{4.17}$$

As A^TA is symmetric ($[A^TA]^T = A^TA$) and non-negative definite (since $\mathbf{x}^TA^TA\mathbf{x} = \|A\mathbf{x}\|_2^2 \ge 0$), all the eigenvalues of A^TA are non-negative.

Let $\lambda_1 > \lambda_2 > \ldots \ge 0$ be the eigenvalues of A^TA and
$\{\mathbf{x}_1,\, \mathbf{x}_2, \ldots,\, \mathbf{x}_n\}$ be the orthonormal set of eigenvectors spanning $\mathbb{R}^n$.

Then there exist constants $\{C_i\}$ such that, for any non-zero vector $\mathbf{x}$, we have

$$\mathbf{x} = \sum_{i=1}^n C_i\mathbf{x}_i$$

$$\begin{aligned}\therefore\quad \left\langle \mathbf{x},\, A^TA\mathbf{x}\right\rangle &= \left\langle \sum_{i=1}^n C_i\mathbf{x}_i,\, A^TA\sum_{j=1}^n C_j\mathbf{x}_j\right\rangle\\ &= \left\langle \sum_{i=1}^n C_i\mathbf{x}_i,\, \sum_{j=1}^n C_j(A^TA\mathbf{x}_j)\right\rangle\\ &= \left\langle \sum_{i=1}^n C_i\mathbf{x}_{i,}\ \ \sum_{j=1}^n C_j\lambda_j\mathbf{x}_j\right\rangle\\ &= \sum_{i=1}^n C_i\sum_{j=1}^n C_j\lambda_j\left\langle \mathbf{x}_i,\, \mathbf{x}_j\right\rangle = \sum_{i=1}^n C_i^2\lambda_i \le \sum_{i=1}^n C_i^2\lambda_1\end{aligned}$$

where $\lambda_1 = \rho(A^TA)$ is the largest eigenvalue of A^TA .

Further, $$1 = \|\mathbf{x}\|_2^2 = \left\langle \sum_{i=1}^n C_i\mathbf{x}_i,\, \sum_{j=1}^n C_j\mathbf{x}_j\right\rangle = \sum_{i=1}^n C_i^2$$

Thus $\left\langle \mathbf{x},\, A^TA\mathbf{x}\right\rangle \le \lambda_1$ for any $\mathbf{x}$ satisfying $\|\mathbf{x}\|_2 = 1$.

and hence $$\max\left\langle \mathbf{x},\, A^TA\mathbf{x}\right\rangle \le \lambda_1 .$$

The above together with (4.17) implies

$$\|A\|_2^2 \le \lambda_1 \le \rho(A^TA) .$$

Hence $$\|A\|_2 \le \sqrt{\rho(A^T A)} \tag{4.18}$$

Also let $\mathbf{v}$ with $\|\mathbf{v}\|_2 = 1$ be the eigenvector corresponding to λ_1.

Then
$$\begin{aligned}\|A\|_2^2 &= \max_{\|\mathbf{x}\|_2=1} \|A\mathbf{x}\|_2^2 \ge \|A\mathbf{v}\|_2^2 \\ &= (A\mathbf{v})^T (A\mathbf{v}) = \mathbf{v}^T (A^T A\mathbf{v}) = \mathbf{v}^T \lambda_1 \mathbf{v} = \lambda_1 \|\mathbf{v}\|_2^2 \\ &= \rho(A^T A)\end{aligned}$$

Therefore, $$\|A\|_2 \ge \sqrt{\rho(A^T A)}. \tag{4.19}$$

From equations (4.18) and (4.19), we have $\|A\|_2 = \sqrt{\rho(A^T A)}$.

Example 4.7 Given $A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$, find $\|A\|_2$ and $\rho(A)$.

Solution

A is a real 3×3 matrix.

$$A^T A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}\begin{pmatrix} 2 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 5 & 3 & 1 \\ 3 & 3 & 3 \\ 1 & 3 & 5 \end{pmatrix}.$$

The characteristic equation is

$$\det(A^T A - I\lambda) = \begin{vmatrix} 5-\lambda & 3 & 1 \\ 3 & 3-\lambda & 3 \\ 1 & 3 & 5-\lambda \end{vmatrix} = 0.$$

Solving the above equations yields eigenvalues $\lambda = 0, 4, 9$.

Hence

$$\rho(A^T A) = \max_i |\lambda_i| = \max_i (0,\ 4,\ 9) = 9$$

$$\|A\|_2 = \sqrt{\rho(A^T A)} = \sqrt{9} = 3.$$

Relation of Matrix Norm $\|A\|$ and Spectral Radius $\rho(A)$

Theorem 4.8 If A is a real $n \times n$ matrix, then $\rho(A) \le \|A\|$ for any natural norm.

Proof

Suppose λ (with $|\lambda| = \max_i |\lambda_i|$) is the eigenvalue of A with largest magnitude, and $\mathbf{x}$ is the associated eigenvector with $\|\mathbf{x}\| = 1$.

Since $$A\mathbf{x} = \lambda\,\mathbf{x},$$

we have from the properties of vector norms and (4.11)

$$|\lambda| = |\lambda|\|\mathbf{x}\| = \|\lambda\mathbf{x}\| = \|A\mathbf{x}\| \le \|A\|\|\mathbf{x}\| = \|A\|$$

Therefore, $\rho(A) = \max|\lambda_i| = |\lambda| \le \|A\|$. □

Theorem 4.9 For any matrix A and any $\varepsilon > 0$, $\exists$ a natural norm $\|\cdot\|_\varepsilon$ with the property that

$$\rho(A) < \|A\|_\varepsilon < \rho(A) + \varepsilon .$$

Thus $\rho(A)$ is the greatest lower bound of the norm of A.

Corollary For a square matrix A, $\rho(A) < 1 \quad \Leftrightarrow \quad \|A\|_\varepsilon < 1$ for some natural norm.

Proof of Corollary

(1°) Prove $\rho(A) < 1$ if $\|A\|_\varepsilon < 1$.

By theorem 4.9, $\rho(A) < \|A\|_\varepsilon$.

Hence, if $\|A\|_\varepsilon < 1$ then $\rho(A) < 1$.

(2°) Prove if $\rho(A) < 1$ then $\|A\|_\varepsilon < 1$.

$\rho(A) < 1 \quad \Rightarrow \quad 0 < 1 - \rho(A) = \varepsilon$.

By the theorem above, for any ε as above, there exists a norm $\|\cdot\|_\varepsilon$ such that

$$\|A\|_\varepsilon \le \rho(A) + \varepsilon < \rho(A) + [1 - \rho(A)] = 1.$$

Convergence of Matrices

Definition: We say an $n \times n$ matrix A convergent if $\lim_{k\to\infty}\left(A^k\right)_{ij} = 0$ for each $i, j = 1, 2, \ldots, n$

Example 4.8 Show that the matrix $A = \begin{bmatrix} 1/2 & 0 \\ 1/4 & 1/2 \end{bmatrix}$ is convergent

Solution

$$A^2 = \begin{bmatrix} 1/2 & 0 \\ 1/4 & 1/2 \end{bmatrix}\begin{bmatrix} 1/2 & 0 \\ 1/4 & 1/2 \end{bmatrix} = \begin{bmatrix} 1/4 & 0 \\ 1/4 & 1/4 \end{bmatrix}$$

$$A^3 = A^2A = \begin{bmatrix} \frac{1}{4} & 0 \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix}\begin{bmatrix} \frac{1}{2} & 0 \\ \frac{1}{4} & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{8} & 0 \\ \frac{3}{16} & \frac{1}{8} \end{bmatrix} = \begin{bmatrix} \left(\frac{1}{2}\right)^3 & 0 \\ \frac{3}{2^{3+1}} & \left(\frac{1}{2}\right)^3 \end{bmatrix}$$

$$A^4 = A^3A = \begin{bmatrix} \frac{1}{16} & 0 \\ \frac{1}{8} & \frac{1}{16} \end{bmatrix} = \begin{bmatrix} \left(\frac{1}{2}\right)^4 & 0 \\ \frac{4}{2^{4+1}} & \left(\frac{1}{2}\right)^4 \end{bmatrix}$$

and in general

$$A^k = \begin{bmatrix} \left(\frac{1}{2}\right)^k & 0 \\ \frac{k}{2^{k+1}} & \left(\frac{1}{2}\right)^k \end{bmatrix}.$$

Since $\lim_{k\to\infty}\left(\frac{1}{2}\right)^k = 0,\ \lim_{k\to\infty}\frac{k}{2^{k+1}} = 0$, A is a convergent matrix.

Remark. $\rho(A) = 1/2$, since ½ is the only eigenvalue. There is a connection between the spectral radius of a matrix and the convergence of the matrix, which is detailed below.

Equivalent Conditions of Convergence of Matrices

Theorem 4.10 The following statements are equivalent

(a) A is a convergent matrix

(b) $\lim_{m\to\infty}\left\|A^m\right\| = 0$ for some natural norm

(c) $\lim_{m\to\infty}\left\|A^m\right\| = 0$ for all natural norms

(d) $\rho(A) < 1$

(e) $\lim_{m\to\infty} A^m\mathbf{x} = 0$ for every $\mathbf{x}$.

Proof

Preliminary:

- Every matrix norm $\|A\|$ is a continuous function of the n^2 elements a_{ij} of A.
- For each pair of matrix norms, say $\|A\|$ and $\|A\|'$, $\exists$ positive constants m, M such that for all matrix A,

$$m\|A\|' \le \|A\| \le M\|A\|'.$$

(1°) (a) $\equiv$ (b), (a) $\Rightarrow$ (c).

Since $\|.\|$ is continuous and $\lim_{k\to\infty}\left\|A^k\right\| = \|\underline{0}\| = 0$, (a) $\Rightarrow \begin{cases}(c)\\(b).\end{cases}$

If (b) holds for some norms, then $\exists\ M$ such that

$$\left\|A^m\right\|_\infty \le M\left\|A^m\right\| \to 0.$$

As $\left\|A^m\right\|_\infty = \max_i \sum_{j=1}^n \left|(a^m)_{ij}\right|$, $\left\|A^m\right\|_\infty \to 0$ implies $\left(a^m\right)_{ij} \to 0$. Hence (a) holds.

(2°) Prove (b) $\equiv$ (d).

Let λ be an eigenvalue of A and $\mathbf{x}$ be an associated eigenvector, then

$$\begin{aligned} A\mathbf{x} &= \lambda\mathbf{x} \\ AA\mathbf{x} &= \lambda A\mathbf{x} = \lambda^2\mathbf{x} \\ &\vdots \\ A^m\mathbf{x} &= \lambda^m\mathbf{x} \end{aligned}$$

Thus, λ^m is an eigenvalue of A^m and we denote $\lambda(A^m) = \left(\lambda(A)\right)^m$

Hence, $\left\|A^m\right\| \ge \rho(A^m) = \left(\rho(A)\right)^m$

If (b) holds, i.e. $\lim_{m\to\infty}\left\|A^m\right\| = 0$, then from the above, $\lim_{m\to\infty}[\rho(A)]^m \to 0$ which implies $\rho(A) < 1$ and so (b) $\Rightarrow$ (d).

If (d) holds, i.e. $\rho\left(A\right) < 1$, then $\exists\ \varepsilon > 0$ and a norm $\|.\|$ such that

$$\|A\| \le \rho\left(A\right) + \varepsilon = \theta < 1.$$

Thus $\left\|A^m\right\| \le \|A\|^m < \theta^m$, so $\lim_{m\to\infty}\left\|A^m\right\| = 0$ and hence (b) holds.

3° (a) $\equiv$ (e)

First prove (a) $\Rightarrow$ (e): A is convergent $\Rightarrow$ $A^k\mathbf{x} \to 0$.

If (a) holds, i.e. A is convergent, then $\lim_{m\to\infty}\left\|A^m\right\| = 0$ and so

$$\left\|A^k\mathbf{x}\right\| \le \left\|A^k\right\| \cdot \|\mathbf{x}\| \to 0 \quad \text{for each } \mathbf{x} \text{ as } \quad k \to \infty,$$

$$\therefore \quad A^k\mathbf{x} \to 0.$$

Then prove (e) $\Rightarrow$ (a): $A^k\mathbf{x} \to 0 \Rightarrow$ A is convergent.

Suppose $A^k\mathbf{x} \to 0$.

W choose $\mathbf{x}$ = eigenvector of A corresponding to the eigenvalue λ, then

$$A^k\ \mathbf{x} = \lambda^k\ \mathbf{x}.$$

Since $A^k\mathbf{x} \to 0$, we have

$$\left\|A^k\mathbf{x}\right\|_\infty \to 0,$$

that is
$$\left\|A^k\mathbf{x}\right\|_\infty = \left\|\lambda^k\mathbf{x}\right\|_\infty = |\lambda|^k\ \|\mathbf{x}\|_\infty \to 0.$$

For the above to be true for any eigenvector, we require $|\lambda| < 1$ for any eigenvactor

$$\therefore\quad \rho(A) < 1$$

and thus A is convergent. Therefore (a) holds. □

4.4 Error Analysis

Stability and a Priori Error Bound

For a linear system $A\mathbf{x} = b$, if small changes are made in A or b, do large change in the solution of $\mathbf{x}_0$ occur? If yes, the solution of the system is said to be unstable or ill-conditioned.

If the system is unstable, there may be no point in even attempting to obtain an approximate solution. This is because if the data of the system are measured quantities or otherwise known to only a limited precision, even an exact solution to the system is quite likely to be meaningless. Thus, how to test the stability of a system is an important issue.

Suppose first that the data A and b in

$$A\mathbf{x} = b \tag{4.20}$$

are perturbed by the quantities δA and δb. Then if the perturbation in the solution $\mathbf{x}$ is $\delta\mathbf{x}$, we have

$$(A+\delta A)(\ \mathbf{x} + \delta\mathbf{x}\ ) = \mathrm{b} + \delta b. \tag{4.21}$$

Assume that $A+\delta\ A$ is nonsingular, then we can have from (4.20) and (4.21) that

$$\delta\mathbf{x} = (A+\delta A)^{-1}(\delta b - \delta A\mathbf{x}).$$

Hence,
$$\|\delta\mathbf{x}\| \le \left\|(A+\delta A)^{-1}\right\|\left(\|\delta b\| + \|\delta A\|\|\mathbf{x}\|\right)$$

Therefore, $$\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|} \le \left\|(A+\delta A)^{-1}\right\| \left(\|\delta A\| + \|A\| \frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|} \right) \tag{4.22}$$

where in the above we have used

$$\|b\| = \|A\mathbf{x}\| \le \|A\| \|\mathbf{x}\|.$$

It is now necessary to obtain an estimate for $\left\|(A+\delta A)^{-1}\right\|$. Firstly introduce the following Lemmas.

Neumann Lemma Let $\rho(B) < 1$. Then $(I-B)^{-1}$ exists and

$$(I-B)^{-1} = \lim_{k\to\infty} \sum_{i=0}^{k} B^i. \tag{4.23}$$

Proof

For any eigenvalue λ of B, $1-\lambda$ is an eigenvalue of $I-B$.

Since $|\lambda| \le \rho(B) < 1$, it follows that no eigenvalue of $I-B$ can be zero and consequently $I-B$ is nonsingular.

Let $\qquad S_m = I + B + B^2 + \dots\dots + B^m$

then $\qquad (I-B)S_m = I - B^{m+1}$.

Since B is convergent (as $\rho(B) < 1$), we have

$$\lim_{m\to\infty} (I-B)S_m = I$$

$$\therefore \quad \lim_{m\to\infty} S_m = \sum_{i=0}^{\infty} B^i = (I-B)^{-1}. \qquad \square$$

Perturbation Lemma, Banch Lemma

If A is nonsingular and $\|\delta A\| \le \dfrac{1}{\|A^{-1}\|}$, then $A+\delta A$ is nonsingular and

$$\left\|(A+\delta A)^{-1}\right\| \le \frac{\|A^{-1}\|}{1 - \|A^{-1}\| \|\delta A\|}. \tag{4.24}$$

Proof

Set $B = -A^{-1}\delta A,$

then $\|B\| \le \|A^{-1}\|\|\delta A\| < 1$

Thus, $\rho(B) < 1$

Therefore, from the Neumann Lemma $(I-B)^{-1} = \sum_{i=0}^{\infty} B^i$

Hence, $\left\|(I-B)^{-1}\right\| \le \sum_{i=0}^{\infty}\|B\|^i = \frac{1}{1-\|B\|} \le \frac{1}{1-\|A^{-1}\|\|\delta A\|},$

where we have used the formula $\frac{1}{1-x} = \sum_{i=0}^{\infty} x^i.$

But $A+\delta A = A(I+A^{-1}\delta A) = A(I-B),$

and so $A+\delta A$, as the product of nonsingular matrices, is nonsingular and

$$\left\|(A+\delta A)^{-1}\right\| = \left\|(I-B)^{-1}A^{-1}\right\| \le \frac{\|A^{-1}\|}{1-\|A^{-1}\|\|\delta A\|}. \qquad \square$$

Now from (4.22) and the perturbation Lemma, we can have the following theorem.

Theorem 4.11 (Priori Error Estimate Theorem)

Suppose A is nonsingular and $\|\delta A\| \le \frac{1}{\|A^{-1}\|}$, then the solution $(\mathbf{x}+\delta\mathbf{x})$ to

$$(A+\delta A)(\mathbf{x}+\delta\mathbf{x}) = (\mathbf{b}+\delta\mathbf{b})$$

approximates the solution $\mathbf{x}$ of $A\mathbf{x}=\mathbf{b}$ with error bound

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \frac{K(A)}{1-K(A)\|\delta A\|/\|A\|}\left(\frac{\|\delta b\|}{\|b\|}+\frac{\|\delta A\|}{\|A\|}\right) \tag{4.25}$$

where $K(A) = \|A\|\|A^{-1}\|$ is known as the condition number of A.

Condition Number *K*(*A*)

We define

$$K(A) = \begin{cases} \|A\|\|A^{-1}\| & \text{if } A \text{ is nonsingular} \\ +\infty & \text{if } A \text{ is singular} \end{cases}$$

as condition number of A relative to a natural norm.

It can be noted that

$$1 \le K(A) < \infty,$$

as for any nonsingular matrix A and natural norm

$$1 = I = \left\| A \cdot A^{-1} \right\| \le \|A\| \left\| A^{-1} \right\| = K(A).$$

In error estimate (4.25), $K(A)\|\delta A\|/\|A\| = \|\delta A\| \left\| A^{-1} \right\| = \alpha < 1$ by assumption. Thus, we may expand $(1-\alpha)^{-1}$ by its geometric series

$$\frac{1}{1-\alpha} = 1 + \alpha + \alpha^2 + \dots ,$$

and conclude that, to a first order approximation, the relative error in $\mathbf{x}$ is $K(A)$ times the relative errors in A and b.

For small value of $K(A)$ (say one), small changes in A and b only cause small changes in $\mathbf{x}$, thus the system (matrix) is said stable (well-conditioned) while for large value of $K(A)$, small changes in A and b will cause relatively large changes in $\mathbf{x}$ and thus the system (matrix) is unstable (ill-conditioned).

Approximate Condition Number

An approximation for the condition number, involved with solving the system $A\mathbf{x} = b$ using Gaussian elimination and the t-digit type of arithmetic, can be derived as

$$K(A) \approx \frac{\|y\|}{\|x\|} 10^t$$

where $A\mathbf{y} = r$ and r is the residual vector ($b - A\mathbf{x}$).

Example 4.10 Consider the stability of a system $A\mathbf{x} = b$ with $A = \begin{bmatrix} 1 & 2 \\ 1.0001 & 2 \end{bmatrix}$.

Solution

$$\|A\|_\infty = 3.0001,$$

$$A^{-1} = \frac{1}{\begin{vmatrix} 1 & 2 \\ 1.0001 & 2 \end{vmatrix}} \begin{bmatrix} 2 & -2 \\ -1.0001 & 1 \end{bmatrix} = \begin{bmatrix} -10000 & 10000 \\ 5000.5 & -5000 \end{bmatrix},$$

$$\left\| A^{-1} \right\|_\infty = 20,000,$$

$$\therefore \quad K_\infty(A) = 3.0001 * 20,000 = 60,002.$$

which means that a small relative change in A or b will cause significant large relative changes in $\mathbf{x}$, so the system is unstable, i.e. ill-conditioned.

Residual and a Posteriori Error Bound

If $\tilde{\mathbf{x}}$ is an approximation to the solution of $A\mathbf{x} = b$ and the residual vector $r = b - A\tilde{\mathbf{x}}$ has the property that $\|r\|$ is small, one may conclude that $\tilde{\mathbf{x}}$ would be satisfactory. However, this is not always the case. In fact, the smallness of $\|r\|$ does not guarantee that $\tilde{\mathbf{x}}$ is close to the true solution $\mathbf{x}$. It is the *condition number* $K(A)$ that *again plays the crucial role.*

For example, the system $A\mathbf{x} = b$ given by

$$\begin{bmatrix} 1 & 1 \\ 1.0001 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 3.0001 \end{bmatrix}$$

has the unique solution $\mathbf{x} = (1, 1)^T$. The poor approximation $\tilde{\mathbf{x}} = (3, 0)^T$ has the residual vector

$$r = b - A\tilde{\mathbf{x}} = \begin{bmatrix} 3 \\ 3.0001 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 1.0001 & 2 \end{bmatrix}\begin{bmatrix} 3 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.0002 \end{bmatrix}$$

and $\|r\|_\infty = 0.0002$.

Although the norm of the residual vector is small, the approximation is obviously poor.

How to Estimate the Posterior Errors?

Let $\qquad r = b - A\tilde{\mathbf{x}} = A\mathbf{x} - A\tilde{\mathbf{x}} = A\delta\mathbf{x}$

Then $\qquad \delta\mathbf{x} = A^{-1}r, \qquad \|\delta\mathbf{x}\| \le \|A^{-1}\|\|r\|.$

Moreover, since $b = A\mathbf{x}$, we have

$$\|b\| \le \|A\|\|\mathbf{x}\|$$

and thus

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \|A^{-1}\|\|A\|\frac{\|r\|}{\|b\|} = K(A)\frac{\|r\|}{\|b\|}.$$

Theorem 4.12 (Posterior error Estimate Theorem)

If $\tilde{\mathbf{x}}$ is an approximation to the solution of $A\mathbf{x} = b$ and A is a nonsingular matrix, then for any natural norm,

$$\|\delta\mathbf{x}\| \le \|r\|\|A^{-1}\|, \qquad (4.26)$$
$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le K(A)\frac{\|r\|}{\|b\|},$$

where $\delta\mathbf{x} = \tilde{\mathbf{x}} - \mathbf{x}$ and r is the residual vector for $\tilde{\mathbf{x}}$ with respect to $A\mathbf{x} = b$.

Remark. If $K(A)$ is close to one, a small residual vector implies a correspondingly accurate approximate solution and the matrix A is said to be well behaved (well-conditioned), while if $K(A)$ is large, small residual vector does not necessarily guarantee the accuracy of results.

4.5 The Residual Correction Method for Solving Linear Systems of Equations

Assume that $A\mathbf{x} = b$ has been solved for an approximate solution $\hat{\mathbf{x}}^{(0)} = \mathbf{x}^{(0)}$. To improve the accuracy, we first calculate the residual vector

$$r^{(0)} = b - A\mathbf{x}^{(0)}.$$

Define $\mathbf{e}^{(0)} = \mathbf{x} - \mathbf{x}^{(0)}$, we then have

$$A\mathbf{e}^{(0)} = r^{(0)}$$

which yields an approximation $\mathbf{e}^{(0)}$.

In general, $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \hat{\mathbf{e}}^{(0)}$ is a more accurate solution to the system $A\mathbf{x} = b$ than $\mathbf{x}^{(0)}$. This process can be repeated, calculating $\mathbf{x}^{(2)}$, $\mathbf{x}^{(3)}$, $\mathbf{x}^{(4)}$,... , to continually reduce the error. The method using this process is called iteration refinement or residual correction method. The method is generally used only for ill-conditioned systems, since this technique will not significantly improve the approximation for a well-conditioned system.

Iteration Refinement Algorithm

```
Input:   Number of equations, the entries of $A$ and $\mathbf{b}$, Maximum number of iterations
         $N$ and tolerance $Tol$.
Output:  The approximate solution $xx = (xx_1, xx_2, ......., xx_n)^T$ or a message that the
         number of iterations was exceeded.
Step 1:  Solve $A\mathbf{x} = b$ for $\mathbf{x}$, saving $LU$ and noting row interchanges.
Step 2:  do steps 3 – 7 for $k$ =1, 2, …, $N$.
Step 3:      for $i$ =1, 2, …, $n$ (calculate $r$)
                 set $r_i = b_i - \sum_{j=1}^{n} a_{ij} x_j$
Step 4:      back substitution to solve $A\mathbf{e} = r$ using the stored $LU$ decomposition
Step 5:      for $i$ =1, 2, …, $n$, set $xx_i = x_i + e_i$
Step 6:      if $\|x - xx\|_\infty < Tol$ then output $xx$ and stop
Step 7:      else set $x_i = xx_i$ for $i$=1, 2, …, $n$, and then go to step 3
Step 8:  Print "Maximum Number of Iterations Exceeded".
```

4.6 Calculating Matrix Norms Using Maple/MATLAB

Calculating Matrix Norms Using Maple

The Maple function "`norm()`" calculates the ℓ_2 matrix norm

Example. Given $A = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix}$, find $\|A\|_2$

```
> norm(A,2);
```

gives

$$\sqrt{\left|\frac{1}{3}\left(46+61\sqrt{237}\right)^{1/3}+\frac{22}{3\left(46+61\sqrt{237}\right)^{1/3}}+\frac{10}{3}\right|}.$$

Calculating Matrix Norms Using MATLAB

The MATLAB built-in function "**`norm()`**" calculates several types of matrix norms:

`norm(A,p)`	returns ℓ_p	matrix norm.
`norm(A)`	returns ℓ_2	matrix norm.
`norm(A,inf)`	returns ℓ_∞	matrix norm.

Example. Given $A = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix}$. Find $\|A\|_2$ and $\|A\|_\infty$.

```
>> norm(A,2)
```

produces

```
ans =
        2.4998
```

and

```
>> norm(A,inf)
```

produces

```
ans =
        3
```

EXERCISE 4

Q4.1 Find $\|\mathbf{x}\|_1$, $\|\mathbf{x}\|_2$ and $\|\mathbf{x}\|_\infty$ for $\mathbf{x} = (\sin k, \cos k, 2^k)^T$ where k is a fixed positive integer.

Q4.2 Verify that $\|\mathbf{x}\|_2 = \left\{\sum_{i=1}^{n} x_i^2\right\}^{\frac{1}{2}}$ is a norm on $\mathbb{R}^n$.

Q4.3 Verify that $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$ is a norm on $\mathbb{R}^n$

Q4.4 Calculate the l_2 and l_∞ distances between $\mathbf{x} = (2,\ 1,\ 0)$ and $\mathbf{y} = (0, -2,\ 1)$

Q4.5 Prove that the following sequences are convergent and find their limits

(a) $\mathbf{x}^{(k)} = \left(\frac{1}{k},\ e^{1-k},\ \frac{-2}{k^2}\right)^T$

(b) $\mathbf{x}^{(k)} = \left(e^k \cos k,\ k \sin\frac{1}{k},\ 3 + k^{-2}\right)^T$

Q4.6 Prove that $\|A\| = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|$ is a matrix norm.

Q4.7 Find $\|\cdot\|_1$ and $\|\cdot\|_\infty$ for $A = \begin{bmatrix} 1 & -1 \\ 2 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 10 & 15 \\ 0 & 1 \end{bmatrix}$

Q4.8 Verify that $\|AB\| \le \|A\|\,\|B\|$ in Q4.7.

Q4.9 Compute the eigenvalues and associated eigenvectors of the following matrices:

(a) $\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$ (b) $\begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$

Q4.10 Find the spectral radius for each matrix in question 4.9.

Q4.11 Find $\|\cdot\|_2$ for each matrix in question 4.9 using both $\|A\|_2 = \sqrt{\rho(A^T A)}$ and $\|A\|_2 = \max_{\|\mathbf{x}\|_2=1} \|A\mathbf{x}\|_2$

Q4.12 Which of the matrices in question 4.9 are convergent?

Q4.13 Show that if A is symmetric, then $\|A\|_2 = \rho(A)$.

Q4.14 Show that $A_1 = \begin{bmatrix} 1 & 0 \\ 1/4 & 1/2 \end{bmatrix}$ is not convergent but $A_2 = \begin{bmatrix} 1/2 & 0 \\ 16 & 1/2 \end{bmatrix}$ is convergent.

Q4.15 Show that for any two vectors $\underline{a}$ and $\underline{b}$ and any vector norm, $\big|\, \|\mathbf{a}\| - \|\mathbf{b}\| \,\big| \le \|\mathbf{a} - \mathbf{b}\|$

Q4.16 Calculate $K_1(A)$, $K_2(A)$ and $K_\infty(A)$ for $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

Q4.17 Compute the condition numbers of the following matrices relative to $\|\cdot\|_\infty$

(a) $\begin{bmatrix} 1 & 2 \\ 1.0001 & 2 \end{bmatrix}$ (b) $\begin{bmatrix} 3.9 & 1.6 \\ 6.8 & 2.9 \end{bmatrix}$

Q4.18 Consider the system

$$A\mathbf{x} = \begin{bmatrix} 1 & 2 \\ 1.0001 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 3.0001 \end{bmatrix}$$

with exact solution $x_1 = x_2 = 1$, and the system

$$(A+\delta A)(x+\delta x) = \begin{bmatrix} 1 & 2 \\ 0.9999 & 2 \end{bmatrix}\begin{bmatrix} x_1+\delta x_1 \\ x_2+\delta x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 3.0001 \end{bmatrix}.$$

Compute $\|\delta\mathbf{x}\|_\infty / \|\mathbf{x}\|_\infty$ and estimate for this from the error estimate theorem 4.11. Is A ill-conditioned?

Q4.19 Show that if B is singular, then $\dfrac{1}{K(A)} \le \dfrac{\|A-B\|}{\|A\|}$

Hint: 1) Show that $\|A\mathbf{x}\| \ge \|\mathbf{x}\| / \|A^{-1}\|$

2) There is $\mathbf{x} \neq 0$, with $\|\mathbf{x}\| = 1$, such that $B\mathbf{x} = 0$ so $A\mathbf{x} = (A-B)\mathbf{x}$.

Q4.20 Using Q4.19, estimate the condition numbers $K_\infty(A)$ for Q4.17(a) and Q4.17(b).

Q4.21 Solve the following using Gauss elimination and iterative refinement using 3 digit arithmetic

(a) $3.9x_1 + 1.6x_2 = 5.5$
$0.8x_1 + 2.9x_2 = 9.7$

(b) $4.56x_1 + 2.18x_2 = 6.74$
$2.79x_1 + 1.38x_2 = 4.13$

Q4.22 Write a computer subroutine to implement the iterative refinement scheme for $A\mathbf{x} = b$ and solve the following system using the program developed.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5.1 & 6 \\ 7.01 & 8 & 9.01 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 15.1 \\ 29.2 \end{pmatrix}$$

CHAPTER 5

Iterative Methods for Systems of Linear Equations

For large systems with a high percentage of zero entries, iterative techniques are usually more efficient in terms of both computer storage and computation time than direct methods.

Solving an $n \times n$ linear system by iterative techniques includes the following general steps

- Convert $A\mathbf{x} = b$ to $N\mathbf{x} = T\mathbf{x} + c$
- Select an initial vector $\mathbf{x}^{(0)}$ approximating $\mathbf{x}$
- Generate a sequence of vectors $\mathbf{x}^{(k)}$ that converges to $\mathbf{x}$ by

$$N\,\mathbf{x}^{(k)} = T\,\mathbf{x}^{(k-1)} + c, \qquad k = 1, 2, \ldots$$

Jacobi and Gauss-Seidel iterative methods are two basic iterative techniques which convert $A\mathbf{x} = b$ to $N\mathbf{x} = T\mathbf{x} + \mathbf{c}$ using different ways.

In this chapter, we first introduce the Jacobi method and Gauss-Seidel method respectively in sections 5.1 and 5.2. Then we present the general convergence condition for iterative methods and particularly the sufficient conditions for convergence of the Jacobi and Gauss-Seidel methods in section 5.3, followed by the topics of error estimate and speed of convergence in section 5.4. Finally in section 5.5, we introduce the successive over relaxation (SOR) and under relaxation methods by modifying the Gauss-Seidel method to include a scaling (relaxation) factor.

5.1 The Jacobi Method

In this method, the system

$$A\mathbf{x} = b \tag{5.1}$$

is converted to $N\mathbf{x} = T\mathbf{x} + \mathbf{c}$ by splitting A into its diagonal and off-diagonal parts

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} - \begin{bmatrix} 0 & 0 & \cdots & 0 \\ -a_{21} & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots \\ -a_{n1} & \cdots & -a_{n(n-1)} & 0 \end{bmatrix} - \begin{bmatrix} 0 & -a_{12} & \cdots & -a_{1n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & -a_{(n-1)n} \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{5.2}$$

$$= D - L - U$$

Thus, equation (5.1) becomes

$$(D - L - U)\mathbf{x} = b$$

which gives

$$D\mathbf{x} - (L + U)\mathbf{x} = b$$

and so

$$\mathbf{x} = D^{-1}(L+U)\mathbf{x} + D^{-1}b\,. \tag{5.3}$$

Hence, we can establish the following iterative formula

$$\mathbf{x}^{(k)} = T_j\mathbf{x}^{(k-1)} + c\,, \tag{5.4}$$

where $T_j = D^{-1}(L+U)$ and $c = D^{-1}b.$

Alternatively, (5.4) can be expressed as

$$\mathbf{x}_i^{(k)} = \frac{1}{a_{ii}}\left[-\sum_{j=1}^{i-1} a_{ij}\mathbf{x}_j^{(k-1)} - \sum_{j=i+1}^{n} a_{ij}\mathbf{x}_j^{(k-1)} + b_i\right], \quad i=1, 2,\dots, n. \tag{5.5}$$

The criterion to stop generating new term is

$$\frac{\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right\|}{\left\|\mathbf{x}^{(k)}\right\|} < Tol.$$

Jacobi Iterative Algorithm for Solving ***Ax*=*b***

To solve $A\mathbf{x} = b$, given an initial approximation $\mathbf{x}^{(0)}$.

Input: Number of equations N; matrix A, vector b, initial guess $\mathbf{x}_0$, tolerance Tol, maximum number of iterations N_{iter}

Output: The solution **x** or a message that the number of iterations was exceeded.

```
Step 1:  $k \leftarrow 1$
Step 2:  while ($k \le N_{iter}$) do steps 3 – 5
Step 3:        for $i = 1$ to $N$ do
                    $\mathbf{x}_i \leftarrow \frac{1}{a_{ii}}\left[-\sum_{j=1, j\neq i}^{n} a_{ij}\mathbf{x}_{0j} + b_i\right]$
Step 4:        if $\|\mathbf{x} - \mathbf{x}_0\| < Tol$ then
                    Output x
                    STOP
Step 5:        else
                    $\mathbf{x}_0 \leftarrow \mathbf{x}$
                    $k \leftarrow k+1$
                    then go to step 3
Step 6:  Output "Number of iterations was exceeded".
         STOP.
```

Example 5.1 Find the solution to the system

$$\begin{bmatrix} 9 & -1 & -1 \\ -1 & 8 & 0 \\ -1 & 0 & 9 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 7 \\ 7 \\ 8 \end{bmatrix}$$

Solution

$$\mathbf{x}^{(k)} = \begin{bmatrix} 0 & 1/9 & 1/9 \\ 1/8 & 0 & 0 \\ 1/9 & 0 & 0 \end{bmatrix} \mathbf{x}^{(k-1)} + \begin{bmatrix} 7/8 \\ 7/8 \\ 8/9 \end{bmatrix}.$$

Select $\mathbf{x}^{(0)} = 0$. The numerical results for k=1,2,3,4,5, obtained by using the Jacobi method, are given in Table 5.1.

Table 5.1 Numerical Solutions by Jacobi and Gauss-Seidel Methods

k	0	1	2	3	4	5
$\mathbf{x}^k$ by Jacobi	0 0 0	0.7798 0.8750 0.8889	0.9738 0.9722 0.9971	0.9942 0.9967 0.9971	0.9993 0.9993 0.9993	0.9998 0.9999 0.9999
$\mathbf{x}^k$ by Gauss-Seidel	0 0 0	0.7778 0.9722 0.9753	0.9942 0.9993 0.9993	0.9998 1.0000 1.0000	1.0000 1.0000 1.0000	

5.2 The Gauss-Seidel Method

A possible improvement to Jacobi's scheme is suggested by an analysis of equation (5.5). To compute $x_i^{(k)}$, the components of $x_i^{(k-1)}$ are used in Jacobi's method. Since, for $i \geq 1$, x_1^k, x_2^k, ..., x_{i-1}^k have already been computed and are likely to be better approximation to the actual solution of x_1, x_2, ..., x_{i-1} than $x_1^{(k-1)}$, $x_2^{(k-1)}$, ..., $x_{i-1}^{(k-1)}$, it is reasonable to compute $\mathbf{x}_i^{(k)}$ using these most recently calculated data.

The Gauss-Seidel iteration scheme is based on this consideration and takes the following form

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[-\sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} + b_i \right]. \tag{5.6}$$

We can also derive this formula and its matrix form directly from the original equation (5.1). Using (5.2), equation (5.1) can be written as

$$(D - L)\mathbf{x} = U\mathbf{x} + b. \tag{5.7}$$

Then the Gauss-Seidel iteration scheme is

$$(D-L)\mathbf{x}^k = U\mathbf{x}^{(k-1)} + b\,, \tag{5.8}$$

which gives
$$D\mathbf{x}^k = L\mathbf{x}^k + U\mathbf{x}^{(k-1)} + b\,. \tag{5.9}$$

or
$$\mathbf{x}^k = (D-L)^{-1}U\mathbf{x}^{(k-1)} + (D-L)^{-1}b\,. \tag{5.10}$$

Formulae (5.9) is precisely formulae (5.6).

Gauss-Seidel Algorithm for solving $A\mathbf{x} = b$

Input: Number of equations N; matrix A, vectors b and $\mathbf{x}_0$, tolerance Tol, and maximum number of iterations N_{iter}.

```
Step 1:  $k \leftarrow 1$
Step 2:  while ($k \le N_{iter}$) do steps 3 to 5
Step 3:  for $i = 1$ to N do
             $\mathbf{x}_i \leftarrow \frac{1}{a_{ii}}\left[-\sum_{j=1}^{i-1} a_{ij}\mathbf{x}_j - \sum_{j=i+1}^{n} a_{ij}\mathbf{x}_{0j} + b_i\right]$
Step 4:  if $\|\mathbf{x}-\mathbf{x}_0\| < Tol$ then
             Output $\mathbf{x}_i$
             STOP
Step 5:  else
             k ← k+1
             $\mathbf{x}_0 \leftarrow \mathbf{x}$
             go to step 3
Step 6:  Output "Maximum number of iterations was exceeded".
         STOP
```

Example 5.2 Find the solution to the following system:

$$\begin{bmatrix} 9 & -1 & -1 \\ -1 & 8 & 0 \\ -1 & 0 & 9 \end{bmatrix}\mathbf{x} = \begin{bmatrix} 7 \\ 7 \\ 8 \end{bmatrix}$$

Solution

Select $\mathbf{x}_1 = \mathbf{x}_2 = \mathbf{x}_3 = 0$ and use the following iterative formula to calculate the value of $\mathbf{x}_i$ for each cycle

$$\mathbf{x}_1 \leftarrow \frac{1}{9}(7+\mathbf{x}_2+\mathbf{x}_3),$$
$$\mathbf{x}_2 \leftarrow \frac{1}{8}(7+\mathbf{x}_1),$$
$$\mathbf{x}_1 \leftarrow \frac{1}{9}(8+\mathbf{x}_1).$$

The computed results are shown in Table 5.1.

5.3 Convergence Conditions

General Convergence Conditions

An iterative technique to solve the $n \times n$ linear system $A\mathbf{x} = b$ starts with an initial approximation $\mathbf{x}^{(0)}$ to the solution $\mathbf{x}$, and then generates a sequence of vectors

$$\mathbf{x}^{(k)} = T\,\mathbf{x}^{(k\text{-}1)} + \mathbf{c}, \qquad k = 1, 2, \ldots.$$

that converges to $\mathbf{x}$.

Questions: 1) Does $\mathbf{x}^{(k)}$ converge to the solution $\mathbf{x} = T\mathbf{x} + \mathbf{c}$ for any case?
2) If not, what is the restriction to $\mathbf{x}^{(0)}$, T or $\mathbf{c}$?

Theorem 5.1 For any $\mathbf{x}^{(0)} \in \mathbb{R}^n$, the sequence $\left\{\mathbf{x}^{(k)}\right\}_{k=0}^{\infty}$ defined by

$$\mathbf{x}^{(k)} = T\mathbf{x}^{(k\text{-}1)} + \mathbf{c} \quad (k \geq 1 \text{ and } \mathbf{c} \neq \mathbf{0}) \tag{5.11}$$

converges to the unique solution of $\mathbf{x} = T\mathbf{x} + \mathbf{c}$ iff $\rho(T)<1$.

Proof

(i) First prove that $\rho(T)<1 \Rightarrow$ the sequence $\mathbf{x}^{(k)}$ generated by (5.11) converges to the unique solution.

From (5.11)

$$\begin{aligned}\mathbf{x}^{(k)} &= T\,\mathbf{x}^{(k\text{-}1)} + \mathbf{c}\\ &= T(T\,\mathbf{x}^{(k\text{-}2)} + \mathbf{c}) + \mathbf{c}\\ &= T^2\,\mathbf{x}^{(k\text{-}2)} + (T+I)\mathbf{c}\\ &\vdots\\ &= T^k\,\mathbf{x}^{(0)} + (T^{k\text{-}1} + \ldots\ldots.+ T + I)\mathbf{c}.\end{aligned}$$

As $\rho(T)< 1$, from Theorem 4.10 (Equivalent conditions of convergence), we have

$$\lim_{k\to\infty} T^k\mathbf{x}^{(0)} = 0.$$

From the Neumann Lemma (in section 4.4),

$$\lim_{k\to\infty}\sum_{j=0}^{k-1}T^j=(I-T)^{-1},$$

Hence

$$\lim_{k\to\infty}\mathbf{x}^{(k)}=\lim_{k\to\infty}T^k\mathbf{x}^{(0)}+\lim_{k\to\infty}(\sum_{j=0}^{k-1}T^j)\mathbf{c}$$
$$=0+(I-T)^{-1}\mathbf{c}$$

Thus, $\mathbf{x}=\lim_{k\to\infty}\mathbf{x}^k=(I-T)^{-1}\mathbf{c}$ is the unique solution to $\mathbf{x}=T\mathbf{x}+\mathbf{c}$.

(ii) Now prove that $\rho(T)<1 \Leftarrow \mathbf{x}^{(k)}$ converges to the unique solution $\mathbf{x}$.

If $\mathbf{x}^{(k)}\to\mathbf{x}$ for any $\mathbf{x}^{(0)}$, then from (5.11) $\mathbf{x}=T\mathbf{x}+\mathbf{c}$.

So for each k

$$\mathbf{x}-\mathbf{x}^{(k)}=T\mathbf{x}+\mathbf{c}-(T\mathbf{x}^{(k-1)}+\mathbf{c})=T\left(\mathbf{x}-\mathbf{x}^{(k-1)}\right)=\ldots=T^k\left(\mathbf{x}-\mathbf{x}^{(0)}\right)$$

Hence, $\lim_{k\to\infty}T^k\left(\mathbf{x}-\mathbf{x}^{(0)}\right)=\lim_{k\to\infty}\left(\mathbf{x}-\mathbf{x}^{(k)}\right)=0$.

As $\mathbf{x}^{(0)}$ is arbitrary, $\mathbf{x}-\mathbf{x}^{(0)}$ is also arbitrary. Thus from theorem 4.10, $\rho(T)<1$.

Strictly Diagonally Dominant Matrices

Definition: The $n\times n$ matrix A is said to be strictly diagonally dominant when

$$|a_{ii}|>\sum_{\substack{j=1\\ j\neq i}}^{n}|a_{ij}|$$

holds for each i=1, 2, ………, n.

For example, $A=\begin{bmatrix}7&2&0\\3&5&-1\\0&5&-6\end{bmatrix}$ is strictly diagonally dominant, but $B=\begin{bmatrix}6&4&-3\\4&-2&0\\-3&0&1\end{bmatrix}$ is not.

Theorem 5.2 A strictly diagonally dominant $n\times n$ matrix A is nonsingular.

Proof

Assume that D, L and U are as defined by (5.2), then $A=D-L-U$. As A is strictly diagonally dominant, $D_{ii}\neq 0$ and thus D is nonsingular. So we can construct

$$B_1 = D^{-1}(L+U),$$

and

$$\|B_1\|_\infty = \left\|D^{-1}(L+U)\right\|_\infty = \max_i \left(\sum_{j=1}^{i-1} \left|\frac{a_{ij}}{a_{ii}}\right| + \sum_{j=i+1}^{n} \left|\frac{a_{ij}}{a_{ii}}\right| \right) < 1$$

Hence, $|\lambda| \le \rho(B_1) < \|B_1\|_\infty < 1$.

For any eigenvalue λ of B_1, $1-\lambda$ is an eigenvalue of $I - B_1 = [I - D^{-1}(L+U)]$. Since $\lambda < 1$, it follows that no eigenvalue of $I - B_1$ can be zero and consequently $I - B_1$ is nonsingular.

Therefore, A is a product of two nonsingular matrices D and ($I - B_1$).

$$A = D(I - B_1) = D - L - U.$$

Hence, $\det A = \det D \ \det(I - B_1) \neq 0$ and thus A is nonsingular. □

Sufficient Conditions for Convergence of the Jacobi & Gauss-Seidel Methods

By analyzing the iteration matrices for the Jacobi method, T_j given in equation (5.4), and the Gauss-Seidel method, T_g given in equation (5.10), i.e.

$$T_j = D^{-1}(L+U), \; T_g = (D-L)^{-1}U,$$

we can derive the following sufficient condition for convergence of the Jacobi and Gauss-Seidel methods.

Theorem 5.3 If A is strictly diagonally dominant, then for any choice of $\mathbf{x}^{(0)}$, both the Jacobi and Gauss-Seidel methods give sequences $\left\{\mathbf{x}^{(k)}\right\}_{k=0}^{\infty}$ that converge to the unique solution of $A\mathbf{x} = \mathrm{b}$.

Proof

(1) For Jacobi method – Exercise

(2) For Gauss-Seidel method

$$\|T_g\|_\infty = \max_{\|x\|_\infty = 1} \|\mathbf{y}\|_\infty$$

where $\mathbf{y} = T_g\mathbf{x} = (D-L)^{-1}U\mathbf{x}$

from which $\mathbf{y} = D^{-1}L\mathbf{y} + D^{-1}U\mathbf{x}.$

Assume $y_k = \max_i (y_i),$

$$\|\mathbf{y}\|_\infty = |y_k| \le \sum_{j=1}^{k-1} \left|\frac{a_{kj}}{a_{kk}}\right| \|\mathbf{y}\|_\infty + \sum_{j=k+1}^{n} \left|\frac{a_{kj}}{a_{kk}}\right| \|\mathbf{x}\|_\infty$$

$$= r_k \|\mathbf{y}\|_\infty + s_k \|\mathbf{x}\|_\infty,$$

where $r_k = \sum_{j=1}^{k-1} \left|\frac{a_{kj}}{a_{kk}}\right|, \quad s_k = \sum_{j=k+1}^{n} \left|\frac{a_{kj}}{a_{kk}}\right|.$

Thus, we have form the above formulae

$$\|\mathbf{y}\|_\infty = \frac{s_k}{1-r_k}\|\mathbf{x}\|_\infty,$$

and consequently

$$\|T_g\|_\infty \le \frac{s_k}{1-r_k}.$$

As $1-(s_k+r_k)>0$ for strictly diagonally dominant systems, we have $1-r_k > s_k$ and hence

$$\|T_g\|_\infty < 1$$

$$\Rightarrow \quad \rho(T_g) < 1.$$

This, from Theorem 5.11, guarantees that the sequence converges to the unique solution.

5.4 Error Bound and Speed of Convergence

We now have the following essential issues:

(i) How to estimate the error?
(ii) How many iterations are needed for a given accuracy requirement?
(iii) When are iterative methods preferable to Gaussian elimination methods in solving $A\mathbf{x}=b$?

An error bound can be derived from theorem 5.1 and is summarized by the following corollary.

Corollary If $\|T\|<1$ for any natural matrix norm, then the sequences $\left\{\mathbf{x}^{(k)}\right\}_{k=0}^{\infty}$ in (5.11) converges, for any $\mathbf{x}^{(0)} \in \mathbb{R}^n$, to a vector $\mathbf{x} \in \mathbb{R}^n$, and the following Error bounds hold

$$\left\|\mathbf{x} - \mathbf{x}^{(k)}\right\| \le \|T\|^k \left\|\mathbf{x}^{(0)} - \mathbf{x}\right\|$$

$$\left\|\mathbf{x} - \mathbf{x}^{(k)}\right\| \le \frac{\|T\|^k}{1-\|T\|} \left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\|$$

Remark 1. Since the above formulae hold for any natural matrix norm, it follows from theorem 4.9 that

$$\left\|\mathbf{x}^{(k)} - \mathbf{x}\right\| \approx \rho^k(T)\left\|\mathbf{x}^{(0)} - \mathbf{x}\right\|.$$

Thus, the rate of convergence is essentially $\rho(T)$.

Proof

(a) First prove that $\|T\| < 1 \;\Rightarrow\; \left\|\mathbf{x} - \mathbf{x}^{(k)}\right\| \le \|T\|^k \left\|\mathbf{x}^{(0)} - \mathbf{x}\right\|$.

From the iterative formulae (5.11)

$$\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c},$$

we have

$$\mathbf{x} - \mathbf{x}^{(k)} = \mathbf{x} - T\mathbf{x}^{(k-1)} - \mathbf{c}. \tag{5.12}$$

As $\mathbf{x}$ is the exact solution, we have

$$\mathbf{x} = T\mathbf{x} + \mathbf{c},$$

and thus (5.12) becomes

$$\begin{aligned}\mathbf{x} - \mathbf{x}^{(k)} &= T(\mathbf{x} - \mathbf{x}^{(k-1)}) \\ &= T\left[T(\mathbf{x} - \mathbf{x}^{(k-2)})\right] \\ &= T^2\left[\mathbf{x} - \mathbf{x}^{(k-2)}\right] \\ &= T^{(k)}\left[\mathbf{x} - \mathbf{x}^{(0)}\right].\end{aligned}$$

Hence, $\left\|\mathbf{x} - \mathbf{x}^{(k)}\right\| = \left\|T^{(k)}\left[\mathbf{x} - \mathbf{x}^{(0)}\right]\right\| \le \|T\|^{(k)}\left\|\mathbf{x} - \mathbf{x}^{(0)}\right\|$.

(b) Now prove that $\|T\| < 1 \;\Rightarrow\; \left\|x - x^{(k)}\right\| \le \dfrac{\|T\|^k}{1 - \|T\|}\left\|x^{(1)} - x^{(0)}\right\|$.

As $\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c}$, we have

$$\mathbf{x}^{(n+1)} - \mathbf{x}^{(n)} = T\left(\mathbf{x}^{(n)} - \mathbf{x}^{(n-1)}\right) = T^2\left(\mathbf{x}^{(n-1)} - \mathbf{x}^{(n-2)}\right) = \ldots = T^n\left(\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right).$$

Thus for $n > k \ge 1$,

$$\begin{aligned}\mathbf{x}^{(n)} - \mathbf{x}^{(k)} &= \left(\mathbf{x}^{(n)} - \mathbf{x}^{(n-1)}\right) + \left(\mathbf{x}^{(n-1)} - \mathbf{x}^{(n-2)}\right) + \ldots\ldots\ldots + \left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right) \\ &= T^{n-1}\left[\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right] + T^{n-2}\left[\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right] + \ldots\ldots + T^k\left[\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right] \\ &= \left(I + T + T^2 + \ldots + T^{n-k-1}\right)T^k\left[\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right]\end{aligned} \tag{5.13}$$

from which and the Neumann lemma in section 4.4, we have

$$\lim_{n\to\infty}\left[x^{(n)}-x^{(k)}\right]=\left(\lim_{n\to\infty}\sum_{j=0}^{n-k-1}T^j\right)T^k\left[\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right]=(I-T)^{-1}T^k\left[\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right] \tag{5.14}$$

Hence

$$(I-T)\left(\mathbf{x}-\mathbf{x}^{(k)}\right) = T^k\left(\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right) \tag{5.15}$$

or

$$\left(\mathbf{x}-\mathbf{x}^{(k)}\right)=T\left(\mathbf{x}-\mathbf{x}^{(k)}\right)+T^k\left(\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right) \tag{5.16}$$

Thus

$$\begin{aligned}\left\|\mathbf{x}-\mathbf{x}^{(k)}\right\| &\le \left\|T\left(\mathbf{x}-\mathbf{x}^{(k)}\right)+T^k\left(\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right)\right\| \\ &\le \left\|T\left(\mathbf{x}-\mathbf{x}^{(k)}\right)\right\|+\left\|T^k\left(\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right)\right\| \\ &\le \|T\|\left\|\mathbf{x}-\mathbf{x}^{(k)}\right\|+\left\|T^k\right\|\left\|\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right\|\end{aligned} \tag{5.17}$$

Therefore, $\left\|\mathbf{x}-\mathbf{x}^{(k)}\right\| \le \dfrac{\|T\|^k\left\|\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right\|}{1-\|T\|}$. □

Remark 2. If the initial error is to be reduced to its ε multiple, then

$$\left\|\mathbf{x}-\mathbf{x}^{(k)}\right\| \le \varepsilon\left\|\mathbf{x}-\mathbf{x}^{(0)}\right\|,$$

and thus we require from the corollary

$$\left(\rho(T)\right)^k \le \varepsilon$$

from which the iteration number needed can be determined by

$$k \ge \frac{+\ln\varepsilon}{+\ln\rho} \quad (\ln\rho<0).$$

Example 5.4 Solve a dense linear system by iteration with accuracy up to about six digits.

Solution

Assume that $\mathbf{x}^{(0)}=0$, then we require

$$\frac{\left\|\mathbf{x}-\mathbf{x}^{(k)}\right\|_\infty}{\|\mathbf{x}\|_\infty} \le \varepsilon = 10^{-6}$$

If A has order n, the number of operations (multiplications) per iteration is n^2. To obtain the required accuracy, the necessary number of iterations is

$$k^* = \frac{+\ln\left(10^{-6}\right)}{+\ln\rho} = \frac{6\ln 10}{-\ln\rho},$$

and the number of operations is

$$k^* n^2 = 6\ln 10 \frac{n^2}{-\ln\rho}.$$

If Gaussian elimination is used to solve $A\mathbf{x} = b$, the number of operations is about $\frac{n^3}{3}$. Therefore, the iterative method will be more efficient than the Gaussian elimination method if

$$k^* n^2 < \frac{n^3}{3}, \text{ that is } \quad k^* < \frac{n}{3}.$$

Table 5.2 shows the k^* value corresponding to different ρ values. Obviously, if n is large and ρ is small, the iterative method is more efficient.

Table 5.2

ρ	k*
0.9	131
0.8	62
0.6	27
0.4	15
0.2	9

5.5 Relaxation Method

Alternative Form of the Gauss-Seidel Iterative Formulae

Firstly, we examine the Gauss-Seidel iterative formulae

$$x_i^{(k)} = \frac{1}{a_{ii}}\left[-\sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} + b_i\right] \tag{5.18}$$

and develop an alternative form of the formulae.

Denote $$r_i^{(k)} = \left(r_{1i}^{(k)}, r_{2i}^{(k)}, \ldots\ldots\ldots, r_{ni}^{(k)}\right)$$

as the residual error corresponding to the approximate solution below at the end of the (i-1)th step of the kth iteration cycle

$$x_i^{(k)} = \left(x_1^{(k)}, x_2^{(k)}, \ldots\ldots\ldots, x_{i-1}^{(k)}, x_i^{(k-1)}, \ldots\ldots x_n^{(k-1)}\right)^T.$$

Then, the mth component of $r_i^{(k)}$ is

$$r_{mi}^{(k)} = b_m - \sum_{j=1}^{i-1} a_{mj} x_j^{(k)} - \sum_{j=i+1}^{n} a_{mj} x_j^{(k-1)} - a_{mi} x_i^{(k-1)} \quad . \tag{5.19}$$

For $m=i$, (5.19) becomes

$$r_{ii}^{(k)} = b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} - a_{ii} x_i^{(k-1)} \ ,$$

from which we have

$$x_i^{(k-1)} + \frac{r_{ii}^{(k)}}{a_{ii}} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right].$$

It is noted that the right hand side of the above formula gives $x_i^{(k)}$ of the Gauss-Seidel method as shown by (5.18). Hence, we can determine $x_i^{(k)}$ by

$$x_i^{(k)} = x_i^{(k-1)} + \frac{r_{ii}^{(k)}}{a_{ii}} , \tag{5.20}$$

which is an alternative formula for calculating the ith component of x in iteration k.

Characteristics of Gauss-Seidel Method

From (5.18) and (5.19), the component of residual vector for the ith equation $r_{i(i+1)}^{(k)}$ after the calculation of $x_i^{(k)}$ and $x_i^{(k-1)}$ becomes

$$\begin{aligned} r_{i(i+1)}^{(k)} &= b_i - \sum_{j=1}^{i} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \\ &= \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right] - a_{ij} x_i^{(k)} = 0 \end{aligned}$$

which indicates that the characteristics of Gauss-Seidel method is that *at each step of calculation, one component of the residual vector is reduced to zero.*

Relaxation Method

In general, the basis of all relaxation methods is to calculate the residual vector $r = b - A\mathbf{x}$**,** and to modify (or relax) one or more components of the approximate solution $\mathbf{x}$ in order to reduce to zero one or more components of r. The Gauss-Seidel method is an example of relaxation methods.

Successive Over Relaxation (SOR) Method and Under Relaxation Method

Reducing one component of the residual vector to zero is not generally the most efficient way to reduce the norm of the vector **r**. The procedure (5.20) can be modified by

$$x_i^{(k)} = x_i^{(k-1)} + \omega \frac{r_{ii}^{(k)}}{a_{ii}}. \tag{5.21}$$

For certain choice of positive ω, the speed of convergence of $\mathbf{x}^{(k)} \to \mathbf{x}$ can be accelerated. For choice of ω <1, the procedure are called *under-relaxation methods*, and can be used to obtain convergence of some systems that are not convergent by the Gauss-Seidel method. For choice ω greater than 1, the procedures are called *over-relaxation methods*, which are used to accelerate convergence for systems that are convergent by the Gauss-Seidel technique. These methods are called *Successive Over- Relaxation (SOR).*

The system of equations (5.21) can be written as

$$x_i^{(k)} = (1-\omega)x_i^{(k-1)} + \frac{\omega}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right]$$

or in matrix form

$$(D-\omega L)\mathbf{x}^{(k)} = \left[(1-\omega)D + \omega U\right]\mathbf{x}^{(k-1)} + \omega b,$$

$$\mathbf{x}^{(k)} = (D-\omega L)^{-1}\left[(1-\omega)D + \omega U\right]\mathbf{x}^{(k-1)} + \omega (D-\omega L)^{-1} b.$$

Choose of ω

In order to make $\mathbf{x}^{(k)}$ converge to **x** *as rapidly as possible*, the ω is to be chosen to minimize $\rho(T_\omega)$ where $T_\omega = (D-\omega L)^{-1}\left[(1-\omega)D + \omega U\right]$. Although *no complete answer* to this question is known for our general $n \times n$ linear systems, the following results can be used in certain situations.

Theorem 5.4 If $a_{ii} \neq 0$, then $\rho(T_\omega) \geq |\omega - 1|$. This implies that $\rho(T_\omega) < 1$ only if $0 < \omega < 2$.

Theorem 5.5 (Ostrowski-Reich). If A is a positive definite matrix and $0 < \omega < 2$, then the SOR converges for any choice of initial approximate solution vector $\mathbf{x}^{(0)}$.

Theorem 5.6 If A is positive definite and tri-diagonal, then $\rho(T_g) = \left[\rho(T_j)\right]^2 < 1$ and the optimal choice of ω for the SOR is

$$\omega = \frac{2}{1+\sqrt{1-\rho(T_g)}} \quad \text{and} \quad \rho(T_\omega) = \omega - 1.$$

SOR Algorithm for Solving $A\mathbf{x} = b$

```
Input:   Number of equations N, $A$, b, $\mathbf{x}_{0i}$, $\omega$, Tol, $N_{iter}$
Output:  The solution $\mathbf{x}_i$ or a message that the number of iterations was exceeded.

Step 1:  k ← 1
Step 2:  while (k ≤ $N_{iter}$) do steps 3 to 5
Step 3:          for $i$ =1 to $N$ do
                     $x_i \leftarrow (1-\omega)x_{0i} + \frac{\omega}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}x_{0j}\right]$
Step 4:          if $\|\mathbf{x}-\mathbf{x}_0\| < Tol$ then
                     Output x and the procedure completed successfully
                     STOP
Step 5:          else
                     k ← k+1
                     $\mathbf{x}_{0i} \leftarrow \mathbf{x}_i$
                     go to step 3
Step 6:  Output "maximum number of iterations was exceeded".
         End
```

EXERCISES 5

Q5.1 Find the first two iterations of the Jacobi method for the following system, using $\mathbf{x}^{(0)} = 0$

$$\begin{bmatrix} 10 & -1 & 0 \\ -1 & 10 & -2 \\ 0 & -2 & 10 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 7 \\ 6 \end{bmatrix}$$

Q5.2 Repeat Q5.1 using the Gauss-Seidel method.

Q5.3 Repeat Q5.2 using the SOR method with $\omega = 1.2$.

Q5.4 Write a computer subroutine to implement the SOR iterative scheme for $A\mathbf{x} = b$. Then solve Q5.1 using Tol = 10^{-2}, maximum number of iteration N_{iter}= 25, and $\omega = 0.5$ and $\omega = 1.1$, respectively.

Q5.5 (a) Prove that $\left\|\mathbf{x}^{(k)} - \mathbf{x}\right\| \le \|T\|^k \left\|\mathbf{x}^{(0)} - \mathbf{x}\right\|$ and $\left\|\mathbf{x}^{(k)} - \mathbf{x}\right\| \le \dfrac{\|T\|^k \left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\|}{1-\|T\|}$ where T is an $n \times n$ matrix with $\|T\| < 1$ and $\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c}$ with $\mathbf{x}^{(0)}$ arbitrary ($\in \mathbb{R}^n$) and $\mathbf{x} = T\mathbf{x} + \mathbf{c}$.

(b) Apply the bounds to Q5.1.

Q5.6 Show that if A is strictly diagonally dominant, then $\|T_j\|_\infty < 1$.

Chapter 6 Solution of Nonlinear Systems of Equations

The general form of a system of nonlinear equations is

$$\begin{cases} f_1(x_1, x_2, \dots\dots\dots, x_n) = 0, \\ f_2(x_1, x_2, \dots\dots\dots, x_n) = 0, \\ \vdots \\ f_n(x_1, x_2, \dots\dots\dots, x_n) = 0. \end{cases}$$

Let $\mathbf{f}(\mathbf{x})$ be the vector valued function

$$\mathbf{f}(x_1, x_2, \dots\dots\dots, x_n) = \begin{bmatrix} f_1(x_1, x_2, \dots\dots\dots, x_n) \\ f_2(x_1, x_2, \dots\dots\dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots\dots\dots, x_n) \end{bmatrix}$$

Then the system can be written in vector notation by

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}.$$

In this chapter, we present various methods for solving nonlinear systems. Section 6.1 briefly reviews the concepts of limit and continuity of functions of multi-variables and vector valued functions. Section 6.2 presents the fixed point iteration scheme and its convergence condition, followed by Newton's method in section 6.3 and the modified Newton and quasi Newton methods in section 6.4. In section 6.5, the steepest descent algorithm is presented. Finally in section 6.6, we show how to solve systems of nonlinear equations using Maple and Matlab built-in functions.

6.1 Preliminary

In this section, we briefly review the concepts of limit and continuity for functions of multi-variables and vector valued functions.

Limit and Continuity of Functions of Multi-variables (from $\mathbb{R}^n$ to $\mathbb{R}$)

Let $f(x_1, x_2, \dots\dots, x_n) = 0$ $\quad \mathbf{x} \in D \subset \mathbb{R}^n$

Definition – Limit: f is said to have the limit l at $\mathbf{x}_0$, denoted by $\lim_{\mathbf{x}\to\mathbf{x}_0} f(\mathbf{x}) = l$, if for any given number $\varepsilon > 0$, there exists a number $\delta > 0$ with the property that $|f(\mathbf{x}) - l| < \varepsilon$ whenever $\mathbf{x} \in D$ and $0 < \|\mathbf{x} - \mathbf{x}_0\| < \delta$.

Definition – Continuity: f is said to be continuous at $\mathbf{x}_0 \in D$ provided that $\lim_{\mathbf{x}\to\mathbf{x}_0} f(\mathbf{x})$ exists and $\lim_{\mathbf{x}\to\mathbf{x}_0} f(\mathbf{x}) = f(\mathbf{x}_0)$; and f is said to be continuous on a set D provided that f is continuous at every points of D, expressed by $f \in C(D)$.

Limit and Continuity of Vector-valued Functions (from $\mathbb{R}^n$ to $\mathbb{R}^n$)

Let

$$\mathbf{f}(x_1, x_2, \ldots\ldots\ldots, x_n) = \begin{bmatrix} f_1(x_1, x_2, \ldots\ldots\ldots, x_n) \\ f_2(x_1, x_2, \ldots\ldots\ldots, x_n) \\ \vdots \\ f_n(x_1, x_2, \ldots\ldots\ldots, x_n) \end{bmatrix}$$

Definition - Limit: If and only if all functions f_i have limit, $\lim_{\mathbf{x}\to\mathbf{x}_0} f_i(\mathbf{x}) = L_i$, we said that $\mathbf{f}$ has limit L, denoted by $\lim_{\mathbf{x}\to\mathbf{x}_0} \mathbf{f}(\mathbf{x}) = L = \begin{pmatrix} L_1 & L_2 & \ldots & L_n \end{pmatrix}^T$.

Definition - Continuity: $\mathbf{f}$ is said to be continuous at $\mathbf{x}_0 \in D$ provided that $\lim_{\mathbf{x}\to\mathbf{x}_0} \mathbf{f}(\mathbf{x})$ exists and is equal to $\mathbf{f}(\mathbf{x}_0)$; and $\mathbf{f}$ is continuous on the set D if $\mathbf{f}$ is continuous at every point $\mathbf{x}$ in D, expressed by $\mathbf{f} \in C(D)$.

6.2 Fixed-Point Iterative Methods

Given a nonlinear system in general form $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, we can rewrite the system in the following form (in a verity of ways)

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

Definition - Fixed Point: A solution to system (6.1) is said to be a fixed point of the vector-valued function $\mathbf{g}: D \subset \mathbb{R}^n$ into $\mathbb{R}^n$.

In the following, we study the following two basic problems:

- when will the function $\mathbf{g}(\mathbf{x})$ have a fixed-point?
- how to determine the fixed-point?

Conditions for Existence of Solution

Let $D = \{(x_1, x_2, \ldots\ldots\ldots, x_n)^T \mid a_i \le x_i \le b_i$ for each i =1, 2, …, $n\}$. If the following conditions hold, 1° $\mathbf{g}(\mathbf{x}) \in C(D)$ from $\mathbb{R}^n$ to $\mathbb{R}^n$

2° $\mathbf{g}(\mathbf{x}) \in D, \quad \forall\, \mathbf{x} \in D$

then $\mathbf{g}(\mathbf{x})$ has a fixed-point in *D*.

Fixed-point Iteration Formula

To find the fixed point of (6.1), we set up a fixed point iteration scheme

$$\mathbf{x}^{(k)} = \mathbf{g}(\mathbf{x}^{(k-1)}) \qquad (k \ge 1) \tag{6.2}$$

or by the Gauss-Seidel method

$$x_i^{(k)} = g_i(x_1^{(k)}, x_2^{(k)}, \ldots., x_{i-1}^{(k)},\ \ x_i^{(k-1)}, x_{i+1}^{(k-1)}, \ldots., x_n^{(k-1)}).$$

With the above iterative formulae, we can generate a sequence $\left\{\mathbf{x}^{(k)}\right\}_{k=1}^{\infty}$ from an initial approximation $\mathbf{x}^{(0)}$, and we hope that the sequence converges to the fixed point of $\mathbf{g}(\mathbf{x})$. In the following, we give the conditions for convergence of the sequence, and then discuss the rate of convergence for converging fixed point iteration schemes.

Convergence Conditions

If $\mathbf{g}$ has a fixed-point in *D* and

1° $\mathbf{g}$ has continuous partial derivatives $\dfrac{\partial g_i(\mathbf{x})}{\partial x_j}$ $(i, j = 1, 2, \ldots, n)$

2° there is $K < 1$, such that $\left|\dfrac{\partial g_i(\mathbf{x})}{\partial x_j}\right| \le \dfrac{K}{n}$, whenever $\mathbf{x} \in D$ $(i, j = 1, 2, \ldots, n)$

then, the sequence $\left\{\mathbf{x}^{(k)}\right\}_{k=1}^{\infty}$ generated from an initial estimate $\mathbf{x}_0$ in *D* by the above iterative formula (6.2) converges to the unique fixed point $\mathbf{p} \in D$ with error bound

$$\left\|\mathbf{x}^{(n)} - \mathbf{p}\right\|_\infty \le \frac{K^n}{1-K}\left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\|_\infty .$$

Rate of Convergence

Definition If $\left\{\mathbf{x}^{(k)}\right\}$ converges to x and

$$\lim_{k\to\infty} \frac{\left|\mathbf{x}^{(k+1)} - \mathbf{x}\right|}{\left|\mathbf{x}^{(k)} - \mathbf{x}\right|^{\alpha}} = \lambda ,$$

then we define

α – order of convergence. If α = 1, we have linear convergence,
α = 2 we have quadratic convergence.

λ – asymptotic error constant.

Theorem 6.1 Suppose $\mathbf{p}$ is a solution of $\mathbf{g}(\mathbf{x}) = \mathbf{x}$ for some function $\mathbf{g} = (g_1, g_2, g_3, \ldots, g_n)^T$ mapping $\mathbb{R}^n$ into $\mathbb{R}^n$. If a number $\delta > 0$ exists with the following properties

1° $\dfrac{\partial g_i(\mathbf{x})}{\partial x_j}$ is continuous on $N_\delta = \left\{\mathbf{x} \,\middle|\, \|\mathbf{x}-\mathbf{p}\| < \delta\right\}$ $\quad j, i = 1, 2, \ldots, n$

2° $\dfrac{\partial^2 g_i(\mathbf{x})}{\partial x_j \partial x_k}$ is continuous and $\left|\dfrac{\partial^2 g_i(\mathbf{x})}{\partial x_j \partial x_k}\right| \le M$ for some constant M, $\forall \mathbf{x} \in \mathbb{R}^n$

3° $G(\mathbf{p}) = \dfrac{\partial g_i(\mathbf{p})}{\partial x_j} = \mathbf{0}$.

Then, a number $\hat{\delta} \le \delta$ exists, such that the sequence generated by $\mathbf{x}^{(k)} = \mathbf{g}\left[\mathbf{x}^{(k-1)}\right]$ converges quadratically to $\mathbf{p}$ for any choice of $\mathbf{x}^{(0)}$, provided $\left\|\mathbf{x}^{(0)} - \mathbf{p}\right\| < \hat{\delta}$. Moreover

$$\left\|\mathbf{x}^{(k)} - \mathbf{p}\right\|_\infty \le \frac{n^2 M}{2} \left\|\mathbf{x}^{(k-1)} - \mathbf{p}\right\|_\infty^2 \quad \text{for} \quad k \ge 1.$$

6.3 Newton's Method

Newton's method is an algorithmic procedure to perform the transformation of $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ to a fixed -point problem. The method is based on the Taylor theorem for functions with multi-variables.

Derivation of Newton's Formula

To determine the n-vector satisfying the system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, we expand the ith component function f_i of the vector valued function $\mathbf{f}$ as a Taylor series,

$$f_i(\mathbf{x}) = f_i(\mathbf{x}^{(k)}) + [\nabla f_i(\mathbf{x}^{(k)})]^T \Delta\mathbf{x}^{(k)} + R\left(\left\|\Delta\mathbf{x}^{(k)}\right\|^2\right), (i = 1, n), \tag{6.3}$$

in the case that f_i has continuous 1st and 2nd order partial derivatives, where $\Delta\mathbf{x}^{(k)} = \mathbf{x} - \mathbf{x}^{(k)}$.

Thus
$$\mathbf{f}(\mathbf{x}) = \mathbf{f}\left(\mathbf{x}^{(k)}\right) + \mathbf{f}'\left(\mathbf{x}^{(k)}\right)\Delta\mathbf{x}^{(k)} + \theta\left(\|\Delta\mathbf{x}\|^2\right) \tag{6.4}$$

where
$$\mathbf{f}' = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} = J(\mathbf{x}^{(k)})$$

is called the Jacobian matrix for $\mathbf{f}$ at $\mathbf{x}$.

Considering that if $\mathbf{x}$ is the solution, $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, we can obtain from (6.4) by ignoring the higher order term $\theta\left(\|\Delta\mathbf{x}\|^2\right)$:

$$\begin{cases} J\left(\mathbf{x}^{(k)}\right)\Delta\mathbf{x}^{(k)} = -\mathbf{f}\left(\mathbf{x}^{(k)}\right) \\ \mathbf{x} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}. \end{cases}$$

Hence, we have the following Newton iterative formula

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J\left(\mathbf{x}^{(k)}\right)^{-1}\mathbf{f}\left(\mathbf{x}^{(k)}\right). \tag{6.5}$$

In the following, we study the convergence property of Newton's method.

Order of Convergence

Newton's Formula (6.5) can be written as

$$\mathbf{x}^{(k+1)} = \mathbf{g}(\mathbf{x}^{(k)})$$

with

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} - J^{-1}(\mathbf{x})\mathbf{f}(\mathbf{x}).$$

Thus

$$\begin{aligned}\frac{\partial\mathbf{g}(\mathbf{x})}{\partial x_j} &= \frac{\partial\mathbf{x}}{\partial x_j} - \frac{\partial}{\partial x_j}\left[J^{-1}(\mathbf{x})\mathbf{f}(\mathbf{x})\right] \\ &= \frac{\partial\mathbf{x}}{\partial x_j} - J^{-1}(\mathbf{x})\frac{\partial\mathbf{f}}{\partial x_j} - \frac{\partial J^{-1}(\mathbf{x})}{\partial x_j}\mathbf{f}(\mathbf{x}).\end{aligned}$$

At the root $\mathbf{x} = \alpha$ where $\mathbf{f}(\alpha) = 0$, $J = \dfrac{\partial\mathbf{f}}{\partial x_j}$

$$\begin{aligned}G(\alpha) &= \begin{bmatrix}\dfrac{\partial\mathbf{g}(\mathbf{x})}{\partial x_1} & \cdots & \dfrac{\partial\mathbf{g}(\mathbf{x})}{\partial x_n}\end{bmatrix} \\ &= I - J^{-1}(\alpha)J(\alpha) - \begin{bmatrix}\dfrac{\partial J^{-1}(\alpha)}{\partial x_1} & \dfrac{\partial J^{-1}(\alpha)}{\partial x_2} & \cdots & \dfrac{\partial J^{-1}(\alpha)}{\partial x_n}\end{bmatrix}\mathbf{f}(\alpha).\end{aligned}$$

By noting that

$$\frac{\partial(JJ^{-1})}{\partial x_j} = J\frac{\partial J^{-1}}{\partial x_j} + \frac{\partial J}{\partial x_j}J^{-1} = \frac{\partial I}{\partial x_j} = 0,$$

we have

$$\frac{\partial J^{-1}(\mathbf{x})}{\partial x_j} = -J^{-1}(\mathbf{x})\frac{\partial J(\mathbf{x})}{\partial x_j}J^{-1}(\mathbf{x}).$$

Thus, if $\mathbf{f}(\mathbf{x})$ has the first and second derivatives and $J(\mathbf{x})$ is nonsingular at the root, then

$$G(\alpha) = I - I - 0 = 0.$$

Hence, according to theorem 6.1, the convergence of Newton's method is quadratic.

Remark: difficulties in using Newton's method are :

1° may lack of convergence because of a poor initial guess;

2° the expense of constructing correctly and then solving the Newton equations for the correction $\Delta\mathbf{x}$ *(h)*.

6.4 Modified Newton Method and Quasi Newton Method

Weakness of Newton's Method

Each iteration of Newton's method requires the construction of an $n \times n$ Jacobian matrix $\{J_{ij}\}$ with

$$J_{ij} = \frac{\partial f_i}{\partial x_j}(\mathbf{x}^{(k)})$$

and the solution of an $n \times n$ linear system

$$J(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)}).$$

In most cases, the exact evaluation of J_{ij} is inconvenient or even impossible, and we need to calculate J_{ij} by a finite difference approximation such as

$$J_{ij} = \frac{\partial f_i}{\partial x_j}(\mathbf{x}^{(k)}) = \frac{f_i(\mathbf{x}^{(k)} + h\mathbf{e}_j) - f_i(\mathbf{x}^{(k)})}{h}, \tag{6.6}$$

where h is small in absolute value and $\mathbf{e}_j$ is the jth column of the $n \times n$ identity matrix. Thus, the construction of J requires ($n^2 + n$) scalar functional evaluations for calculating $f_i(\mathbf{x}^{(k)} + h\mathbf{e}_j)$, ($i$,$j$=1,$n$) and $f_i(\mathbf{x}^{(k)})$ (i=1,n). The number of arithmetic operations required for solving the linear systems is in the order of n^3 i.e. $O(n^3)$. Hence, the total amount of computational effort required for just one iteration of Newton's method is $n^2 + n$ scalar functional evaluations plus $O(n^3)$ arithmetic operations. This amount of computation is huge for large n, which limits the application of Newton's method and is the weakness of Newton's method.

In this section, we study two other types of methods, namely ***modified Newton method*** and ***quasi-Newton method*** in an attempt to reduce the amount of computation in each iteration.

Modified Newton Method

In the (k+1)th iteration, Newton's method finds the improved result by

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}, \tag{6.7}$$

where the correction $\Delta\mathbf{x}^{(k)}$ is determined by

$$J(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)}). \tag{6.8}$$

In general, $J(\mathbf{x}^{(k)})$ is a full matrix of order n and thus $O(n^3)$ operations are required to obtain $\Delta\mathbf{x}^{(k)}$. On the other hand, if the iteration is convergent and $\mathbf{f}'(\mathbf{x})$ depends continuously on $\mathbf{x}$, then $J(\mathbf{x}^{(k+m)})$ will differ little from $J(\mathbf{x}^{(k)})$. It is then reasonably to use $J(\mathbf{x}^{(k)})$ in place of $J(\mathbf{x}^{(k+m)})$ for the correction $\Delta\mathbf{x}^{(k)}$ at the (m + k)th step, namely using

$$J(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k+m)} = -\mathbf{f}(\mathbf{x}^{(k+m)}) \quad \text{for} \quad m = 1, 2, \ldots \tag{6.9}$$

until a slowdown in convergence signals that $J(\mathbf{x}^{(k+m)})$ be taken as a more recent Jacobian matrix. This method is called *modified Newton method* and can reduce the computation amount greatly since once $J(\mathbf{x}^{(k)})$ are factored, we can solve for an additional systems with different right hand side at a cost of $O(n^2)$ operations.

Quasi Newton Method

At the (k+1)th iteration of Newton's method, the correction is calculated by

$$\Delta\mathbf{x}^{(k)} = -J^{-1}(\mathbf{x}^{(k)})\mathbf{f}(\mathbf{x}^{(k)}). \tag{6.10}$$

Quasi Newton methods replace the Jacobian matrix (or its inverse) with an approximation matrix that is updated at each iteration. Broyden's method is a widely used Quasi-Newton method. In this method, $\mathbf{x}^{(1)}$ is calculated from $\mathbf{x}^{(0)}$ using Newton's method, and $\mathbf{x}^{(k)}$ $(k \geq 2)$ is calculated as follows.

Being analogy to the secant method for a single nonlinear equation

$$f'(\mathbf{x})(\mathbf{x}_1 - \mathbf{x}_0) = f(\mathbf{x}_1) - f(\mathbf{x}_0),$$

the Broyden's method uses

$$A_1[\mathbf{x}^{(1)} - \mathbf{x}^{(0)}] = \mathbf{f}(\mathbf{x}^{(1)}) - \mathbf{f}(\mathbf{x}^{(0)}), \tag{6.11}$$

which represents n equations in terms of n^2 unknowns a_{ij}, and thus requires $(n-1)^2$ additional equations.

Obviously, equation (6.11) describes the change in $\mathbf{f}$ when A_1 is applied on the vector $\mathbf{x}^{(1)} - \mathbf{x}^{(0)}$, but do not describes how A_1 operates on the other $n-1$ vectors z_i ($i = 1, 2, \ldots, n-1$) orthogonal to $\mathbf{x}^{(1)} - \mathbf{x}^{(0)}$. Thus, we impose

$$A_1 z_i = J[\mathbf{x}^{(0)}]\, z_i \quad \text{whenever } (\mathbf{x}^{(1)} - \mathbf{x}^{(0)}) z_i = 0, \tag{6.12}$$

which means any vector orthogonal to $\mathbf{x}^{(1)} - \mathbf{x}^{(0)}$ is unaffected by the update from $J^{(0)}$ to A_1. Conditions (6.11) and (6.12) gives n^2 equations in term of n^2 unknowns and thus uniquely define A_1 (see Dennis and More, SIAM review, 19, No. 1, 46 – 89) as

$$A_1 = J\left(\mathbf{x}^{(0)}\right) + \frac{\left[F\left(\mathbf{x}^{(1)}\right) - F\left(\mathbf{x}^{(0)}\right) - J\left(\mathbf{x}^{(0)}\right)\left(\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right)\right]\left(\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right)^T}{\left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\|_2^2}. \tag{6.13}$$

This matrix is then used in place of $J(\mathbf{x}^{(1)})$ to determine $\mathbf{x}^{(2)}$:

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} - A_1^{-1}\mathbf{f}\left(\mathbf{x}^{(1)}\right).$$

The method is then repeated to determine $\mathbf{x}^{(3)}$, using A_2 in place of A_1 and with $\mathbf{x}^{(2)}$ and $\mathbf{x}^{(1)}$ in place of $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(0)}$. In general, once $\mathbf{x}^{(i)}$ has been determined, $\mathbf{x}^{(i+1)}$ is computed by

$$A_i = A_{i-1} + \frac{y_i - A_{i-1}s_i}{\|s_i\|_2^2} s_i^T \tag{6.14}$$

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - A_i^{-1}\mathbf{f}\left(\mathbf{x}^{(i)}\right), \tag{6.15}$$

where the notation $y_i = \mathbf{f}\left(\mathbf{x}^{(i)}\right) - \mathbf{f}\left(\mathbf{x}^{(i-1)}\right)$ and $s_i = \mathbf{x}^{(i)} - \mathbf{x}^{(i-1)}$ is introduced into (6.14) to simplify the equation.

If the method is performed as outlined in equation (6.14) and (6.15), the number of scalar function evaluations is reduced from $n^2 - n$ to n (those required for evaluating $F(\mathbf{x}^{(i)})$), but the method still requires $O(n^{(3)})$ calculation to solve the associated $n \times n$ linear system

$$A_i \Delta\mathbf{x}^{(i)} = -\mathbf{f}\left(\mathbf{x}^{(i)}\right). \tag{6.16}$$

Employing the method in this form would not ordinarily be justified because of the reduction to superlinear convergence from the quadratic convergence Newton's method.

A considerable improvement can be incorporated, however, by employing a matrix inversion formula of Sherman and Morrison. This result states that if A is a nonsingular matrix and x and y are vectors, then $A + \mathbf{x}\mathbf{y}^T$ is nonsingular provided $\mathbf{y}^T A^{-1}\mathbf{x} \neq -1$. Moreover, in this case,

$$(A + \mathbf{x}\mathbf{y}^T)^{-1} = A^{-1} - \frac{A^{-1}\mathbf{x}\mathbf{y}^T A^{-1}}{1 + \mathbf{y}^T A^{-1}\mathbf{x}}. \tag{6.17}$$

This formula permits A_i^{-1} to be computed directly from A_{i-1}^{-1}, eliminating the need for a matrix inversion with each iteration. By letting $A = A_{i-1}$, $\mathbf{x} = (y_i - A_{i-1}s_i)/\|s_i\|_2^2$, and $\mathbf{y} = s_i$, formula (6.14) together with (6.17) implies that

$$A_i^{-1} = \left(A_{i-1} + \frac{\mathbf{y}_i - A_{i-1}s_i}{\|s_i\|_2^2} s_i^T \right)^{-1}$$

$$= A_{i-1}^{-1} - \frac{A_{i-1}^{-1}\left(\frac{\mathbf{y}_i - A_{i-1}s_i}{\|s_i\|_2^2} s_i^t \right) A_{i-1}^{-1}}{1 + s_i^t A_{i-1}^{-1}\left(\frac{\mathbf{y}_i - A_{i-1}s_i}{\|s_i\|_2^2} \right)}$$

$$= A_{i-1}^{-1} - \frac{\left(A_{i-1}^{-1}\mathbf{y}_i - s_i \right) s_i^t A_{i-1}^{-1}}{\|s_i\|_2^2 + s_i^t A_{i-1}^{-1}\mathbf{y}_i - \|s_i\|_2^2};$$

so

$$A_i^{-1} = A_{i-1}^{-1} - \frac{\left(s_i - A_{i-1}^{-1}\mathbf{y}_i \right) s_i^t A_{i-1}^{-1}}{s_i^t A_{i-1}^{-1}\mathbf{y}_i}. \tag{6.18}$$

This computation involves only matrix multiplication at each step and therefore requires only $O(n^{(2)})$ arithmetic calculations. The calculation of A_i is bypassed, as the necessity of solving the linear system (6.16).

6.5 The Steepest Descent Method

This method can be used for finding an initial approximation for Newton-based techniques. For the solution of a nonlinear system

$$\begin{aligned} f_1(x_1, x_2, \ldots\ldots\ldots, x_n) &= 0, \\ f_2(x_1, x_2, \ldots\ldots\ldots, x_n) &= 0, \\ &\vdots \\ f_n(x_1, x_2, \ldots\ldots\ldots, x_n) &= 0, \end{aligned}$$

we construct a function from $\mathbb{R}^n$ to $\mathbb{R}$

$$g(x_1, x_2, \ldots\ldots\ldots, x_n) = \sum_{i=1}^{n} f_i^2 .$$

Obviously, the solution of the system is the n-vector with the property that

$$g(x_1, x_2, \ldots\ldots\ldots, x_n) = 0 .$$

The Steepest Descant method is to change x continuously in such a way that the function value of g is gradually reduced toward zero.

Basic Steps

1° Evaluate $g[x^{(0)}]$;
2° Determine a direction from $x^{(0)}$ that results in a decrease in g ;
3° Decide the amount that should be moved in this direction and calculate $x^{(1)}$;
4° Repeat 2° and 3°.

Determine Steepest Descent Direction

Gradient: $\nabla g(\mathbf{x}) = \left(\frac{\partial g}{\partial x_1}, \frac{\partial g}{\partial x_2}, \ldots\ldots\ldots, \frac{\partial g}{\partial x_n} \right)^T$

Directional derivative: $D_v\, g = \nabla g \,.\, l_v$

Steepest Descent direction: $-\nabla g(\mathbf{x})$

$$\therefore \text{ choose } \quad \mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \alpha \nabla g\left[\mathbf{x}^{(0)}\right].$$

Determine α

Principle: α should be chosen so that

$$g\left[x^{(0)} - \alpha \nabla g\left[x^{(0)}\right]\right] := q(\alpha)$$

is minimized and is significantly less than $g(x^{(0)})$.

Theoretical Method: Differentiate *q*(α) with respect to α and then solve a root finding problem to determine α . This method is too costly.

Practical Method: To determine the value of α which minimizes *q*(α), we firstly approximate $q(\alpha)$ by an interpolation polynomial $P(\alpha)$ at the neighborhood of $\mathbf{x}^{(0)}$ that are hopefully close to the minimum value. Then we determine α_c by minimizing *P*(α). For example, to approximate $q(\alpha)$ by a quadratic interpolating polynomial $P(\alpha) = a + b\alpha + c\alpha^2$, firstly we find a γ value such that $q(\gamma) < q(0)$ then use the following three points $\{(\alpha_i, g_i)\}_{i=1}^{3}$ for constructing the interpolating polynomial:

$$\begin{aligned} \alpha_1 &= 0, & g_1 &= q(\alpha_1) \\ \alpha_2 &= \gamma/2, & g_2 &= q(\alpha_2) \\ \alpha_3 &= \gamma, & g_2 &= q(\alpha_3) \end{aligned}$$

By setting $P(\alpha_i) = g_i$, we can determine *a, b* and *c* and hence obtain

$$P(\alpha) = g_1 + b\alpha + c\alpha^2,$$

where $b = (-3g_1 + 4g_2 - g_3)/\gamma$, $c = (2g_1 - 4g_2 + 2g_3)/\gamma^2$.

Thus, the critical α_c, which minimize $P(\alpha)$ ($q(\alpha)$) can be obtained by

$$\frac{\partial P(\alpha)}{\partial \alpha} = b + 2c\alpha = 0 \qquad \Rightarrow \quad \alpha_c = -\frac{b}{2c}.$$

6.6 Solution of Nonlinear Systems using Maple/MATLAB

Solution of Nonlinear System of Equations using Maple

The Maple **`fsolve`** function can be used to solve nonlinear systems of equations. The syntax is

```
fsolve({eq1,eq2,…,eqn},{x1,x2,…,xn} [,options])
```

Example 6.1 . Solve the following system for roots in $D = \{(x, y): 0 \le x \le 2, 0 \le y \le 2\}$

$$x^2 - y = 1$$
$$x + y^2 = 2$$

The Maple commands

```
> eq1:=x^2-y=1
> eq2:=x+y^2=2
> fsolve({eq1,eq2},{x,y},{x=0..2, y=0..2})
```

give $\{x = 1.345089393, y = 0.8092654740\}$

Example 6.2 . Solve the following system using initial guess (x1,x2,x3)=(0,0,0)

$$x_1^2 + x_2 - 37 = 0$$
$$x_1 - x_2^2 - 5 = 0$$
$$x_1 + x_2 + x_3 - 3 = 0$$

The Maple commands

```
> eq1:=x1^2+x2-37=0
> eq2:=x1-x2^2-5=0
> eq3:=x1+x2+x3-3=0
> fsolve({eq1,eq2,eq3},{x1=0,x2=0,x3=0})
```

give `{x1=6, x2=1, x3=-4}`

Solution of Nonlinear Systems of Equations using MATLAB

The Matlab **fsolve ( )** function can be used to find roots (zeros) of a system of nonlinear equations **f**(**x**)=**0** where **x** is an n-component vector and $\mathbf{f}(\mathbf{x}) = \left(f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})\right)$ is a vector valued function. The syntax is

```
[x, fval [,option]] = fsolve(@fun, x0 [,options])
```

where	`fun`	: the vector valued function **f**(**x**) of the nonlinear system of equations to solve.
	x0	: initial guess of the solution vector.
	x	: solution vector of the nonlinear system
	fval	: the value of the vector valued function **f**(**x**) at the solution **x**

[,options] are optional arguments .

Example. Solve the following system for roots in $D = \{(x, y): 0 \le x \le 2, 0 \le y \le 2\}$

$$\begin{aligned} x^2 - y &= 1 \\ x + y^2 &= 2 \end{aligned} .$$

Solution.

First, we rewrite the system as

$$\begin{aligned} f_1(x) &= x^2 - y - 1 = 0 \\ f_2(x) &= x + y^2 - 2 = 0 \end{aligned}$$

or $\mathbf{f} = (x^2 - y - 1, x + y^2 - 2)^T = \mathbf{0}$ and choose an initial guess **x**0 = [1 0].

Then write a Maple program to define the vector valued function **f**(**x**) via a Maple function fun(x), set up the initial guess of the solution vector, call **`optimset`**`()` to set option for displaying results in each iteration cycle, and then call **`fsolve`**`()` to solve the nonlinear system **f**(**x**)=**0** to get the solution vector **x** and the corresponding value of **f**(**x**) via x and fval. The program is as follows

```
function  F = fun(x)
F =[ x(1).^2-x(2)-1; x(1)+x(2).^2-2];
>> x0 = [1, 0];
>> option = optimset('Display', 'iter');
>> [x, fval] = fsolve(@fun, x0, option)
```

which gives

```
                                         Norm of     First-order    Trust-region
 Iteration  Func-count        f(x)          step       optimality       radius
     0          3                  1                            1            1
     1          6           0.332274             1           1.85            1
     2          9         0.00109938      0.186226         0.0913          2.5
     3         12       1.17936e-008     0.0115633       0.000311          2.5
     4         15       1.88824e-018  3.73868e-005       3.72e-009         2.5
Optimization terminated: first-order optimality is less than options.TolFun.

x =

   1.3451    0.8093

fval =

  1.0e-008 *

   0.1374
   0.0024
```

Note: In the above program, x is a one-dimensional array with x[1] representing variable x, and x[2] representing y.

EXERCISE 6

Q6.1 For the nonlinear system

$$x_1^2 - 10x_1 + x_2^2 + 8 = 0$$

$$x_1x_2^2 + x_1 - 10x_2 + 8 = 0$$

a) transform the system into a fixed-point problem and hence find a fixed point G
b) show that G has a unique fixed point

$$D = \left\{(x_1,\, x_2)^T \mid 0 \le x_1,\ x_2 \le 1.5\right\}$$

c) perform two fixed-point iterations of the Gauss-Seidel method to approximate the solution of the system using $\mathbf{x}^{(0)} = (0,\, 0)^T$.

Q6.2 For system

$$\begin{cases} x_1 + x_2 + x_3 = 0 \\ x_1^2 - x_2 - 1 = 0 \\ x_1x_3^2 - x_2^4 = 0 \end{cases}$$

using $\mathbf{x}^{(0)} = (4,\, -2,\, -2)^T$,

a) apply two iterations of Newton's method
b) apply two iterations of Broyden's method .

Q6.3 Using the method of Steepest Descent with *Tol* = 0.005 to approximate the solution of the system

$$3x_1^2 - x_2^2 = 0$$
$$3x_1x_2^2 - x_1^3 - 1 = 0$$

using $\mathbf{x}^{(0)} = (1, 1)^T$.

Q6.4 Write a computer subroutine for solving the nonlinear system in Q6.3 using the Steepest Descent algorithm.

Q6.5 Write a computer subroutine for solving the nonlinear system in Q6.2 using the Newton's method.

CHAPTER 7

Interpolation and Polynomial Approximation

Concept of Interpolation

For a given set of data points $\{(x_i, y_i)\}_{i=0}^{n}$, we can select a function *p(x)* in such a way that the graph of $y = p(x)$ passes through the points. With $p(x)$, we can then estimate the unknown values of $y(x)$ at any value of x (not only at the specified points). This numerical technique is called interpolation and $p(x)$ is called an interpolating function.

Remarks:

- Interpolation has a wide range of applications, such as finding intermediate values in tables and replacing functions by simpler functions for differentiation and integration.
- Given a set of data points $\{(x_i, y_i)\}_{i=0}^{n}$, the interpolation function $p(x)$ is usually not unique. In this case, one should choose the best function to fit the data.

Polynomial Interpolation

One of the most important and popular interpolations is polynomial interpolation. This is because the derivatives and integrals of polynomials are easy to determine. Another more important reason is that for any continuous function, there exists a polynomial that is as 'close' to the given function as desired, as implied by the following theorem.

Weierstrass Approximation Theorem:

If f is defined and continuous on $[a, b]$, then for any ε > 0, there exists a polynomial p_n, defined on $[a, b]$, with the property that

$$|f(x) - P_n(x)| < \varepsilon \text{ for all } x \in [a,b].$$

Proof: (see Bartle "The element of Real Analysis", p165-172).

Scope of this chapter

Given a continuous function $f(x)$, one could establish a Taylor polynomial of degree $P_n(x)$ for the function at a point x_0, as given in section 1.1. However, the Taylor polynomial agrees with the function only at the point of expansion $(x_0, f(x_0))$ and the approximation error magnifies with distance from this point. Thus, the use of Taylor polynomials is limited to the situation in which approximation is needed only in a small region around the point x_0. In this chapter we learn how to construct interpolating polynomials which exactly agree with a collection of data and thus provide a relatively accurate approximation over an entire interval instead of near a point only.

7.1 Lagrange Polynomials

In this section, we consider the construction of a polynomial that passes through a given set of data points.

Problem: Let $\{x_{o,}x_1,......x_n\}$ be distinct numbers,

$\{y_{o,}y_1,......y_n\}$ be associated function values.

Find a polynomial $P_n(x)$ of degree n that passes through the $(n+1)$ data points, i.e.:

$P(x_i)=y_i, (i=0,n)$.

Now the **key questions** are: *Does such polynomial exist? If so, How to construct it? Is it unique?*

Uniqueness: If there exists a polynomial of degree *n*, $P_n(x)$, that passes through $\{(x_i,y_i)\}_{i=0}^{n}$, then it is unique.

Proof: (By contradiction)

Suppose $P_n(x)$ is not unique, then there exists another polynomial $Q(x)$ with degree at most n satisfying the above conditions. Then $P_n(x)-Q(x)$ is a polynomial of degree n with $(n+1)$ zeros since

$$P_n(x_i)-Q(x_i)=y_i-y_i=0 \ \ (i=0 \text{ to } n),$$

which is impossible as a polynomial of degree n has n roots only. Thus, the uniqueness is proved.

□

Existence: There exists a polynomial of degree n passing through the $(n+1)$ data points and is given by

The nth Lagrange interpolation polynomial

$$P_n(x)=\sum_{k=0}^{n} y_k l_k(x),$$

where l_k $(k=0,1,.., n)$ are basic Lagrange interpolating polynomials defined by

$$l_k(x)=\prod_{\substack{i=0,\\ i\neq k}}^{n}\frac{(x-x_i)}{(x_k-x_i)}.$$

Proof Firstly, we note that l_k has the property that

$$l_k(x_i)=\begin{cases}1 & i=k\\ 0 & i\neq k\end{cases}$$

Thus,

$$P_n(x_i)=y_0l_0(x_i)+y_1l_1(x_i)+...+y_il_i(x_i)+...+y_nl_n(x_i)=y_i \quad (i=0, 1,...n),$$

which indicates that the graph $y=P_n(x)$ passes through the points (x_i, y_i) ($i=0,1, ...n$). □

Example 7.1 Fit a line to (x_0, y_0), (x_1, y_1)

Solution $n = 1,\quad P_1(x) = \left(\dfrac{x - x_1}{x_0 - x_1}\right) y_0 + \left(\dfrac{x - x_0}{x_1 - x_0}\right) y_1$.

Error Estimate:

Polynomial interpolation usually can only yield approximate results. The following theorem is for estimating the error or the remainder term.

Theorem 7.1 (Lagrange polynomial error formula) If $x_0, x_1, \ldots\ldots x_n$ are distinct numbers in the interval $[a, b]$ and $f \in C^{n+1}[a, b]$, then for each x in $[a, b]$, a number $\xi(x)$ in $[a, b]$ exists with

$$f(x) = P_n(x) + \frac{f^{(n+1)}[\xi(x)]}{(n+1)!} \prod_{i=0}^{n} (x - x_i),$$

where P_n is the nth Lagrange interpolation polynomial.

Notes: The reminder term above has its practical use restricted to those functions for whose derivatives there are well-known bounds.

Example 7.2 $f(x) = e^{-x}$. Assume using linear interpolation using (0, 1), $(1, e^{-1})$. Find the maximum error for $x \in [0,1]$.

$$\text{Error} = \frac{f''(\xi)}{2}(x - x_0)(x - x_1)$$

$$\max|error| < \frac{1}{2} \max_{x \in [0,1]} |(x-0)(x-1)| \max_{\xi(x) \in [0,1]} \left|e^{-\xi(x)}\right| = \frac{1}{2} \max_{x \in [0,1]} (x^2 - x) \bullet 1 = \frac{1}{2} \bullet \frac{1}{4} = \frac{1}{8}.$$

Limitation:

Using Lagrange polynomial interpolation, it is very inconvenient to pass from one interpolation polynomial to another of degree one greater.

To avoid this problem, we can use the more general methods that utilize finite difference. Thus, in the following section, we introduce the concept of finite difference. In section 7.3, we introduce Newton interpolating polynomial.

7.2 Finite Difference

In this section, we first define the shift operator and various difference operations, then demonstrate how to calculate forward/backward and central differences by using difference table. Finally, we give several relations between operators.

Shift and Difference Operators

For a given sequence $f_i = f(x+ih)$, we define various operators as follows:

shift operator E	:	$\mathrm{E}\, f(x) = f(x+h)$
forword difference operator Δ	:	$\Delta f(x) = f(x+h) - f(x)$
backword difference operator ∇	:	$\nabla f(x) = f(x) - f(x-h)$
central difference operator δ	:	$\delta f(x) = f\left(x+\dfrac{h}{2}\right) - f\left(x-\dfrac{h}{2}\right)$

Remarks: Higher order operators can be defined as follows:

$$E^k f(x) = f(x+kh),$$
$$\Delta^{k+1} = \Delta(\Delta^k).$$

One of the convenient ways for calculating differences of different orders is by using difference tables, as detailed below.

Calculation of Forward Differences

$$\begin{array}{cccccc}
 & x_0 & f_0 & & & \\
h & & & \Delta f_0 = f_1 - f_0 & & \\
 & x_1 & f_1 & & \Delta^2 f_0 = \Delta f_1 - \Delta f_0 & \\
h & & & \Delta f_1 = f_2 - f_1 & & \Delta^3 f_0 = \Delta^2 f_1 - \Delta^2 f_0 \\
 & x_2 & f_2 & & \Delta^2 f_1 = \Delta f_2 - \Delta f_1 & \\
h & & & \Delta f_2 = f_3 - f_2 & & \\
 & x_3 & f_3 & & &
\end{array}$$

Calculation of Backward Differences

$$\begin{array}{cccccc}
 & x_0 & f_0 & & & \\
h & & & \nabla f_1 = f_1 - f_0 & & \\
 & x_1 & f_1 & & \nabla^2 f_2 = \nabla f_2 - \nabla f_1 & \\
h & & & \nabla f_2 = f_2 - f_1 & & \nabla^3 f_3 = \nabla^2 f_2 - \nabla^2 f_1 \\
 & x_2 & f_2 & & \nabla^2 f_3 = \nabla f_3 - \nabla f_2 & \\
h & & & \nabla f_3 = f_3 - f_2 & & \\
 & x_3 & f_3 & & &
\end{array}$$

Calculation of Central Differences

$$
\begin{array}{cllllll}
 & x_0 & f_0 & & & & \\
h & & & \delta f_{1/2} = f_1 - f_0 & & & \\
 & x_1 & f_1 & & \delta^2 f_1 = \delta f_1 - \delta f_0 & & \\
h & & & \delta f_{3/2} = f_2 - f_1 & & \delta^3 f_{3/2} = \delta^2 f_1 - \delta^2 f_0 & \\
 & x_2 & f_2 & & \delta^2 f_2 = \delta f_2 - \delta f_1 & & \delta^4 f_2 = \delta^3 f_1 - \delta^3 f_0 \\
h & & & \delta f_{5/2} = f_3 - f_2 & & \delta^3 f_{5/2} = \delta^2 f_2 - \delta^2 f_1 & \\
 & x_3 & f_3 & & \delta^2 f_3 = \delta f_3 - \delta f_2 & & \\
h & & & \delta f_{7/2} = f_4 - f_3 & & & \\
 & x_4 & f_4 & & & &
\end{array}
$$

Example 7.3 Construct a finite difference table for $f(x) = x^2 + x$ using $h = 1$.

x	$f(x)$	Δ	Δ^2	Δ^3
-1	0			
		0		
0	0		2	
		2		0
1	2		2	
		4		0
2	6		2	
		6		
3	12			

Notes: Δ^2 is constant, for polynomial of degree 2. In general if

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots\ldots a_1 x + a_0$$

then

1) $\Delta^n P_n = a_n n! h^n$, where h is the step length in table,
2) $P_n^{(n)}(x) = a_n n!$,

Relations between Operators

(1) $\Delta = E - 1$, or $E = 1 + \Delta$

Proof: $\Delta f(x) = f(x+h) - f(x) = Ef(x) - f(x) = (E-1)f(x)$ □

(2) $\nabla = 1 - E^{-1}$, or $E = (1-\nabla)^{-1}$

Proof: Use the inverse of E: $E^{-1}Ef(x) = f(x)$

then $E^{-1}f(x+h) = f(x)$,

thus $E^{-1}f(x) = f(x-h)$.

As $\nabla f(x) = f(x) - f(x-h) = f(x) - E^{-1}f(x) = (1-E^{-1})f(x)$,

we have $\nabla = (1-\mathrm{E}^{-1})$. □

(3) $\delta = E^{\frac{1}{2}} - E^{-\frac{1}{2}}$, or $\delta E^{\frac{1}{2}} = E - 1$

Proof: $\delta f(x) = f\left(x + \frac{h}{2}\right) - f\left(x - \frac{h}{2}\right) = \left(E^{\frac{1}{2}} - E^{-\frac{1}{2}}\right) f(x)$,

Thus $\delta = E^{\frac{1}{2}} - E^{-\frac{1}{2}}$ and $\delta E^{\frac{1}{2}} = E - 1$. □

From (1) and (3), the following formulae can be derived:

$$\Delta^2 = (E-1)^2 = E^2 - 2E + 1 \qquad \Rightarrow \Delta^2 f_0 = f_2 - 2f_1 + f$$

$$\delta^2 = \left(E^{\frac{1}{2}} - E^{-\frac{1}{2}}\right)^2 = E^{-1} - 2 + E \qquad \Rightarrow \delta^2 f_0 = f(x-h) - 2f(x) + f(x+h)$$

As $\qquad Ef(x) = f(x+h) = f(x) + hDf(x) + \frac{(hD)^2}{2!} f(x) + \dots$

we have

$$E = e^{hD} = 1 + hD + \frac{(hD)^2}{2} + \dots$$

from which

$$hD = \ln E = \ln(1+\Delta) = \Delta - \frac{\Delta^2}{2} + \frac{\Delta^3}{3} + \dots$$

Propagation of Round-off Error in a Difference Table

Function values are given to a fixed number of decimal places. We assume, in general, that these are obtained by rounding procedure so that at most each function value has an error (δf) of $\pm\frac{1}{2}$ unit in the last decimal place. At worst, the errors (expressed as units in last decimal place) will propagate as follows:

δf	Δ	Δ^2	Δ^3	Δ^4
$+1/2$				
	-1			
$-1/2$		$+2$		
	$+1$		-4	
$+1/2$		-2		$+8$
	-1		$+4$	
$-1/2$		$+2$		-8
	$+1$		-4	
$+1/2$		-2		
	-1			
$-1/2$				

Obviously, the maximum error in the nth difference $= 2^{n-1}$ units in the last decimal place.

Remarks

(i) Differences are said to have converged when they oscillate within the limits of $\pm 2^{n-1}$ unit of the last decimal place in the n^{th} column of differences (nth order differences).

(ii) All differences from the one that has converged onwards should be neglected, they should not be used in interpolation.

7.3 Newton Interpolation Formulae

The forward, backward and central differences can be used to construct interpolation formulae for tables in which the abscissas $\{x_i\}$ are evenly spaced. In addition, the differences can be used to determine the maximum degree of interpolation polynomial that can be used safely, based on the accuracy of the table entries. Finite differences can be used to defeat noise in data, when the noise is large with respect to the rounding errors or uncertainty errors of physical measurements.

Newton-Gregory Forward Formula

Given $\{x_0, \quad x_1, \quad ...... \quad x_n\}, \quad x_k = x_0 + kh, \quad k = 0, 1, 2.....$

$\{f_0, \quad f_1, \quad ...... \quad f_n\}$

Find $f(x_s)$ where $x_s = x_0 + sh \ \ (s \neq \text{integer})$.

From the Binomial theorem, we have

$$f_s = E^s f(x_0) = (1+\Delta)^s f(x_0) = \left[1 + \binom{s}{1}\Delta + \binom{s}{2}\Delta^2 + \binom{s}{3}\Delta^3 + ...\right] f(x_0),$$

where $\displaystyle\binom{u}{k} = \frac{u(u-1)(u-2)[u-(k-1)]}{k!}$ is the binomial coefficient.

Neglecting differences of order higher than n, we obtain

the nth Degree Newton-Gregory Forward Interpolating Polynomial

$$P_n(x) = P_n(x_0 + sh) = f_0 + s\Delta f_0 + \binom{s}{2}\Delta^2 f_0 + ... + \binom{s}{n}\Delta^n f_0$$

Error in Interpolation

As proved in section 7.1, the nth degree polynomial passing through (n+1) given data points $\{(x_i, y_i)\}_{i=0}^{n}$ is unique. Thus, the nth degree Newton-Gregory interpolating polynomial must be identical to the nth degree Lagrange interpolating polynomial. Consequently, the error in the Newton polynomial must be the same as that in the Lagrange polynomial and so the error formula for Lagrange polynomial (in section 7.1) can be used for the Newton polynomial:

$$\text{Error} = f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{i=0}^{n}(x - x_i)$$

where $\xi \in [x_0, x_n]$. As the interpolation points are equidistant, we have

$$x_k = x_0 + kh\ , \qquad x = x_s = x_0 + sh$$

Substituting the above into the error estimate formula yields

The Newton-Gregory Interpolating Polynomial Error Formula

$$E(x_s) = f^{(n+1)}(\xi_s)\frac{h^{n+1}}{(n+1)!}\prod_{i=0}^{n}(s-i) = \binom{s}{n+1}h^{n+1}f^{(n+1)}(\xi_s)$$

Remarks: If $f^{(n+1)}$ is fairly constant, then $E(x_s) \approx \binom{s}{n+1}\Delta^{n+1}f_0$

which is the 1st neglected term in the Newton-Gregory Forward formula.

Example 7.4 Find $e^{0.12}$ using the following difference table.

Solution First construct the difference table below.

	x	e^x	Δ	Δ^2	Δ^3	Δ^4
	0.0	1.000000				
			0.105171			
$x_0 \rightarrow$	$\underline{0.1}$	$\underline{1.105171}$ →		0.011061		
x_s			$\underline{0.116232}$ →		0.001163	
	0.2	1.221403		$\underline{0.012224}$ →		0.000123
			0.128456		$\underline{0.001286}$ →	
	0.3	1.349859		0.013510		$\underline{0.000134}$
			0.141966		0.001420	
	0.4	1.491825		0.014930		0.000152
			0.156896		0.001572	
	0.5	1.648721		0.016502		
			0.173398			
	0.6	1.822119				

We take $x_0 = 0.1$. Then since $h = 0.1$ and $x{=}0.12 = x_0 + sh$, we have

$$s = \frac{0.12 - 0.1}{0.1} = 0.2$$

and

$$e^{0.12} \approx f_0 + s\Delta f_0 + \binom{s}{2}\Delta^2 f_0 + \binom{s}{3}\Delta^3 f_0 \ldots\ldots$$

$$= 1.105171 + (0.2)(0.116232) + \frac{(0.2)(0.2-1)}{2}(0.012224) + \frac{(0.2)(0.2-1)(0.2-2)}{6}(0.001286) + ..$$

$$= 1.127501$$

$$\text{Error } \approx \binom{s}{4}\Delta^4 f_0 = \frac{(0.2)(-0.8)(-1.8)(-2.8)}{4!}(0.000134) = -0.000005$$

$\therefore$ Answer is correct to 4D; $e^{0.12} \approx 1.1275$

Exercise. Solve the problem with $x_0 = 0.0$.

Newton-Gregory Backward Formula

Obviously it is inconvenient to use the Forward formula near the end of a table. In this case, we should use the following backward formula.

From $E = (1-\nabla)^{-1}$ and $E^s = (1-\nabla)^{-s}$, we have

$$\boxed{\begin{aligned} f(x_0+sh) &= E^s f_0 = (1-\nabla)^{-s} f_0 = \left[1 + s\nabla + \frac{s(s+1)}{2!}\nabla^2 + \frac{s(s+1)(s+2)}{3!}\nabla^3 + \ldots\ldots\right] f(x_0) \\ &= \sum_{k=0}^{n} (-1)^k \binom{-s}{k} \nabla^k f(x_0) \end{aligned}}$$

with error

$$\boxed{E(x_s) = \frac{f^{(n+1)}(\xi_s)}{(n+1)!}\prod_{i=0}^{n} h(s+i) = (-1)^{n+1}\binom{-s}{n+1} h^{n+1} f^{(n+1)}(\xi)}$$

Example 7.5 In the table of example 7.4, find $e^{0.52}$.

Solution Choose $x_0 = 0.6$, $h = 0.1$ then $s = -0.8$.

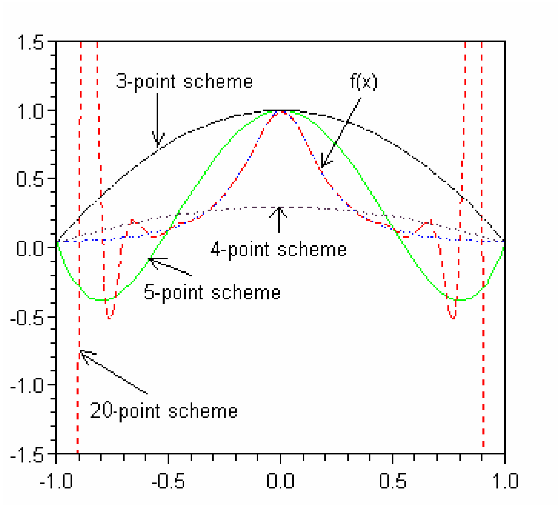
$$\begin{aligned} e^{0.52} &\approx f_0 + s\nabla f_0 + \frac{s(s+1)}{2!}\nabla^2 f_0 + \frac{s(s+1)(s+2)}{6}\nabla^3 f_0 \ldots\ldots \\ &= 1.822119 - (0.8)(0.173398) - (0.08)(0.016502) - (0.032)(0.001572) \\ &= 1.682030 \end{aligned}$$

7.4 Piecewise Interpolation and Cubic Splines

Motivation of Piecewise Interpolation

As the number of interpolating points increases, the degree of the interpolating polynomial increases. However, high degree polynomials tend to be very wiggly and thus using one polynomial to fit all the data points may cause highly fluctuation or divergence in some region.

Eg. For $f(x)=\dfrac{1}{1+25x^2}$ defined in [-1, 1], the graphs of the function and its interpolating polynomials of degree 2, 3, 4 and 19 are shown in the diagram below. It is clear that for $0.726<|x|<1$, $P_n(x)$ diverges as $n\to\infty$.



Why does error increase with n ?

Let $w(x)=\prod_{i=0}^{n}(x-x_i)$, then $R_n(x)=\dfrac{f^{(n+1)}(\xi)}{(n+1)!}w(x)$

For some functions, $f^{(n+1)}(\xi)$ increases faster than $(n+1)!$ with increasing n and thus the error increases.

An alternative approach that avoids the above problem is to divide the interval into a collection of subintervals and construct a (generally) different approximating polynomial on each subinterval. Approximation by functions of this type is called **piecewise polynomial approximation.**

The simplest piecewise polynomial approximation is piecewise linear interpolation that consists of joining a set of data points

$$\{(x_0,f(x_0)),(x_1,f(x_1)),\ldots\ldots(x_n,f(x_n))\}$$

by a series of straight lines. A disadvantage of this approximation is that at each of the endpoints of the subinterval, there is no assurance of differentiability, which, in a geometrical context, means that the interpolating function is not smooth at these points.

Another piecewise interpolation is piecewise quadratic interpolation, which consists of joining the set of data points $\{(x_i,f(x_i))\}_{i=0}^{n}$ by a series of parabolas. The interpolating function can thus have continuous derivatives on $[x_0, x_n]$, but may give large oscillation and high local curvature.

The cubic spline interpolation is the most common piecewise polynomial approximation using cubic polynomials between each successive pair of nodes. A general cubic polynomial involves four constants. So there is sufficient flexibility in the cubic spline procedure to ensure

- Not only that the interpolation is continuously differentiable on the interval, but also that it has a continuous 2nd order derivative on the interval.
- The construction of the cubic spline does not, however, assume that the derivatives of the interpolation agree with those of the function, even at the nodes.

Problem Description

Given points $(x_1, y_1), (x_2, y_2), \ldots\ldots (x_n, y_n)$, find in each of the n-1 intervals $(x_i, x_{i+1})_{i=1,n-1}$ a cubic polynomial

$$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{7.1}$$

which satisfies the following conditions

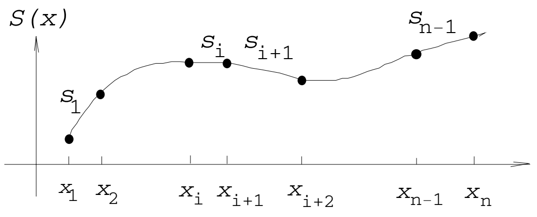
(a) $s_i(x_i) = y_i$ $(i = 1, 2, \ldots, n-1)$ - spline passes the left hand knot of the interval.

(b) $s_i(x_{i+1}) = y_{i+1}$ $(i = 1, 2, \ldots\ldots n-1)$ - spline passes the right hand knot of the interval.

(c) $s_i'(x_{i+1}) = s_{i+1}'(x_{i+1})$ $(i = 1, 2, \ldots, n-2)$ - gradients match at internal knots.

(d) $s_i''(x_{i+1}) = s_{i+1}''(x_{i+1})$ $(i = 1, 2, \ldots, n-2)$ - curvatures match at internal knots.

where s_i is a cubic polynomial defined on the subinterval $[x_i, x_{i+1}]$.



Obviously, there are $(4n-4)$ coefficients to be determined, but the constraints (a, b) and the continuity conditions (c) and (d) only give $4n-6$ equations. Thus there are at least two degrees of freedom in choosing the coefficients of (7.1).

These two extra equations can come from specifying the behaviors of the end points. Different choices of these end conditions lead to different types of splines:

(i)	$s''(x_1) = s''(x_n) = 0$	free or natural splines;
(ii)	$s'(x_1) = f'(x_1),\ s'(x_n) = f'(x_n)$	clamped splines.

Determination of a_i, b_i, c_i and d_i

First, introduce the simpler notation

$$h_i = x_{i+1} - x_i, \quad \lambda_i = s_i''(x_i)$$

For the free cubic spline, we can derive (from conditions *a*-*d* and *i*) the following linear system

$$\begin{bmatrix} 1 & 0 & 0 & \cdot & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & \mathbf{0} & \\ & h_2 & 2(h_2+h_3) & h_3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & \mathbf{0} & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ & & & \cdot & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{n-1} \\ \lambda_n \end{bmatrix} = 6 \begin{bmatrix} 0 \\ \dfrac{y_3-y_2}{h_2} - \dfrac{y_2-y_1}{h_1} \\ \vdots \\ \dfrac{y_n-y_{n-1}}{h_{n-1}} - \dfrac{y_{n-1}-y_{n-2}}{h_{n-2}} \\ 0 \end{bmatrix}$$

for the determination of λ_i. The coefficient matrix is tridiagonal and strictly diagonally dominant and thus the linear system can be solved with no pivoting in about $5n$ operations.

Once λ_i are obtained, the coefficients of each cubic polynomial can be determined by

$$\begin{cases} a_i = y_i \\ b_i = \dfrac{y_{i+1}-y_i}{h_i} - \dfrac{\lambda_{i+1}+2\lambda_i}{6} h_i \\ c_i = \dfrac{\lambda_i}{2} \\ d_i = \dfrac{\lambda_{i+1}-\lambda_i}{6h_i} \end{cases}$$

Thus, we obtain the cubic spline which is defined by $(n-1)$ cubic polynomials

$$s_i = a_i + b_i(x-x_i) + c_i(x-x_i)^2 + d_i(x-x_i)^3, \quad (i = 1, 2, \ldots n-1),$$

each of which is for one interval between two successive nodes. For example, $s_i(x)$ is the approximation to $y(x)$ on the interval (x_i, x_{i+1}).

Example 7.6 Given $f(2.2) = 0.520$, $f(2.4) = 0.510$, $f(2.6) = 0.481$, establish a cubic spline.

Solution From the information given, we know

$$x1 = 2.2, \quad y1 = 0.520,$$
$$x2 = 2.4, \quad y2 = 0.510,$$
$$x3 = 2.6, \quad y3 = 0.481,$$
$$h1 = h2 = 0.2.$$

Thus, the system of equations for λ_1, λ_2 and λ_3 is

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.2 & 2(0.2+0.2) & 0.2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = 6 \begin{bmatrix} 0 \\ \dfrac{0.481-0.510}{0.2} - \dfrac{0.510-0.520}{0.2} \\ 0 \end{bmatrix}$$

$$\Rightarrow \quad \lambda_1 = \lambda_3 = 0, \quad \lambda_2 = -0.7125\,.$$

Spline s_1 for the interval $[x_1, x_2]$:

$x1=2.2$

$$a_1 = y_1 = 0.520, \quad b_1 = \frac{y_2 - y_1}{h_1} - \frac{\lambda_2 + 2\lambda_1}{6} h_1 = -0.02625,$$

$$c_1 = \frac{\lambda_1}{2} = 0, \qquad d_1 = \frac{\lambda_2 - \lambda_1}{6h_1} = -0.5938.$$

Thus, $s_1(x) = 0.520 - 0.02625(x-2.2) - 0.5938(x-2.2)^3$

Spline s_2 for interval $[x_2, x_3]$:

$x2 = 2.4$

$$a_2 = y_2 = 0.510, \qquad b_2 = \frac{y_3 - y_2}{h_2} - \frac{\lambda_3 + 2\lambda_2}{6},\ h_2 = -0.0975,$$

$$c_2 = \frac{\lambda_2}{2} = -0.35625, \quad d_2 = \frac{\lambda_3 - \lambda_2}{6h_2} = 0.5938$$

Thus, $s_2(x) = 0.510 - 0.0975(x-2.4) - 0.35625(x-2.4)^2 + 0.5938(x-2.4)^3$

7.5 Constructing Interpolating Polynomials using Maple/MATLAB

Constructing Interpolating Polynomials using Maple

(1) Given $\{(x_i, y_i)\}_{i=1}^{N}$, the Maple statements for constructing an interpolating polynomial is as follows

```
> with(CurveFitting):
> PolynomialInterpolation(X,Y,x, form=option)
```

where `X=[x0,x1,…,xn]`, `Y=[y0,y1,...,yn]`, and options specify the form of results to be displayed, i.e., Newton, Lagrange or Power.

Example

```
 > with(CurveFitting):
 > p:=PolynomialInterpolation([2, 2.5, 4],
                 [0.5, 0.4, 0.25], x, form=power);
```

yields

$$\boxed{p := 0.05000000000x^2 - 0.4250000000x + 1.150000000}$$

To evaluate $p(3)$ to approximate $f(3)$, enter

```
> sub(x=3.0)
```

which gives 0.325.

(2) Given $\{(x_i, y_i)\}_{i=1}^{N}$, the Maple function "**spline( )**" can be used to construct piecewise splines on [x1,xn] via

```
> spline([x1,x2,…,xn],[y1,y2,...,yn], x, degree);
```

where ***degree*** is an integer or name (cubic) which refers to the degree of spline (default value 3).

Example

```
> s:=spline( [0,1,2,3], [0,1,4,3], x, cubic);
```

yields

$$\boxed{s := \begin{cases} \frac{1}{5}x + \frac{4}{5}x^3 & x < 1 \\ \frac{14}{5} - \frac{41}{5}x + \frac{42}{5}x^2 - 2x^3 & x < 2 \\ -\frac{114}{5} + \frac{151}{5}x - \frac{54}{5}x^2 + \frac{6}{5}x^3 & otherwise \end{cases}}$$

Constructing Interpolating Polynomials using MATLAB

Given $\{(x_i, y_i)\}_{i=1}^{N}$, the MATLAB command

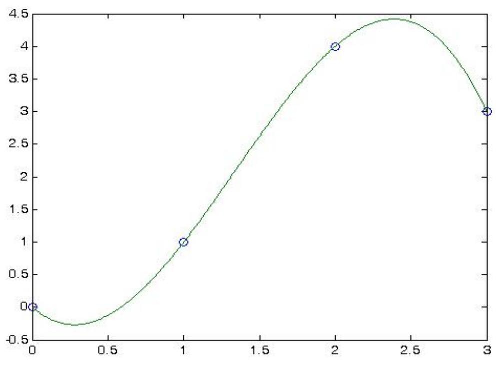
```
>> yi = spline(x,y,xi);
```

can be used to receive a set of data points (*x, y*) and return the values of the cubic spline interpolation function `yi` at the intermediate points `xi`.

Note : `spline(x,y,xi)` is equivalent to `interp1(x,y,xi, 'spline')`.

Example

```
>> x = [0 1 2 3];
>> y = [0 1 4 3];
>> xi = 0:.01:3;
>> yi = spline(x,y,xi);
>> plot(x, y, 'o', xi, yi, '-')
```



EXERCISE 7

Q7.1 Use the following data to construct a third degree Lagrange polynomial approximation to $f(1.09)$. The function being approximated is $f(x) = \log_{10} \tan x$. Use this knowledge to find a bound for the error in the approximation.

$f(1.00) = 0.1924$, $f(1.05) = 0.2414$, $f(1.10) = 0.2933$, $f(1.15) = 0.3492$

(Ans: 0.2826; $7.4*10^{-6}$)

Q7.2 Use the following data to approximate $f(2.15)$ using Newton forward difference formulas with degree 3 and 4 respectively.

x:	2.0	2.1	2.2	2.3	2.4
$f(x)$:	1.414214	1.449138	1.483240	1.516575	1.549193

Then approximate $f(2.38)$ using Newton's backward difference formula of degree 2.
(Ans: 1.466288; 1.466288)

Q7.3 Use the following data to construct a free cubic spline and then find an approximation to $f(5.3)$

x:	5.0	5.2	5.4
$f(x)$:	2.168861	1.797350	1.488591

(Ans: 1.637087)

PROGRAMMING

Q7.4 Write a well-structured Fortran program for constructing a free cubic spline passing through n distinct points (x_0, y_0), … ..., (x_n, y_n). Then, use the program to

(1) solve the problem in Q7.3;

(2) construct a free cubic spline to approximate $f(x) = \cos \pi x$ by using the values given by *f(x)* at $x = 0, 0.25, 0.5, 0.75$ and 1.0.

Algorithm:

Main Program

```
INPUT: $n, x_1, x_2, ..., x_n,\ y_1, y_2, ..., y_n$
Call CTriDS (Input: N, X, Y; Output: aa, dd, cc, bb)   – Construct the tridiagonal system (Coef. & RHS)
Call SolTriDS (Input: n, aa, dd, cc, bb; Output: $\lambda_i$ )   – Solve the tridiagonal system.
For $i = 1$ to $n-1$ Do
    set $a_i = y_i,\quad b_i = \frac{y_{i+1} - y_i}{h_i} - \frac{\lambda_{i+1} + 2\lambda_i}{6} h_i,\quad c_i = \frac{1}{2}\lambda_i,\quad d_i = \frac{\lambda_{i+1} - \lambda_i}{6 h_i}$
Output ( 'On subinterval [** , **]' $S_i = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$ using the result obtained)
End
```

Subroutine CTriDS(N, X, Y, AA, DD, CC, BB)

where	*N*	*= Input: number of data points.*
	X(:),*Y*(:)	*= Input: before entry, must contain the x and y coordinates of the points.*
	AA(:)	*= Output: on exit, contains the sub-diagonal elements of the coefficient matrix.*
	DD(:)	*= main diagonal*
	CC(:)	*= super-diagonal*
	BB(:)	*= Output: on exit, contains the right hand side of the tridiagonal system.*

```
For $i = 1$ to $n - 1$
    set $h_i = x_{i+1} - x_i$
set $dd_1 = 1,\ cc_1 = 0,\ dd_n = 1,\ aa_n = 0,\ bb_1 = 0,\ bb_n = 0$
For $i = 2$ to $n - 1$
    set $aa_i = h_{i-1}$ , $dd_i = 2(h_{i-1} + h_i)$ , $cc_i = h_i$   $bb_i = 6*\left[(y_{i+1} - y_i)/h_i - (y_i - y_{i-1})/h_{i-1}\right]$
```

CHAPTER 8 Numerical Integration and Differentiation

The need to evaluate $\int_a^b f(x)dx$ numerically usually arises in the following cases

- $f(x)$ is known as tabulated values only;
- $f(x)$ is not integrable analytically, for example, when $f(x) = e^{x^2}$ or $\sqrt{1-x^3}$.

The method for approximating $\int_a^b f(x)dx$ is called numerical quadrature that uses a sum of the type $\sum_{i=0}^{n} c_i f(x_i)$ to approximate the integral. In this chapter, we will study various numerical quadrature rules.

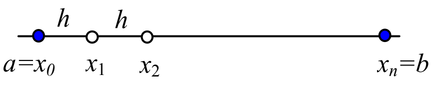
8.1 Newton-Cotes Formulas

The Newton-Cotes formulas are based on Newton's interpolating formulas. To evaluate

$$\int_a^b f(x)dx ,$$

we first select a set of equidistant nodes $\{x_o = a, x_1 ,........., x_n = b\}$ with step size $h = (b-a)/n$ in the interval $[a, b]$, and let

$$x = x_0 + sh .$$



Thus from Newton's forward interpolating formula, we have

$$f(x) = P_n(x_0 + sh) + R[\xi(x)] , \tag{8.1}$$

where $\xi(x) \in [x_0, x_n]$ and

$$P_n(x_0 + sh) = \left[1 + \binom{s}{1}\Delta + \binom{s}{2}\Delta^2 + ... + \binom{s}{n}\Delta^n\right] f(x_0) ,$$

$$R[\xi(x)] = \binom{s}{n+1} h^{n+1} f^{(n+1)}[\xi(x)] .$$

Approximating the integrand $f(x)$ by $P_n(x)$, we obtain Newton-Cotes formula of degree n.

Newton-Cotes formula of degree n : $\int_a^b f(x)\,dx \approx \int_a^b P_n(x_0+sh)\,dx$ (8.2)

with error: $E(\xi)=\int_a^b R(\xi)\,dx = h^{n+1}\int_a^b \binom{s}{n+1} f^{(n+1)}(\xi)\,dx.$ (8.3)

Remarks The form of quadrature formulae depends on the degree of the interpolating polynomial used. The following quadrature formulae arise from the cases where n=1, 2 and 3.

Trapezoidal Rule (n =1)

For n =1, the integration interval is [x_0, x_1] and $f(x)$ is approximated by the interpolating polynomial of degree one, namely

$$f(x)\approx P_1(x)=f_0+s\Delta f_0$$

with interpolating error $\binom{s}{2}h^2 f''(\xi)$ where $\xi\in[x_0,x_1]$. Thus

$$\begin{aligned}\int_{x_0}^{x_1} f(x)dx &\approx \int_{x_0}^{x_1}(f_0+s\Delta f_0)\,dx = h\int_0^1 (f_0+s\Delta f_0)\,ds \\ &= \left[hf_0 s\right]_0^1 + h\Delta f_0\left[\frac{s^2}{2}\right]_0^1 = h\left(f_0+\frac{1}{2}\Delta f_0\right)=\frac{1}{2}h(f_0+f_1),\end{aligned} \tag{8.4}$$

$$E(\xi)=\int_{x_0}^{x_1}\binom{s}{2}h^2 f''[\xi(x)]dx = \frac{h^3}{2}\int_0^1 s(s-1)f''[\xi(x)]ds\,. \tag{8.5}$$

As $g(s)=s(s-1)$ is of one sign on [0, 1] , by using the mean value theorem for integrals (see section 1.1), we have from (8.5) that

$$E[\xi]=\frac{h^3}{2}f''(\xi_1)\int_0^1 s(s-1)ds = -\frac{h^3}{12}f''(\xi_1)\,. \tag{8.6}$$

In summary, for n=1, Newton-Cotes formula reduces to the Trapezoidal rule as below:

Trapezoidal Rule: $\int_{x_0}^{x_1} f(x)dx = \frac{1}{2}h(f_0+f_1)$

with Error: $E(\xi_1)=-\frac{h^3}{12}f''(\xi_1),\ \xi_1\in[x_0,x_1].$

Remark:

Graphically, in Trapezoidal rule, the curve of $f(x)$ is approximated by a straight line of $P_1(x)$.

The exact value of the integral = area between the x-axis and the curve $y = f(x)$ from x_0 to x_1; while the numerical value of the integral = area between the x-axis and the straight line $y = P_1(x)$ from x_0 to x_1.

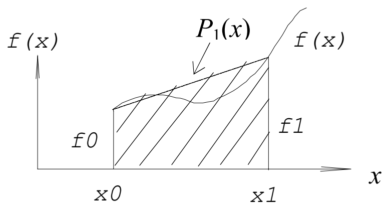


Fig. 8.1 Schematic diagram for Trapezoidal rule

Simpson's 1/3 Rule (n = 2)

For n = 2, the integration interval consists of two equal subintervals with nodes x_0, x_1 and x_2. The integrand $f(x)$ is approximated by the interpolating polynomial of degree two $P_2(x)$. Thus, we have

$$\int_{x_0}^{x_2} f(x)dx \approx \int_{x_0}^{x_2} \left(f_0 + s\Delta f_0 + \frac{s(s-1)}{2}\Delta^2 f_0 \right) dx$$

$$= h\int_0^2 \left(f_0 + s\Delta f_0 + \frac{s(s-1)}{2}\Delta^2 f_0 \right) ds = \frac{1}{3}h(f_0 + 4f_1 + f_2)\,. \tag{8.7}$$

As the integral of the next term

$$\int_{x_0}^{x_2} \frac{s(s-1)(s-2)}{6}\Delta^3 f_0 dx = 0\,,$$

we have the error estimate formula

$$E(\xi) = \int_{x_0}^{x_2} \frac{s(s-1)(s-2)(s-3)}{24} h^4 f^{(4)}(\xi)dx = -\frac{1}{90}h^5 f^{(4)}(\xi_1)\,.$$

Note that, s(s –1)(s –2)(s –3) changes sign on [0, 2].

In summary, for n=2, Newton-Cotes formula reduces to Simpson's 1/3 rule as below:

Simpson's 1/3 Rule: $\displaystyle\int_{x_0}^{x_2} f(x)dx = \frac{1}{3}h(f_0 + 4f_1 + f_2)$

with Error: $\displaystyle E(\xi_1) = -\frac{h^5}{90} f^{(4)}(\xi_1), \quad \xi_1 \in [x_0, x_2].$

Simpson's 3/8 Rule ($n = 3$)

By approximating the integrand $f(x)$ by Newton's forward interpolating polynomial of degree three $P_3(x)$, we obtain

Simpson's 3/8 Rule: $$\int_{x_0}^{x_3} f(x)dx \approx \int_{x_0}^{x_3} P_3(x_0 + sh)dx = \frac{3h}{8}\left(f_0 + 3f_1 + 3f_2 + f_3\right),$$

with Error $$E(\xi_1) = -\frac{3}{80}h^5 f^{(4)}(\xi_1), \quad \xi_1 \in [x_0, x_3]$$

Notes: Both the Newton-Cotes formulas for n=2 and n=3 have error in the order of h^5, i.e $O(h^5)$, and the coefficients in the error expressions indicate that the quadrature for n=2 is unexpectedly accurate. This phenomenon is true for all even-order Newton-Cotes formulas.

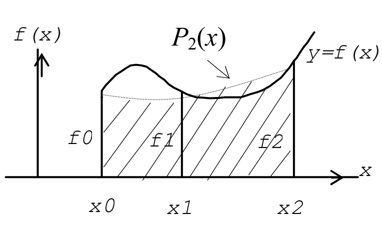


Fig 8.2 Diagram for Simpson's 1/3 rule

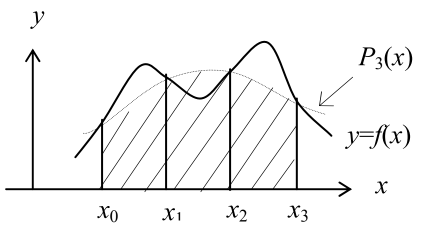


Fig 8.3 Diagram for Simpson's 3/8 rule

8.2 Composite Numerical Integration

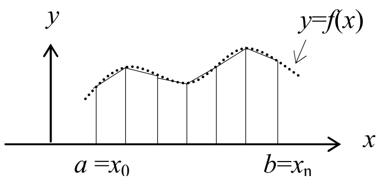
The Newton - Cotes formulas are generally unsuitable for use over large integration intervals, as high-degree polynomials would be required for use over such interval but the coefficients in these polynomials are difficult to obtain. In addition, due to the oscillatory nature of high-degree polynomials, integration using such polynomials may yield inaccurate results. Thus, a piecewise approach to numerical integration that uses the low-order Newton - Cotes formulas is generally applied in practice.

Composite Trapezoidal Rule

Subdividing the interval $[a, b]$ into n subintervals $[x_{i-1}, x_i]$ (i=1,2,…n) and then using the Trapezoidal rule on each subinterval, we have

$$\int_a^b f(x)dx = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x)dx = \sum_{i=1}^{n} \frac{h}{2}(f_{i-1} + f_i)$$

$$= \frac{h}{2}(f_0 + 2f_1 + \ldots + 2f_{n-1} + f_n);$$



$$E=\sum_{i=1}^{n}-\frac{1}{12}h^3 f''(\xi_i)=-\frac{1}{12}h^3\sum_{i=1}^{n}f''(\xi_i)\,.$$

Assume that f'' is continuous, then by the mean value theorem for integrals, there exists $\xi\in[a,b]$ such that

$$f''(\xi)=\frac{1}{n}\sum_{i=1}^{n}f''(\xi_i)\,,$$

and thus,

$$E=-\frac{1}{12}h^2(b-a)f''(\xi)\,.$$

In summary, the composite trapezoidal rule with error is as follows:

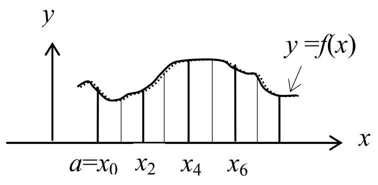
Composite Trapezoidal Rule: $\int_a^b f(x)dx=\frac{h}{2}\left[f(a)+f(b)+2\sum_{i=1}^{n-1}f(x_i)\right]$ where $x_i=a+ih$

with Error: $E(\xi)=-\frac{h^2}{12}(b-a)f''(\xi)$ where $\xi\in[a,b]$.

Notes: Local errors (in one interval) are $O(h^3)$; global error (in n intervals) is $O(h^2)$.

Composite Simpson' 1/3 Rule

We first subdivide the interval $[a, b]$ into n subintervals $[x_{i-1}, x_i]$ (i=1, n) and then use Simpson's 1/3 rule on each consecutive pair of subintervals. As each application of Simpson's 1/3 rule requires two subintervals, n must be an even number, that is $n=2m$, so that the n intervals can be grouped into m pairs $[x_{2(i-1)},x_{2i}]$ (i=1, m) each with 3 equidistant nodes x_{2i-2},x_{2i-1} and x_{2i}.



Application of Simpson's 1/3 rule on each pair yields

Composite Simpson's 1/3 Rule:

$$\int_a^b f(x)dx=\sum_{i=1}^{m}\int_{x_{2(i-1)}}^{x_{2i}}f(x)dx=\frac{h}{3}\left(f_0+4f_1+2f_2+4f_3+\dots..+4f_{n-1}+f_n\right)$$

with Error $E(\xi)=-\frac{b-a}{180}h^4 f^{(4)}(\xi)\,.$

Notes: We have used $\sum_{i=1}^{m}f^{(4)}(\xi_i)=\frac{n}{2}f^{(4)}(\xi)\,.$

Algorithm for Composite Simpson' 1/3 Rule

```
To approximate the integral $I=\int_a^b f(x)dx$

INPUT endpoints a,b; even positive integer n
Set h=(b-a)/n
    XI0= f(a)+ f(b)
    XI1= 0 (for summation of f($x_{2i-1}$))
    XI2= 0 (for summation of f($x_{2i}$) )
For i=1,..., n-1 do
    Set x = a+ih
    If i is even then
       set XI2=XI2+f(x)
    else
       set XI1=XI1+f(x)
Set XI=h*(XI0+2*XI2+4*XI1)/3.0
OUTPUT (XI)
STOP
```

Composite Simpson's 3/8 Rule

We first subdivide the interval $[a, b]$ into n subintervals $[x_{i-1}, x_i]$ (i=1, n) and then use Simpson's 3/8 rule on each three of subintervals. Since each application of Simpson's 3/8 rule requires three subintervals, n must be equal to $3m$ so that the n subintervals can be made into m groups $[x_{3(i-1)}, x_{3i}]$ (i=1, m) each with 4 equidistant nodes $x_{3i-3}, x_{3i-2}, x_{3i-1}$ and x_{3i}. Application of Simpson's 3/8 rule on each 3 consecutive subintervals yields

Composite Simpson's 3/8 Rule

$$\int_a^b f(x)dx \approx \frac{3h}{8}\left(f_0+3f_1+3f_2+2f_3+3f_4+3f_{5}+\ldots+2f_{n-3}+3f_{n-2}+3f_{n-1}+f_n\right)$$

Notes: This rule is useful in the case in which we cannot apply Simpson's 1/3 - rule when there are only an odd number of subintervals.

8.3 Romberg Integration

General Form of Error Term

From the *Euler - Maclaurin Theorem* (see Atkinson, An introduction to Numerical Analysis), we can determine the specific form of the error term for different quadrature rules.

In general, provided that the integrand $f(x)$ has continuous derivatives, the truncation error for the composite Trapezoidal rule (Trap) and the midpoint rule (MPR) is

$$E_T = Ah^2 + Bh^4 + Ch^6 + ... = A\frac{1}{n^2} + B\frac{1}{n^4} + C\frac{1}{n^6} + ... \tag{8.8}$$

while for the composite Simpson's rule,

$$E_s = Ah^4 + Bh^6 + Ch8 + ... = A\frac{1}{n^4} + B\frac{1}{n^6} + C\frac{1}{n^8} + ...$$

Romberg Integration Scheme

Let *In* be the numerical approximation of $I = \int_a^b f(x)dx$ obtained by using the composite Trapezoidal rule with n subintervals, then

$$I = I_n + \frac{A}{n^2} + \frac{B}{n^4} + \frac{C}{n^6} + ... \tag{8.9}$$

$$I = I_{2n} + \frac{A}{2^2 n^2} + \frac{B}{2^4 n^4} + \frac{C}{2^6 n^6} + ... \tag{8.10}$$

4*(8.10)-(8.9) and rearranging yields

$$I = \frac{4I_{2n} - I_n}{3} - \frac{B}{4n^4} - \frac{5C}{16n^6} - ... := U_{2n,2} + \frac{B'}{n^4} + \frac{C'}{n^6} + ... \tag{8.11}$$

Now, by using the above extrapolation in a systematic way, we can gradually eliminate the lower order terms of $(1/n)$ in the error and thus increase the accuracy of the approximation.

From (8.11), by replacing n by $2n$, we have

$$I = U_{4n,2} + \frac{B'}{2^4 n^4} + \frac{C'}{2^6 n^6} + ... \tag{8.12}$$

16*(8.12)-(8.11) and rearranging yields

$$I = \frac{16U_{4n,2} - U_{2n,2}}{16-1} + \frac{C''}{n^6} - ... := U_{4n,3} + \frac{C''}{n^6} + ... \tag{8.13}$$

Thus we can obtain the following table (Romberg integration table) row by row starting from row one,

$$\begin{tikzcd}
I_n \arrow[dr] & & \\
I_{2n} \arrow[r] \arrow[dr] & U_{2n,2} \arrow[dr] & \\
I_{4n} \arrow[r] & U_{4n,2} \arrow[r] & U_{4n,3}
\end{tikzcd}$$

To simplify notation, we denote

$I_{2^{i-1}}(I_1, I_2, I_4, \ldots)$ by $R_{i,1}(R_{1,1}, R_{2,1}, R_{3,1}, \ldots)$ and

$U_{2^{i-1},j}$ $(U_{2,j}, U_{4,j}, U_{8,j} \ldots)$ by $R_{i,j}$ $(R_{2,j}, R_{3,j}, R_{4,j} \ldots)$.

Then we have the following Romberg integration table

$$\begin{tikzcd}
R_{1,1} \arrow[dr, dash] & & & \\
R_{2,1} \arrow[r, dash] \arrow[dr, dash] & R_{2,2} \arrow[dr, dash] & & \\
R_{3,1} \arrow[r, dash] \arrow[d, dotted, dash] & R_{3,2} \arrow[r, dash] & R_{3,3} \arrow[dr, dotted, dash] & \\
R_{n,1} \arrow[r, dash] & R_{n,2} \arrow[rr, dotted, dash] & & R_{n,n}
\end{tikzcd}$$

Remark 1 (Construction Strategy for the Romberg Table)

The truncation error associated with the *j*th column of the table is $O(1/n^{2j})$. Thus, the diagonal elements converge to the solution with the highest rate, and so we should construct the table row by row. For each row (say the *i*th row), we first calculate $R_{i,1}$ by using the composite Trapezoidal rule

$$R_{i,1} = I_{2^{i-1}}(h_i) = \frac{1}{2}\left[R_{i-1,1} + h_{i-1}\sum_{k=1}^{2^{i-2}} f\left(a + (k - 1/2)h_{i-1}\right)\right], \text{ where } h_i = \frac{b-a}{2^{i-1}}, \tag{8.14}$$

then using the calculated value to obtain the successive elements in the row $R_{i,j}$ (j = 2, 3, ..., i) by

$$R_{ij} = \frac{4^{j-1} R_{i,j-1} - R_{i-1,j-1}}{4^{j-1} - 1}. \tag{8.15}$$

Remark 2 (Conditions for using Romberg integration**)**

Romberg integration applied to *f(x)* on [*a, b*] relies on the assumption that the composite trapezoidal rule has an error term that can be expressed in the form of equation (8.8); that is, we must have $f(x) \in C^{2k+2}[a,b]$ for the *k*th row to be generated.

Example 8.1: Calculate $\int_0^{\pi} \sin x dx$ using Romberg integration scheme.

Solution. Firstly, generate column one using the composite trapezoidal rule with 1, 2, 4 and 8 subintervals respectively. Then calculate $R_{i,j}$ using (8.15) for j=2,3,4; i=j to 4. The computed results are as follows.

N	i	*Trap.*	$R_{i,2}$	$R_{i,3}$	$R_{i,4}$
1	1	0			
2	2	1.57079633	2.09439511		
4	3	1.89611890	2.00455976	1.99857073	
8	4	1.97423160	2.00026917	1.99998313	2.00000555

eg. $$R_{2,2}=\frac{4^{2-1}R_{2,1}-R_{1,1}}{4^{2-1}-1}=\frac{4*1.57079633-0}{3}=2.09439511$$

$$R_{3,2}=\frac{4^{2-1}R_{3,1}-R_{2,1}}{4^{2-1}-1}=\frac{4*1.89611890-1.57079633-0}{3}=2.00455976$$

$$\therefore \quad \int_0^{\pi}\sin x dx \approx R_{4,4}=2.00000555\,.$$

8.4 Gaussian Quadrature

Since $t=\dfrac{2x-(a+b)}{b-a}$ maps the interval [*a, b*] for *x* onto [-1, 1] for *t*, we only need to consider $\int_{-1}^{1} w(x)f(x)dx$ where *w(x)* is a positive weight function. Clearly, any integral can be written in this form as:

$$\int g(x)dx=\int w(x)\frac{g(x)}{w(x)}dx=\int w(x)f(x)dx\,.$$

Principle of Gaussian Quadrature

Consider evaluating a definite integral numerically by

$$\int_{-1}^{1} f(x)dx=\sum_{i=1}^{n} c_i f(x_i).$$

All the Newton-Cotes formulas require that x_i are spaced equally. If the function is given explicitly, however, the points for evaluating the function can be chosen in a manner that leads to increased accuracy of approximation.

Gaussian quadrature is concerned with choosing the points x_i and the constants c_i for evaluation in an optimal manner to minimize the error in the approximation.

Methods for Determination of x_i and c_i for second order Gaussian Formula

For the cases where $n = 2$ **and** $w(x) = 1$

$$\int_{-1}^{1} f(x)dx \cong a_1 f(x_1)+a_2 f(x_2)\,. \tag{8.16}$$

The integration points x_i and the constants c_i are such determined to make the above formula exact for $f(x)=1$, x, x^2, x^3, i.e., to make it exact for any cubic polynomials. Substituting $f(x)=1$, x, x^2, x^3 into (8.16) respectively yields

$$2 = a_1 + a_2,$$

$$0 = a_1 x_1 + a_2 x_2,$$

$$\frac{2}{3} = a_1 x_1^2 + a_2 x_2^2,$$

$$0 = a_1 x_1^3 + a_2 x_2^3.$$

Solving the above system of nonlinear equations yields

$$x_1 = \frac{1}{\sqrt{3}}, \qquad x_2 = -\frac{1}{\sqrt{3}}, \qquad a_1 = a_2 = 1.$$

$$\therefore \quad \int_{-1}^{1} f(x)dx = f(-\frac{1}{\sqrt{3}}) + f(\frac{1}{\sqrt{3}}).$$

This is a 2-point Gaussian quadrature formula. It is exact for any third (2*2-1) degree polynomials. In general, an *n*th order Gaussian formula is exact for any 2*n*-1degree polynomials.

Determination of x_i and c_i for Higher - Order Gaussian Formulas.

For higher-order Gaussian formulas, the method presented above requires solving a set of nonlinear equations of higher order that is very difficult. Thus, an alternative procedure, as described by the theorem below, has been derived for the determination of x_i and c_i, based on the properties of orthogonal polynomials.

Theorem: If *P* is any polynomial of degree less than or equal to 2*n* –1, then

$$\int_{-1}^{1} P(x)dx = \sum_{i=1}^{n} c_i P(x_i), \text{ where } c_i = \int_{-1}^{1} \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)} dx$$

and $x_1, x_2, \ldots \ldots, x_n$ are the zeros of the *n*th Legendre polynomial.

Proof: (see Burden & Faires, Numerical Analysis).

So, for a general function *f*(*x*), we use the following Gaussian quadrature formulas:

$$\int_{-1}^{1} f(x)dx = \sum_{i=1}^{n} c_i f(x_i).$$

where $\{x_i\}$ and $\{c_i\}$ are defined in the above theorem but in practice they are tabulated and are available in mathematics handbooks.

Types of Gaussian Quadrature Formulae

Depending on the integrand and the integration limits, there are different types of Gaussian quadrature formulae as listed below

Gauss-Legendre: $\int_{-1}^{1} f(x)dx = \sum_{i=1}^{n} w_i f(x_i)$

Gauss-Laguerre: $\int_{0}^{\infty} \mathrm{e}^{-x} f(x)dx = \sum_{i=1}^{n} w_i f(x_i)$

Gauss-Hermite: $\int_{-\infty}^{\infty} \mathrm{e}^{-x^2} f(x)dx = \sum_{i=1}^{n} w_i f(x_i)$

Gauss-Chebyshev: $\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x)dx = \sum_{i=1}^{n} w_i f(x_i)$.

Tables for the values of w_i and x_i for the various types of Gaussian formulae can be found from "Handbook of Mathematical Functions" by Abramowitz and Stegun. The following table is an example.

Parameters for Gaussian Quadrature $\int_{-1}^{1} f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i)$

n	x_i	w_i
2	±0.57735027	1
3	0 ±0.77459667	0.88888889 0.55555556
4	±0.33998104 ±0.86113631	0.65214515 0.34785485

Example 8.2: Evaluate $I = \int_0^{\pi} e^x \cos x\, dx$ using the 3-point Gaussian quadrature formula.

Solution: Let $t = \frac{2x-(a+b)}{b-a} = \frac{2x-\pi}{\pi}$.

Then $x = \frac{\pi}{2}(1+t)$, $dx = \frac{\pi}{2}dt$ and

$$I = \frac{\pi}{2}\int_{-1}^{1} e^{\frac{\pi}{2}(t+1)} \cos\left[\frac{\pi}{2}(t+1)\right]dt := \frac{\pi}{2}\int_{-1}^{1} f(t)dt$$

$$= \frac{\pi}{2}\left[0.556 f(-0.775) + 0.889 f(0) + 0.556 f(0.775)\right] = -12\,.$$

8.5 Multiple Integrals

The techniques for evaluating single integrals can be adapted for use in the approximation of multiple integrals. To evaluate the double integral

$$\int_a^b \int_c^d f(x,y)\,dydx, \tag{8.17}$$

we first write the integral as an iterated integral

$$\int_a^b \left(\int_c^d f(x,y)\,dy \right) dx. \tag{8.18}$$

Then, treating x as a constant, we can use the techniques for single integrals to evaluate the inner integral

$$\int_c^d f(x,y)\,dy$$

to yield a function $F(x)$, and thus the double integral becomes

$$\int_a^b F(x)\,dx, \tag{8.19}$$

which again can be evaluated using the techniques for single integrals. In the following, we present two specific forms of quadrature for double integrals.

Composite Trapezoidal Rule for Double Integrals

Subdivide the integral $[a, b]$ into n subintervals with step size $h = \dfrac{b-a}{n}$ and nodes $x_0, x_1, ..., x_n$; and subdivide the integral $[c, d]$ into m subintervals with step size $k = \dfrac{d-c}{m}$ and nodes $y_0, y_1, ..., y_m$. Then by using the composite Trapezoidal rule, we have

$$\begin{aligned}
\int_a^b \int_c^d f(x,y)\,dydx &= \int_a^b \left[\tfrac{k}{2}(f(x,y_0) + f(x,y_m) + 2\sum_{j=1}^{m-1} f(x,y_j)) \right] dx \\
&= \int_a^b F(x)\,dx \\
&= \frac{h}{2}\left[F(x_0) + F(x_n) + 2\sum_{i=1}^{n-1} F(x_i) \right] \\
&= \frac{h}{2}\Big\{ \frac{k}{2}(f(x_0,y_0) + f(x_0,y_m) + 2\sum_{j=1}^{m-1} f(x_0,y_j)) \\
&\quad + \frac{k}{2}(f(x_n,y_0) + f(x_n,y_m) + 2\sum_{j=1}^{m-1} f(x_n,y_j)) \\
&\quad + \frac{k}{2} 2\sum_{i=1}^{n-1} [\ f(x_i,y_0) + f(x_i,y_m) + 2\sum_{j=1}^{m-1} f(x_i,y_j)\ ] \Big\}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{kh}{4}\Big\{f(x_0,y_0)+f(x_0,y_m)+f(x_n,y_0)+f(x_n,y_m) \\
&\quad +2\sum_{j=1}^{m-1}\left[f(x_0,y_j)+f(x_n,y_j)\right]+2\sum_{i=1}^{n-1}\left[f(x_i,y_0)+f(x_i,y_m)\right] \\
&\quad +4\sum_{i=1}^{n-1}\sum_{j=1}^{m-1} f(x_i,y_j)\Big\}. \qquad (8.20)
\end{aligned}
$$

Gaussian Quadrature for Double Integrals

We first must transform the region of integration

$$\mathrm{R}=\{(x,y)\,|\,a\le x\le b,\quad c\le y\le d\}$$

into

$$\bar{\mathrm{R}}=\{(u,v)\,|\,-1\le u\le 1,\quad -1\le v\le 1\}$$

by using the linear transformations

$$u=\frac{2x-(a+b)}{b-a},\quad v=\frac{2y-(c+d)}{d-c}. \qquad (8.21)$$

Hence, we have

$$
\begin{aligned}
&x=\frac{1}{2}[(a+b)+(b-a)u],\quad y=\frac{1}{2}[(c+d)+(d-c)v], \\
&dxdy=\frac{1}{4}(b-a)(d-c)dudv := |J|\,dudv, \\
&I=\int_a^b\int_c^d f(x,y)dxdy=\int_{-1}^{1}\int_{-1}^{1}\bar{f}(u,v)\,|J|\,dudv,
\end{aligned}
$$

where $\bar{f}(u,v)=f(x(u),y(v))$, and $|J|=(b-a)(d-c)/4$.

By applying an n-point Gaussian quadrature with respect to v and u respectively, we have

$$
\begin{aligned}
I&=\int_{-1}^{1}\int_{-1}^{1}\bar{f}(u,v)\,|J|\,dudv=\int_{-1}^{1}\left(\int_{-1}^{1}\bar{f}(u,v)\,|J|\,dv\right)du=\int_{-1}^{1}\Big(\sum_{j=1}^{n}\bar{f}(u,v_j)\,|J|\,w_j\Big)du \\
&=\sum_{i=1}^{n}\sum_{j=1}^{n}\bar{f}(u_i,v_j)\,|J|\,w_iw_j\,, \qquad (8.22)
\end{aligned}
$$

where $u_i\,(v_j)$ are integration points and $w_i\,(w_j)$ are their associated weights. For n=3, the $u_i\,(v_j)$ and $w_i\,(w_j)$ are as follows:

n	$u_i\,(v_j)$	$w_i\,(w_j)$
3	-0.774596 0 +0.774596	0.555556 0.888889 0.555556

Remarks

(1) The techniques in the above can be modified to approximate double integrals with variabe inner limits, i.e., over the region $\mathrm{R}=\{(x,y)\,|\,a\le x\le b,\quad c(x)\le y\le d(x)\}$.

(2) The technique in the above can be extended for triple integrals.

8.6 Numerical Differentiation

Let $x_0, x_1, \dots, x_n$ are $(n+1)$ distinct numbers in some interval $\mathbf{I}$ and that $f\in C^{n+1}(\mathbf{I})$. Then

$$f(x)=\sum_{i=0}^{n} f(x_i)l_i(x)+\frac{f^{(n+1)}[\xi(x)]}{(n+1)!}\prod_{i=0}^{n}(x-x_i) \tag{8.23}$$

for some $\xi(x)$ in **I**, where $l_i(x)$ is the *i*th Lagrange coefficient polynomial for *f*, i.e.

$$l_i(x)=\prod_{\substack{j=0\\ j\ne i}}^{n}\frac{(x-x_j)}{(x_i-x_j)}.$$

Differentiation of (8.23) with respect to *x* yields

$$\begin{aligned}f'(x)=\sum_{i=0}^{n} f(x_i)l_i'(x)&+\frac{1}{(n+1)!}f^{(n+1)}[\xi(x)]\frac{\partial}{\partial x}\prod_{i=0}^{n}(x-x_i)\\ &+\frac{1}{(n+1)!}\prod_{i=0}^{n}(x-x_i)\frac{\partial}{\partial x}\left(f^{(n+1)}[\xi(x)]\right).\end{aligned} \tag{8.24}$$

At $x=x_k$,

$$f'(x_k)=\sum_{i=0}^{n} f(x_i)l_i'(x_k)+\frac{1}{(n+1)!}f^{(n+1)}[\xi(x_k)]\prod_{\substack{i=0\\ i\ne x_k}}^{n}(x_k-x_i) \tag{8.25}$$

Applying this technique using the first order Lagrange polynomial at x_0 and $x_0+\lambda$ for some value λ produces the following

2-point formulae: $f'(x_0)=\dfrac{f(x_0+\lambda)-f(x_0)}{\lambda}-\dfrac{\lambda}{2}f''(\xi),\quad$ where $\xi\in[x_0,x_0+h]$.

Remarks

(i) If $\lambda=h>0$ then $f'(x_0)=\dfrac{f(x_0+h)-f(x_0)}{h}$ is called the ***forward difference formula*** and the error is $E=-\frac{h}{2}f''(\xi)$ where $\xi\in[x_0,x_0+h]$;

If $\lambda=-h<0$ then $f'(x_0)=\dfrac{f(x_0)-f(x_0-h)}{h}$ is called the ***backward difference formula*** and the error is $E=\frac{h}{2}f''(\xi)$ where $\xi\in[x_0-h,x_0]$.

(ii) The above formulae can also be derived from Taylor's theorem.

Using (8.25) at $x_0, x_1 = x_0 + h$ and $x_2 = x_0 + 2h$ produces the following

3-point forward difference formulae:

$$f'(x_0) = \frac{1}{2h}\left(-3f(x_0) + 4f(x_0+h) - f(x_0+2h)\right) + \frac{h^2}{3}f^{(3)}(\xi).$$

where $\xi \in [x_0, x_0+2h]$

Using (8.25) at $x_0 - h, x_0$ and $x_0 + h$ yields the following

3-point central difference formulae:

$$f'(x_0) = \frac{1}{2h}(f(x_0+h) - f(x_0-h)) - \frac{h^2}{6}f^{(3)}(\xi), \quad \text{where} \quad \xi \in [x_0-h, x_0+h].$$

Remarks: The above formulae can also be derived from Taylor's theorem.

Using (8.25) at $x_0 - 2h, x_0 - h, x_0, x_0 + h$ and $x_0 + 2h$ yields the following

5-point central difference formulae:

$$f'(x_0) = \frac{1}{12h}(f(x_0-2h) - 8f(x_0-h) + 8f(x_0+h) - f(x_0+2h)) + \frac{h^4}{30}f^{(5)}(\xi),$$

where $\xi \in [x_0 - 2h, x_0 + 2h]$.

8.7 Numerical Integration and Differentiation Using Maple/MATLAB

Numerical Integration and Differentiation Using Maple

(a) The following Maple commands can be used to evaluate $\int_a^b f(x)\,dx$ numerically

```
with(student[Calculus1]):
ApproximateInt(f(x), x = a..b, method = simpson, partition=n)
```

Notes: By default, the interval (a,b) is divided into 10 subintervals and Simpson's 1/3 rule is applied to each subinterval. To change the number of subintervals from 10 to a different number *n*, simple add in the optional argument "`partition=n`" where ***n*** is an integer number.

Example 1 Evaluate $\displaystyle\int_0^1 \frac{e^{-x^3}}{x^2+1}\,dx$.

```
>with(Student[Calculus1]):
>Intval := ApproximateInt(exp(-x^3)/(x^2+1),x=0..1,
                             method=simpson,partition=12);
>evalf(Intval);
```

yields

$$\boxed{0.6649370529}$$

Given $\int_c^d \int_a^b f(x,y)\,dxdy$, by writing the integral as an iterative integral

$$\int_c^d \left(\int_a^b f(x,y)\,dx\right)dy,$$

we can use the Maple function "`ApproximateInt( )`" to evaluate the integral.

Example 2 Evaluate $\displaystyle\int_0^2 \int_0^1 x^2y^2\,dx\,dy$.

```
>with(Student[Calculus1]):
>Intval := ApproximateInt(ApproximateInt(x^2*y^2,x=0..1),y=0..2);
>evalf(Intval);
```

yields

$$\boxed{0.8888888889}$$

(b) The following Maple commands can be used to compute a numerical approximation of the ordinary derivative and partial derivative of an expression evaluated at a point.

```
fdiff(f(x), x, x=a, proc)
fdiff(f(x,y,…),[x,y,…],[x=a,y=b,…], proc)
```

Example Evaluate $\left.\dfrac{d}{dx}\cos(x)\right|_{x=1}$.

```
> fdiff(cos(x),x=1);
```

yields

$$\boxed{-.8414709848}$$

Numerical Integration Using MATLAB

The MATLAB function "`trapz()`" evaluates $\int_a^b f(x)\,dx$ via the trapezoidal rule. The syntax is

```
>> trapz(x,f);
```

where `f` defines the integrand and `x` gives the integration points. The MATLAB function "`dblquad()`" can be used to evaluate $\int_c^d \int_a^b f(x,y)\,dxdy$. The syntax is

```
dblquad(@f(x,y), a, b, c, d)
```

Example 1 Evaluate $\int_0^{\pi} \sin x \, dx$

```
>> x = 0:pi/100:pi;
>> f = sin(x);
>> trapz(x,f)
```

produces

```
ans =
     1.9998
```

Example 2 Evaluate $\int_0^2 \int_0^1 x^2 y^2 dx\, dy$ **.**

To evaluate this integral, first create a function “`integrnd()`” containing the integrand.

```
Function out = integrnd(x,y)
out =x.^2+y.^2;
```

and enter the MATLAB statement

```
>>result=dblquad(@integrnd, 0, 1, 0, 2)
```

which produces

```
result =
          0.8889
```

EXERCISE 8

Q8.1 Use the composite Trapezoidal rule with the indicated number of intervals n to approximate the following definite integrals.

$(a)\ \int_1^3 \frac{dx}{x} \quad n = 4; \qquad (b)\ \int_0^3 x\sqrt{1+x^2}\,dx \quad n = 6$

(Ans: 1.1167; 10.3122)

Q8.2 Repeat Q8.1 using the composite Simpson 1/3 rule. (Ans: 1.100000; 10.20635).

Q8.3 Use Romberg integration to calculate $R_{3,3}$ and $R_{4,4}$ for the integrals in Q8.1.

Q8.4 Use Gaussian quadrature with $n = 2$ and 3 to find approximations to $\int_1^3 e^x \sin x dx$

(Ans: 11.141495; 10.948403

Programming

Q8.5 Write a well-structured Fortran/C++/Maple/Matlab function subprogram SIMPS which evaluates a definite integral using the Composite Simpson' 1/3 Rule. Then, write a main program to call the function to evaluate $\int_0^1 f(x)dx$ using $h = 0.05$, where $f(x)$ is defined by

$$f(x) = \begin{cases} x^2 - 1 & 0 \le x \le 0.5 \\ e^x \sin x & 0.5 < \text{x} \le 1 \end{cases}$$

(Hint. see algorithm in Sec. 8. 2)

Q8.6 Design an algorithm and then write a F95/C++/Maple/Matlab function subprogram TRAP to calculate a definite integral using the composite trapezoidal rule. Then write a main program to call the function to evaluate the definite integral given in question Q8.5

Q8.7 Write a well-structured Fortran program ROMB which evaluates a definite integral using the Romberg integration scheme and print the Romberg integration table (n rows). Then, use the program to approximate $\int_1^3 \frac{1}{x}dx$ using Romberg integration with $n = 6$.

Algorithm:

```
INPUT    end points $a, b$, integer $n$.
OUTPUT  $R_{i,j}$ ($j$ =1 to $i$,  $i$ =1 to $n$)
Set  $h = b - a,$
$R_{1,1} = h[f(a) + f(b)]/2$
For $i$ = 2 to $n$ do
    $R_{2,1} = \frac{1}{2}\left\{R_{1,1} + h\sum_{k=1}^{2^{i-2}} f[a + (k - 0.5) * h]\right\}$
    For $j$ = 2 to i do
        $R_{2,j} = \frac{4^{j-1}R_{2,j-1} - R_{1,j-1}}{4^{j-1} - 1}$
    Output ( $R_{2,j}$   $j = 1$ to $i$ )
    Set $h = h/2$
    For $j$ = 1 to $i$
        Set $R_{1,j} = R_{2,j}$
End
```

(*Note: The construction of a new line of the Romberg table only requires data in the immediate previous line. Thus, in the program, we use two one-dimensional arrays, one for storing the value of the current line $R_{2,j}$ and the other for the immediate previous line $R_{1,j}$.*)

Q8.8. Solve Q8.1 and Q8.5 using Maple/Matlab built-in functions.

CHAPTER 9

Solution of Initial Value Problems for Ordinary Differential Equations

In modeling many real world problems, one needs to solve a differential equation or a set of differential equations subject to certain initial and boundary conditions. However, there are relatively few cases for which an analytical solution can be found. Numerical methods for the solution of differential equations are therefore extremely important.

In this chapter, we will first describe several methods for generating numerical solutions of the first order initial value problem:

$$\begin{cases} \dfrac{dy}{dx} = f(x, y) \\ y(a) = y_0. \end{cases} \tag{9.1}$$

In other words, we derive several methods (Taylor series method, Runge-Kutta methods and predictor-corrector methods in sections 9.1-9.4) for generating numerical solutions $\left(x_i, y_i\right)\Big|_{i=1}^{N}$ starting from the initial point $\left(x_0, y_0\right)$, i.e.,

$$\begin{array}{|c|ccccccccc} x_0 & & x_1 & & x_2 & & \dots & & x_n \\ \hline y_0 & \to & y_1 & \to & y_2 & \to & \dots & \to & y_n \\ \hline \end{array}$$

Then we analyze the stability of the numerical methods in section 9.5, and then we show how the methods for first order initial value problems can be adapted to solve systems of first order equations and higher order initial value problems in sections 9.6 and 9.7. Finally in section 9.8, we show how to solve initial value problems for ODEs using Maple and Matlab built-in functions.

For convenience, various concepts/definitions, to be used throughout this chapter, are summarized below.

Ordinary Differential Equation (ODE)

ODE is an equation which relates one and only one independent variable with an unknown function and its derivatives. The order of an ordinary differential equation is the order of the highest derivative in the equation.

Initial Conditions (ICs)

ICs are the conditions that describe the unknown function or/and its derivatives at one and only one point.

Boundary Conditions (BCs)

BCs are the conditions that describe the unknown function or/and its derivatives at more than one points.

Initial Value Problem (IVP)

An IVP consists of a differential equation subject to certain imposed initial conditions.

Boundary Value Problem (BVP)

An BVP consists of a differential equation subject to certain imposed boundary conditions.

Analytical Solution

An analytical solution is a function y(x) satisfying the differential equation and the imposed initial conditions or boundary conditions.

Numerical Solution

If we wish to know the relation of y and x for $x \in [a, b]$, we divide the interval $[a, b]$ into N subintervals with nodes $a = x_0,\ x_1, \dots \dots,\ x_n = b$. Then for each x_i, we compute a corresponding y_i approximating the exact solution $y(x_i)$. The sequence $\left(x_i, y_i\right)\big|_{i=0}^{N}$ is called the numerical solution to the differential equation.

Numerical Method

Numerical method refers to the method for generating $\left(x_i, y_i\right)\big|_{i=0}^{N}$ from the differential equation and the initial condition or the boundary condition given.

Existence

If $f(x, y)$ is defined and continuous on some bounded domain D of the x-y plane, then for every $(x_0, y_0) \in D$ there exists a solution $y(x)$ for the initial value problem

$$y' = f(x), \quad y(x_0) = y_0$$

on some interval containing x_0 and the solution can be extended until $(x, y(x))$ approaches the boundary of D.

Uniqueness

If $f(x, y)$ is defined and continuous on D, and satisfies the Lipschitz condition with respect to y in D

$$\left|f(x, y) - f(x, y^*)\right| < L\left|y - y^*\right|,$$

where L is a constant, and if $(x_0, y_0) \in D$, then there exists a unique solution to the initial value problem on some interval containing x_0 and this solution can be extended uniquely until it approaches the boundary of D.

Note: The Lipschitz condition can be replaced by the weaker condition: $\frac{\partial f}{\partial x}$ continuous and bounded.

9.1 Taylor Series Methods

The Taylor series methods are based on Taylor's series expansion theorem. Suppose the solution $y(x)$ to the IVP has $(p+1)$ continuous derivatives. If we let $x_n = x_0 + nh$, then

$$y(x_{n+1}) = y(x_n) + hy'(x_n) + \frac{h^2}{2!}y''(x_n) + \ ... \ + \frac{h^p}{p!}y^{(p)}(x_n) + \frac{h^{p+1}}{(p+1)!}y^{(p+1)}(\xi), \qquad (9.2)$$

where $\xi \in [x_n, x_{n+1}]$. By deleting the remainder term involving ξ, we obtain the Taylor series formula with the error equal to the term neglected, as below

Talor Series Methods of order p: $y_{n+1} = y_n + hy_n' + \frac{h^2}{2!}y_n'' + \ ... \ + \frac{h^p}{p!}y_n^{(p)}$ (9.3)

with truncation error in each step $E = \frac{h^{p+1}}{(p+1)!}y^{(p+1)}(\xi)$ (9.4)

Remark. With (9.3), one can generate numerical solutions $(x_i, y_i)\big|_{i=0}^{N}$ *starting from the known initial value* (x_0, y_0), and the error in each step of updating is as given in (9.4)

Taylor Series Method of Order One (Euler's Method)

For $p=1$, from (9.3) and $y' = f$, we obtain Euler's method with error as follows.

Euler's Method: $y_{n+1} = y_n + hf(x_n, y_n)$ (9.5)

with error $E = \frac{h^2}{2}y''(\xi)$

Remark. In Euler's method, δy is approximated by Δy as shown in Figure 9.1

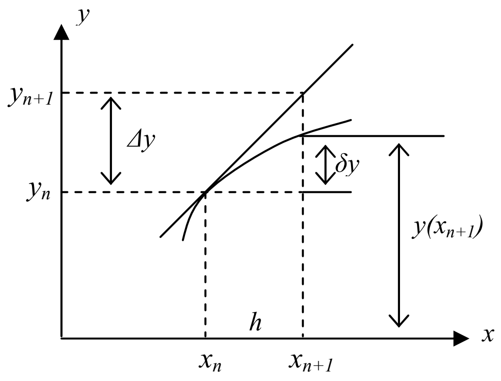


Figure 9.1 Diagram showing Euler's method

Taylor Series Method of Order Two (p >2)

For $p = 2$, from (9.3) and

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

we obtain

$$y_{n+1} = y_n + h\left(f(x_n, y_n) + \frac{h}{2!} f'(x_n, y_n)\right) = y_n + hT(x_n, y_n) \tag{9.6}$$

with error $E = \frac{h^3}{6} y^{(3)}(\xi)$.

Exercise. Solve $\begin{cases} y' = -y + x + 1 & x \in [0,\ 1] \\ y(0) = 1 \end{cases}$

Truncation Error in Taylor Series Methods

The difference between $y(x_{n+1})$ and y_{n+1} results from truncating the Taylor series to a finite number of terms. To calculate y_{n+1} from y_0, we need (n+1) steps. We define

Local Truncation Error: Error in each step.

Global Truncation Error: Total error accumulated from step 1 to the current (n+1)th step.

For Taylor series method of order p, the local truncation error in calculating $y(x+ h)$ from $y(x)$ is

$$\frac{h^{p+1}}{(p+1)!} y^{(p+1)}(\xi), \quad \xi \in [x, x+h]. \tag{9.7}$$

Thus, Local truncation error is of order h^{p+1}, denoted by $O(h^{p+1})$. That is, the local truncation error $\rightarrow Mh^{P+1}$ as $h \rightarrow 0$.

Global truncation error (accumulated from steps 1 to n) is approximately

$$n\frac{1}{(p+1)!} h^{p+1} y^{(p+1)}(\xi) = \frac{b-a}{(p+1)!} h^p y^{(p+1)}(\xi) \quad \text{for } \xi \in [a,b],$$

that is of order h^p, $O(h^p)$. For Euler's method, p =1 and thus the global truncation error is of order h.

Problems with Taylor Series Methods

Taylor series methods involve calculation of derivatives of the function. Thus, users must do some preliminary analytical work before writing a computer program specifically for the task. Ideally, we require a method solely based on function evaluations. The Runge-Kutta method to be presented in the next section is one of this type of methods.

9.2 Runge-Kutta Methods

The Runge-Kutta method is also based on Taylor's series expansion theorem. Given an initial value problem

$$\begin{cases} \dfrac{dy}{dx} = f(x, y) & x \in [a, b] \\ y(a) = y_0 \end{cases}$$

Suppose that the solution $y(x)$ has $(p+1)$ continuous derivatives, then we have the Taylor Series Expansion

$$y(x+h) = y(x) + hy'(x) + \frac{h^2}{2!}y''(x) + \dots.. = y(x) + h\left[f(x, y) + \frac{h}{2}\frac{d}{dx}f(x, y) + \dots..\right] \tag{9.8}$$
$$= y(x) + h\Delta(x, y, h),$$

where $\Delta(x, y, h)$ contains derivatives of $f(x, y)$, and thus is inconvenient to use the above formulae.

The Idea Behind the Runge-Kutta Method

To avoid derivative calculation, the Runge-Kutta method constructs a $\phi(x, y, h)$ which agrees with $\Delta(x, y, h)$ as far as possible (agrees up to the h^{p-1} term) but does not contain derivatives, and replace $\Delta(x, y, h)$ by $\phi(x, y, h)$ to form an one step formula (as below) for generating numerical solutions step by step.

$$y_{n+1} = y_n + h\phi(x_n, y_n, h)\,. \tag{9.9}$$

Definition (order p one-step methods): An one step method is said to be of **order p** if $\phi(x, y, h)$ agree with $\Delta(x, y, h)$ up to the h^{p-1} term (that is y_{n+1} agrees with the Taylor series up to the h^p term)

Method for Deriving $\phi(x,y,h)$ for Runge-Kutta Method of Order p

(1) Let $\phi(x, y, h)$ be the weighted average of the slope of the curve $y = y(x)$ in the interval $[x, x+h]$. For example, for $p = 2$, choose $\phi(x, y, h)$ as in (9.10) with several constants to be determined.

(2) Express both $\phi(x, y, h)$ and $\Delta(x, y, h)$ as polynomials of h.

(3) Determine the constants in $\phi(x, y, h)$ such that ϕ and Δ agree up to and including the term in h^{p-1}.

Derivation of Runge-Kutta Order Two Methods

Let ϕ be the weighted average of the slope of the curve $y = y(x)$ at two points x and $x + p_1 h$ $(0 < p_1 \le 1)$ in the interval $[x, x+h]$, i.e.

$$\begin{aligned}\phi(x,\ y,\ h) &= a_1 y'(x) + a_2 y'(x + p_1 h)\\ &= a_1 f(x,\ y) + a_2 f\left[x + p_1 h,\ y + p_2 h f(x,\ y)\right]\end{aligned} \tag{9.10}$$

where in the above, we approximate the y value at $x + p_1 h$ by

$$y + p_1 h(\beta y'(x)) = y + p_2 h f(x, y) \quad (\because p_2 = \beta p_1).$$

Denoting $\dfrac{\partial f}{\partial x}$ as f_x and $\dfrac{\partial f}{\partial y}$ as f_y, and using Taylor series in two variables, we have

$$\begin{aligned}\phi(x,\ y,\ h) &= a_1 f(x,\ y) + a_2\left[f(x,\ y) + p_1 h f_x + p_2 h f f_y\right] + O(h^2)\\ &= (a_1 + a_2) f + h a_2 (p_1 f_x + p_2 f f_y) + O(h^2).\end{aligned} \tag{9.11}$$

On the other hand

$$\begin{aligned}\Delta(x,\ y,\ h) = f(x,\ y) + \frac{h}{2}\frac{d}{dx} f(x,\ y) + \ \ldots\ &= f + \frac{h}{2}[f_x + f_y y'] + \ldots\\ &= f + \frac{h}{2}\left[f_x + f f_y\right] + O(h^2)\end{aligned} \tag{9.12}$$

Matching ϕ and Δ for the terms with h^0 and h^1 yields

$$\begin{cases} a_1 + a_2 = 1 \\ a_2 p_1 = 1/2 \\ a_2 p_2 = 1/2 \end{cases} \quad \Rightarrow \quad \begin{cases} a_1 = 1 - \alpha \\ a_2 = \alpha \\ p_1 = p_2 = \dfrac{1}{2\alpha} \end{cases} \qquad \alpha \neq 0.$$

Hence the Runge-Kutta order two methods have the form

$$y(x+h) = y(x) + h\left((1-\alpha) f(x, y) + \alpha f\left(x + \frac{1}{2\alpha} h, y + \frac{1}{2\alpha} h f(x, y)\right)\right). \tag{9.13}$$

As α is arbitrary there is an infinite family of R-K schemes each of which is $O(h^2)$. The following are two of the order two schemes, namely improved Euler's method and modified Euler method.

Improved Euler's Method (a =1/2)

$$y_{n+1} = y_n + \frac{1}{2} h\left[f(x_n,\ y_n) + f\left(x_n + h,\ y_n + h f(x_n,\ y_n)\right)\right] \tag{9.14}$$

Modified Euler's Method (α =1)

$$y_{n+1} = y_n + h f\left[x_n + \frac{h}{2},\ y_n + \frac{h}{2} f(x_n,\ y_n)\right] \tag{9.15}$$

Remark (graphic interpretation of the improved Euler method)

As shown in figure 9.2, in the improved Euler method, the average slope of the curve *y*=*y*(*x*) is approximated by the average of the slopes at the left end and the right end of the interval (x_n, x_{n+1})

$$\bar{y}' = \frac{1}{2}(y'_A + y'_B) = \frac{1}{2}[f(x_n, y_n) + f(x_n + h,\, y_n + hf(x_n, y_n)),$$

and the increment of function δy is approximated by $\Delta y = h\,\bar{y}'$.

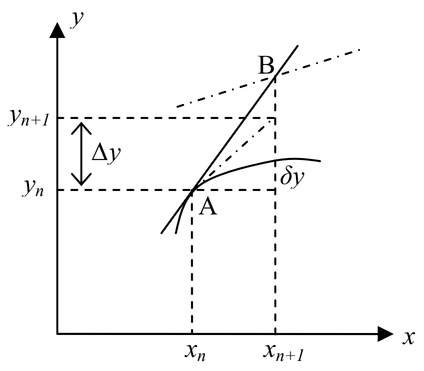


Figure 9.2 graphic interpretation of the improved Euler method

Fourth-Order Runge -Kutta Method

None of the second order Runge-Kutta methods are widely used since the error is only $O(h^2)$ globally. One algorithm in common use for the initial value problems is the R-K method of order four, which is as follows.

Fourth order Runge-Kutta Method

$$\begin{aligned} k_1 &= hf(x_n, y_n), \\ k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1), \\ k_3 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2), \\ k_4 &= hf(x_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{9.16}$$

Remark: 1) From Taylor series, $y(x_n + h) = y(x_n) + h\left[f_n + \frac{h}{2!}f_n' + \frac{h^2}{3!}f_n'' + \frac{h^3}{4!}f_n''' + O(h^4)\right]$.

The Runge-Kutta method (4th order) agrees with the above series to the third order derivative term and thus

local truncation error is in the order of h^5, $O(h^5)$;

global truncation error is in the order of h^4, $O(h^4)$.

2) Problems arise when the relevant Taylor series is very slowly convergent.

Error Estimation

The truncation errors given for the above methods only give us the order of magnitude of the errors since they are given in terms of a derivative at an unknown point ξ.

For 4th order R-K method, local truncation error = $\frac{1}{5!}y^{(5)}(\xi)h^5$;

global truncation error = Kh^4.

If we assume that the 5th order derivative is fairly constant, we can estimate this error as detailed below.

Let $y(x)$ be exact value, $y_h(x)$ and $y_{2h}(x)$ be approximations of $y(x)$ obtained by the Runge-Kutta order four method with step size h and $2h$ respectively.

Then
$$y_h(x) = y(x) + Kh^4, \tag{9.17}$$
$$y_{2h}(x) = y(x) + K(2h)^4.$$

From the above, we have
$$Kh^4 = \frac{y_{2h} - y_h}{15}. \tag{9.18}$$

Therefore, the approximate error for the 4th order Runge-Kutta method using size h is

$$E(x) = y(x) - y_h(x) = \frac{y_{2h}(x) - y_h(x)}{15}. \tag{9.19}$$

This method of error estimation is called Richardson extrapolation.

Example 9.1 Given an I. V. P. $y' = x + y, \quad y(0) = 1$

1) Estimate $y(0.2)$ using the Runge-Kutta (order 4) method with h = 0.1 and h = 0.2
2) Estimate the error in the approximation to $y(0.2)$ with h = 0.1

Solution 1) Using h = 0.1

$n = 0, \quad x_0 = 0, \quad y_0 = 1$

$k_1 = hf(x_0, y_0) = 0.1(0+1) = 0.1000$

$k_2 = hf(x_0 + h/2,\ y_0 + \frac{1}{2}k_1) = 0.1(0+0.05 + 1+0.05) = 0.1100$

$k_3 =$

$k_4 =$

$\therefore \quad y_1 = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) = 1.11034$

$n = 1, \quad x_1 = 0.1, \quad y_1 = 1.11034$

$k_1 =$ ---, $\quad k_2 =$ ---, $\quad k_3 =$ ---, $\quad k_4 =$ ---, $\qquad y_2 =$ --- $:= y_{0.1}(0.2)$

2) Using h = 0.2

$n = 0, \quad x_0 = 0, y_0 = 1$

$k_1 = 0.2(0+1) = 0.20$

$k_2 = 0.2(0.1 + 1.1) = 0.24$

$k_3 = \ldots$

$k_4 = \ldots$

$$\therefore \qquad y_1 = \bar{y}_1 = 1 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) = 1.242803 := y_{0.2}(0.2)$$

$$\text{Error} = \frac{y_{0.2}(0.2) - y_{0.1}(0.2)}{15} \approx -2\times 10^{-7}.$$

9.3 Multistep Methods

The methods covered so far for solving initial value problem $y' = f(x), \quad y(x_0) = y_0$, are one step methods.

Definition (One Step Methods): Methods of the form $y_{n+1} = g(x_n, y_n)$ which use information from the immediate last step to advance solution to next step are called one step methods.

Taylor series method, Euler method and Runge-Kutta methods are all one step methods.

Definition (Multistep Methods):

Methods of the form $y_{n+1} = g(x_n, y_n;\, x_{n-1}, y_{n-1};\, \dots\dots\dots;\, x_{n-p}, y_{n-p})$ which take account of previous information at $x_n, x_{n-1}, \dots\dots\dots, x_{n-p}$ to find y_{n+1} is called multistep methods.

Derivation of Multistep Formulae

Consider
$$\frac{dy}{dx} = f(x, y). \tag{9.20}$$

Integrating the above equation from x_{n-p} to x_{n+1} yields

$$y_{n+1} = y_{n-p} + \int_{x_{n-p}}^{x_{n+1}} f(x, y)dx \qquad p \ge 0. \tag{9.21}$$

From Newton's backward interpolation formula, we have

$$f(x, y) = P_m(s) + \text{Error} = \sum_{k=0}^{m} (-1)^k \binom{-s}{k} \nabla^k f_n + (-1)^{m+1} \binom{-s}{m+1} h^{m+1} f^{(m+1)}(\xi) \tag{9.22}$$

where P_m is the mth degree interpolating polynomial and

$$x = x_n + sh, \quad \xi(x) \in [x_{n-m}, x_{n+1}]. \tag{9.23}$$

Approximating $f(x, y)$ by $P_m(s)$, we obtain from (9.21)

$$y_{n+1} = y_{n-p} + h\int_{-p}^{1} P_m(s)ds, \tag{9.24}$$

with error

$$\mathrm{E} = h\int_{-p}^{1} (-1)^{m+1}\begin{pmatrix} -s \\ m+1 \end{pmatrix} f^{(m+1)}(\xi)\,ds\,. \tag{9.25}$$

In the following we derive a selection of formulae using different values of p and m.

Adams-Bashforth Formulae (p = 0)

Here consider m=3. From (9.22), we can construct a 3rd degree interpolating polynomial $P_3(s)$ to approximate $f(x, y)$ and estimate the interpolating error from information at x_n, x_{n-1}, x_{n-2}, and x_{n-3}, from which and (9.25), we obtain

$$\begin{aligned} y_{n+1} &= y_n + \int_0^1 \left(f_n + s\nabla f_n + \frac{s(s+1)}{2}\nabla^2 f_n + \frac{s(s+1)(s+2)}{3!}\nabla^3 f_n\right)h\,ds \\ &= y_n + f_n + \frac{1}{2}\nabla f_n + \frac{5}{12}\nabla^2 f_n + \frac{3}{8}\nabla^3 f_n\ . \end{aligned} \tag{9.26}$$

The following difference table can be used to assist in the calculation of the backward finite differences

$$\begin{array}{llllll}
x_{n-3} & y_{n-3} & f_{n-3} & & & \\
 & & & \nabla f_{n-2} = f_{n-2} - f_{n-3} & & \\
x_{n-2} & y_{n-2} & f_{n-2} & & \nabla^2 f_{n-1} = \nabla f_{n-1} - \nabla f_{n-2} & \\
 & & & \nabla f_{n-1} = f_{n-1} - f_{n-2} & & \nabla^3 f_n = \nabla^2 f_n - \nabla^2 f_{n-1} \\
x_{n-1} & y_{n-1} & f_{n-1} & & \nabla^2 f_n = \nabla f_n - \nabla f_{n-2} & \\
 & & & \nabla f_n = f_n - f_{n-1} & & \\
x_n & y_n & f_n & & &
\end{array}$$

from which

$$\nabla^2 f_n = \nabla f_n - \nabla f_{n-1} = [f_n - f_n] - [f_{n-1} - f_{n-2}] = f_n - 2f_{n-1} + f_{n-2}$$

$$\nabla^3 f_n = \nabla^2 f_n - \nabla^2 f_{n-1} = f_n - 3f_{n-1} + 3f_{n-2} - f_{n-3}$$

Substituting the above differences into (9.26) yields

$$\begin{aligned} y_{n+1} &= y_n + f_n + \frac{1}{2}(f_n - f_{n-1}) + \frac{5}{12}(f_n - 2f_{n-1} + f_{n-2}) + \frac{3}{8}(f_n - 3f_{n-1} + 3f_{n-2} - f_{n-3}) \\ &= y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) \end{aligned}$$

The error in the numerical approximation is

$$\begin{aligned} E &= \int_0^1 \frac{s(s+1)(s+2)(s+3)}{4!}h^4 f^{(4)}(\xi)\,hds \\ &= \frac{1}{24}h^5 f^{(4)}(\xi_1)\int_0^1 s(s+1)(s+2)(s+3)\,ds = \frac{251}{720}h^5 f^{(4)}(\xi_1) \end{aligned}$$

where $\xi_1 \in (x_{n-3},\, x_{n+1})$

In Summary Adams-Bashforth formulae with error is as follows

Adams-Bashforth Formulae: $y_{n+1} = y_n + \frac{h}{24}\left(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}\right)$

with error $E = \frac{251}{720}h^5 y^{(5)}(\xi), \qquad \xi \in [x_{n-3}, x_{n+1}].$

Midpoint formulae ($p = 1$, $m = 1$) (Comparable to Euler)

$$y_{n+1} = y_{n-1} + 2hf_n , \quad \text{with error } E = \frac{h^3}{3}y'''(\xi).$$

Milne's formulae ($p = 3$, $m = 3$)

$$y_{n+1} = y_{n-3} + \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}), \quad \text{with error } E = \frac{14}{45}h^5 y^{(5)}(\xi).$$

It can be shown that all formulae of type (9.21) with $m = p$ have the property that the coefficient of the mth difference ($\nabla^m f_n$) vanishes, and thus the formulae is of higher order than might be expected.

Advantages of Multi-step Methods

Require only one function evaluation per step. Compared to (say) Runge-Kutta order 4 (4 function evaluations), the method is considerably faster and less computational work.

Disadvantages of Multi-step Methods

– not self-starting
– Although A-B's discretization error is $O(h^5)$, the coefficient in the error term is somewhat larger than Runge-Kutta type of the same order. Runge-Kutta methods are generally (although not always) more accurate for this reason.

9.4 Predictor-Corrector Methods

$$\frac{dy}{dx} = f(x, y) \quad \Rightarrow \quad y_{n+1} = y_{n-p} + \int_{x_{n-p}}^{x_{n+1}} f(x, y)dx, \qquad p \ge 0$$

Corrector

If we approximate the integral on the right hand side by fitting an interpolating polynomial based on x_{n+1}, x_n and backwards, we can get an implicit formula for y_{n+1}. For example, taking $p = 0$ and using the Trapezoidal rule, we have

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})], \text{ with error } E = \frac{h^3}{12}y'''(\xi) \tag{9.27}$$

This error is smaller than that of Euler's method $\frac{h^2}{2}y''(\xi)$. Unfortunately, this equation in general is not immediately useful because we can not evaluate $f(x_{n+1}, y_{n+1})$ until y_{n+1} is known. Thus we need a predictor for y_{n+1}.

Predictor

If we approximate the integral on the right hand side by fitting an interpolating polynomial based on x_n, x_{n-1} and backwards, we can get an explicit formula for y_{n+1}. For example, taking $p = 1$ and using the Midpoint rule for the integral, we have

$$y_{n+1} = y_{n-1} + 2hf_n \,, \qquad \text{with error} \quad E = \frac{h^3}{3} y'''(\xi) \tag{9.28}$$

Predictor-Corrector Method

1) Use the explicit (predictor) formula (eg. equation (9.28)) to predict the value of $y(x_{n+1})$, i.e., $y^p{}_{n+1}$;
2) Use the implicit (corrector) formula (eg. equation (9.27)) to check or to correct the predicted value of $y(x_{n+1})$, i.e., $y^c{}_{n+1}$;
3) Repeat the above process for a new value of x.

Adams-Moulton Multistep Predictor-Corrector Method

Take $p = 0$, then $y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f(x, y)dx$

Predictor: Approximating $f(x, y)$ by using Newton-Gregory backward interpolating polynomial based on 4 points x_n, x_{n-1}, x_{n-2}, x_{n-3} , we obtain the Adams-Bashforth formula (see section 9.3) as predictor:

$$y^p_{n+1} = y_n + \frac{h}{24}\left(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}\right) \,.$$

Corrector: To derive a correct formula, approximate $f(x, y)$ by Newton-Gregory backward interpolating polynomial based on 4 points x_{n+1}, x_n , x_{n-1}, x_{n-2} ,

$$f(x) \approx P_3(s) = f_{n+1} + s\nabla f_{n+1} + \frac{s(s+1)}{2!}\nabla^2 f_{n+1} + \frac{s(s+1)(s+2)}{3!}\nabla^3 f_{n+1} \,,$$

where $x = x_{n+1}+sh$.

Thus,
$$y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f(x, y)dx = y_n + h\int_{-1}^{0} P_3(s)\,ds$$

$$y^c_{n+1} = y_n + \frac{h}{24}\left(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}\right) \,.$$

Remark: 1) The above process can be called *PEC* process (*P*: predictor, *E*: estimate function value $f(x_{n+1}, y^p{}_{n+1})$, *C*: Corrector for finding y^c_{n+1} ;

2) Having obtained a corrected estimate $y^c{}_{n+1}$ of $y(x_{n+1})$, it is sensible to use it to improve the estimate of $y(x_{n+1})$ before proceeding to the next step. Thus, one could use the *PEC* process: $y^p{}_{n+1} \rightarrow f(x_{n+1}, y^p{}_{n+1}) \rightarrow y^c{}_{n+1} \rightarrow f(x_{n+1}, y^c{}_{n+1})$. The corrector can also be used repeatedly to improve the estimate of $y(x_{n+1})$, i.e., use $P(EC)^m E$ process.

Convergence Criteria for Predictor-Corrector Method

Example 9.2 Suppose the corrector is $y_{n+1} = y_n + \frac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, y_{n+1})\right]$.

Obviously, the corrector step is an iteration of the form $y_n^{(k)} = F\left(y_n^{(k-1)}\right)$, $k = 1, 2, \ldots$ for finding the solution y_{n+1} to the above equation. We hope that the sequence generated will converge to the root, namely

$$\left(y_{n+1}^{p} = \right) y_{n+1}^{(0)},\ y_{n+1}^{(1)}, y_{n+1}^{(2)}, \ldots\ldots\ldots \to Y = F(Y).$$

The usual condition for convergence of a fixed-point iteration process applies, namely

$$\left|F'(y)\right| < 1$$

on the interval containing the root. Thus, for the above example, we have

$$y_{n+1} = C + \frac{h}{2} f(x_{n+1}, y_{n+1}) \quad \Rightarrow \quad F(Y) = \frac{h}{2} f(x_{n+1}, Y) + C$$

Hence, the condition for convergence is

$$\frac{h}{2}\left|\frac{\partial f}{\partial y}\right| < 1.$$

The above implies that it may be possible to adjust h in a non-convergent situation to produce convergence.

Discretization Error

We need to estimate truncation error at each step to reduce number of iterations of the corrector.

Example 9.3 P: $y_{n+1} = y_{n-1} + 2hf_n$, with error $E_{pm} = \frac{h^3}{3} y'''(\xi_1)$ – Midpoint rule

C: $y_{n+1} = y_n + \frac{h}{2}\left(y_n' + y_{n+1}'\right)$, with error $E_{Trap} = -\frac{h^3}{12} y'''(\xi_2)$ – Trapezoidal Rule

Let y_{n+1}^{p} be the value of y_{n+1}obtained from the predictor and y_{n+1}^{c} be the value of y_n obtained from the corrector, then.

$$\begin{cases} y(x_{n+1}) = y_{n+1}^{p} + \frac{h^3}{3} y'''(\xi_1) \\ y(x_{n+1}) = y_{n+1}^{c} - \frac{h^3}{12} y'''(\xi_2) \end{cases}$$

Assuming that the 3rd order derivative of $y(x)$ is firstly constant, then

$$D_{n+1} = Error = y(x_{n+1}) - y_{n+1}^{c} = -\frac{h^3}{12} y'''(\xi) = -\frac{1}{5}\left(y_{n+1}^{c} - y_{n+1}^{p}\right) \tag{9.29}$$

$$\therefore \quad \text{if } |D| = \begin{cases} < \varepsilon & \text{(some tolerance) then proceed to next integration step,} \\ \geq \varepsilon & \text{then halve step size, recompute starting value and restart.} \end{cases}$$

9.5 Stability of Numerical Methods

Inherent instability

Consider
$$\begin{cases} y'' - 10y' - 11y = 0 \\ y(0) = 1, \quad y'(0) = -1 \end{cases} \Rightarrow \quad y = e^{-x} \tag{9.30}$$

Suppose that there is a slight change in initial condition, i.e. $y(0) = 1+\varepsilon$, $y'(0) = -1$, the solution becomes

$$y = (1 + \frac{11}{12}\varepsilon)e^{-x} + \frac{\varepsilon}{12}e^{11x}$$

Obviously, for large x, a small change ε in initial condition will cause a large change in y. This kind of problems is called inherent instability.

Numerical Instability

Example 9.4 Consider $y' = -3y$, $y(0) = 1$ (stable problem).

Solution

The exact solution is $y = e^{-3x}$

Using the midpoint rule $y_{n+1} = y_{n-1} + 2hf_n$, (9.31)

we obtain the following numerical result

x_n	0	0.1	0.2	0.3	1.0	1.0	1.2	1.3	1.4
y_n	1	0.7408	0.5555	0.4075	0.0816	–0.0013	0.0824	–0.0507	0.1128

Remark: Growing oscillations show instability in the method and the Phenomenon exhibited here is known as numerical instability.

Definition: Numerical Instability

If a small perturbation on the starting values y_i $(0 < i < p)$ causes a large change in y_i $(p < i < N)$ obtained by a numerical method, then the numerical method is unstable. This kind of problems is called numerical instability.

Cause of Numerical Instability

To understand the numerical instability, let us examine the difference equation (9.31) more closely. For the example being considered $f_n = -3y_n$, and hence from the midpoint rule

$$y_{n+1} = y_{n-1} + 2hf_n = y_{n-1} + 2h(-3y_n)$$
$$\Rightarrow \quad y_{n+1} + 6hy_n - y_{n-1} = 0$$
$$\Rightarrow \quad (\mathrm{E}^2 + 6h\mathrm{E} - 1)y_{n-1} = 0,$$

where $\mathrm{E}y_n = y_{n+1}$ which is a 2nd order difference equation. The characteristic equation is

$$\lambda^2 + 6h\lambda - 1 = 0,$$

from which $\lambda = -3h \pm \sqrt{1+9h^2} = -3h \pm \left(1 + \frac{9}{2}h^2 + O(h^3)\right), \quad h < \frac{1}{3},$

where we have used the formulae $(1 \pm x)^{1/2} = 1 \pm \frac{1}{2}x + O(x^2)$ for $\|x\| \le 1$.

Thus $\lambda_1 = 1 - 3h + O(h^2), \quad \lambda_2 = -(1+3h) + O(h^2).$

Therefore, $y_n = C_1\lambda_1^n + C_2\lambda_2^n$.

As $x_n = nh$ is fixed,

$$\lambda_1^n = (1-3h)^n = [1 + (-3h)]^{x_n/h} = [1+(-3h)]^{1/(-3h).(-3x_n)} .$$

Further, as $\lim\limits_{\varepsilon \to 0} (1+\varepsilon)^{1/\varepsilon} = e,$

we have $\lim\limits_{h \to 0} \lambda_1^n = e^{-3x_n}$ and $\lim\limits_{h \to 0} \lambda_2^n = (-1)^n e^{3x_n}$.

Therefore, $y_n = C_1 e^{-3x_n} + C_2(-1)^n e^{3x_n}$. (9.32)

The first term e^{-3x_n} is a true solution. The 2nd term $(-1)^n e^{3x_n}$ is an extraneous term due to the fact that a 1st order differential equation is approximated by a second order difference equation.

Imposing the initial condition will, if all arithmetic operations are exact, result in choosing $C_2 = 0$ so that the correct solution will be selected from (9.32). In practice, however, some errors will be introduced, primarily due to round off or to inexact starting values and hence C_2 will not be exactly zero. A small error will therefore be introduced at each step of the integration, and thus error will subsequently be magnified exponentially. As the major part of the solution decreases exponentially, the error will eventually dominate the solution.

Loosely speaking, we can say that a method is unstable if errors introduced into the calculations grow at an exponential rate as the computation proceeds.

One-step methods like Runge-Kutta type do not exhibit any numerical instability for *h* sufficiently small. Multistep methods may, in some cases, be unstable for all values of h and in other cases for a range of values of h.

Method for Analyzing the Stability of Multistep Methods

If the multistep method leads to a difference equations of order *k*, find the roots of the characteristic equation corresponding to the homogeneous difference equation. Denoting these roots as β_i $(i = 1, \ldots k)$, then the general solution is

$$y_n = C_1\beta_1^n + C_2\beta_2^n + \ldots + C_k\beta_k^n$$

One of these solutions, say β_1^n, will tend to the exact solution of the differential equation as $h \to 0$. All the other solutions are extraneous. Obviously, if $|\beta_i| < 1$ $(i = 2, 3, \ldots, k)$ then any errors introduced into the computation will decay as n increases. Whereas if any of the extraneous $|\beta_i| > 1$, the errors will grow exponentially.

Definiton: Strongly Stable

A multistep method is defined to be strongly stable if the extraneous roots satisfy as $h \to 0$ the conditions

$$|\beta_i| < 1 \quad (i = 2, 3, \ldots, k).$$

For the general differential equation $y' = f(x, y)$, it will be difficult to obtain the roots β_i of the characteristic equation. However, as $y' = f(x, y)$ can be expanded in the neighborhood of $x = x_n$ as

$$y'(x) = f(x_n, y_n) + (x - x_n)\frac{\partial f}{\partial x} + (y - y_n)\frac{\partial f}{\partial y} + \ \ldots \ = \lambda y + \mu x + C\,,$$

we usually test for stability of a numerical technique on $y' = \lambda y$ to give an indication of the stability of a method.

Example 9.5 For Milne's method $y_{n+1} = y_{n-1} + \frac{h}{3}(f_{n+1} + 4f_n + f_{n-1})$.

If we set $f = \lambda y$, then

$$(1-\gamma)y_{n+1} - 4\gamma y_n - (1+\gamma)y_{n-1} = 0, \quad \text{where } \gamma = \lambda h / 3.$$

The associated *Characteristic equation is*

$$(1-\gamma)\beta^2 - 4\gamma\beta - (1+\gamma) = 0,$$

which yields

$$\beta = \frac{1}{1-\gamma}\left[2\gamma \pm \sqrt{1+3\gamma^2}\right].$$

From the Taylor series, we have

$$\begin{aligned}(1+3\gamma^2)^{1/2} &:= f(\gamma) = f(0) + \gamma f'(0) + O(\gamma^2)\\ &= 1 + O(\gamma^2),\\ (1-\gamma)^{-1} &:= g(\gamma) = g(0) + \gamma g'(0) + O(\gamma^2)\\ &= 1 + \gamma[(1-\gamma)^{-2}]_{\gamma=0} + O(\gamma^2)\\ &= 1 + \gamma + O(\gamma^2),\end{aligned}$$

and thus

$$\begin{aligned}\beta &= \left(1 + \gamma + O(\gamma^2)\right)\left(2\gamma \pm (1 + O(\gamma^2))\right)\\ &= \left(2\gamma \pm (1+\gamma) + O(\gamma^2)\right).\end{aligned}$$

Therefore on using $\gamma = \frac{\lambda h}{3}$, we have

$$\beta_1 = 1 + \lambda h + O(h^2), \qquad \beta_2 = -(1 - \frac{\lambda h}{3}) + O(h^2).$$

Hence, the general solution is

$$y_n = C_1\left[1+\lambda h+O(h^2)\right]^n + C_2(-1)^n\left[1-\frac{\lambda h}{3}+O(h^2)\right]^n$$

$$\lim_{h\to 0} y_n = C_1 e^{\lambda x_n} + C_2(-1)^n e^{-\lambda x_n/3},$$

where $x_n = nh$.

The extraneous term is $C_2(-1)^n e^{-\lambda x_n/3}$ and thus stability depends on the sign of λ. If $\lambda>0$, the desired solution will increase exponentially, while the extraneous term decreases exponentially. Thus Milne's method in this case will be stable. On the other hand if $\lambda<0$, the method is unstable.

Definition: Weakly Stable

Methods whose stability depends on the sign of λ for the test equation $y' = \lambda y$ is said to be weakly stable.

For the more general equation $y' = f(x, y)$, we can expect weak stability from Milne's method whenever $\partial f / \partial y < 0$ on the interval of integration.

Eg. $y' = -3y$ by Milne's method is unstable since $\dfrac{\partial f}{\partial y} = -3 < 0$.

Interval of Absolutely Stable

In practice, all multistep methods will exhibit some instability for some range of values of the step size h. As described before, the roots of the characteristic equation are

$$\beta_1(\lambda h),\ \beta_2(\lambda h),\ \ldots,\ \beta_n(\lambda h).$$

These roots depend on λh. If for certain interval of λh, $|\beta_i(\lambda h)| < 1$ ($i = 2, 3, \ldots, k$), then the error due to the extraneous terms tends to zero.

Definition- Interval of Absolute Stability

All the values of λh for which all the roots of the characteristic equation are less than one in magnitude.

Relatively Stable

For equations of the form $y' = \lambda y$ where $\lambda > 0$. The solution of the difference equation associated with a multistep method is

$$y_n = C_1\beta_1^n + C_2\beta_2^n + \ldots + C_k\beta_k^n$$

One of the term (say $C_1\beta_1^n$) will be growing exponentially. If all the other terms grow slower than the $C_1\beta_1^n$ term, the method still can be stable.

Definition: (Relatively stable Method)

A method which has the property that all extraneous roots of the characteristic equation are less than the principal root in magnitude is said to be relatively stable.

9.6 Systems of First Order Initial Value Problems

In this section, we describe how the methods in sections 9.1-9.4 for solving a 1st order initial value problem can be adapted to solve a system of first order equations

$$\begin{cases} y_1' = f_1(x, y_1, y_2, \dots, y_m) \\ y_2' = f_2(x, y_1, y_2, \dots, y_m) \\ \vdots \\ y_m' = f_1(x, y_1, y_2, \dots, y_m) \end{cases} \quad \text{subject to} \quad \begin{cases} y_1(a) = C_1 \\ y_2(a) = C_2 \\ \vdots \\ y_m(a) = C_m \end{cases}$$

which can be written in vector form

$$\begin{cases} \mathbf{y}' = \mathbf{f}(\mathrm{x}, \mathbf{y}) \\ \mathbf{y}(\mathbf{a}) = \mathbf{C}. \end{cases}$$

Numerical Solutions

Any of the numerical methods described in section 9.1-9.4 can be adapted to solve the above system. For example to solve the folowing system using Runge-Kutta method (order 4)

$$\begin{cases} y' = f(x, y, z) \\ z' = g(x, y, z) \end{cases} \quad \text{subject to} \quad \begin{cases} y(x_0) = y_0 \\ z(x_0) = z_0 \end{cases}$$

we first divide the interval $[a, b]$ into N subintervals with nodes $a = x_0, x_1, \dots, x_N = b$. Then for each x_i, we compute $\begin{cases} y_i & \rightarrow \text{ approximating } y(x_i) \\ z_i & \rightarrow \text{ approximating } z(x_i) \end{cases}$ *using*

$$\begin{aligned} k_1 &= hf(x_n, y_n, z_n), & l_1 &= hg(x_n, y_n, z_n), \\ k_2 &= hf(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_1, z_n + \frac{1}{2}l_1), & l_2 &= hg(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_1, z_n + \frac{1}{2}l_1), \\ k_3 &= hf(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_2, z_n + \frac{1}{2}l_2), & l_3 &= hg(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_2, z_n + \frac{1}{2}l_2), \\ k_4 &= hf(x_n + h, y_n + k_3, z_n + l_3), & l_4 &= hg(x_n + h, y_n + k_3, z_n + l_3) \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), & z_{n+1} &= z_n + \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4). \end{aligned}$$

9.7 Higher Order Initial Value Problems

The general form of the mth order initial value problems is

$$\begin{cases} y^{(m)} = f(x, y, y', \dots, y^{(m-1)}) \\ y(a)=C_1, \ y'(a) = C_2, \dots, y^{(m-1)}(a) = C_m. \end{cases}$$

To solve the above problem, we convert the problem into a system of differential equations and then solve as so, as detailed below.

Let $\quad y_1 = y, \quad y_2 = y_1', \quad y_3 = y_2', \; \dots, \; y_m = y_{m-1}'(x)$.

Then

$$y_m' = (y_{m-1}')' = y_{m-1}'' = (y_{m-2}')'' = y_{m-2}^{(3)} = \; \dots = y_{m-(m-2)}^{(m-1)} = (y_1')^{(m-1)} = y_1^{(m)} = f$$

and hence the problem becomes

$$\begin{cases} y_1' = y_2 \\ y_2' = y_3 \\ \vdots \\ y_{m-1}' = y_m \\ y_m' = f(x, y_1, y_2, \dots, y_m) \end{cases} \qquad \text{subject to} \qquad \begin{cases} y_1(a) = C_1 \\ y_2(a) = C_2 \\ \vdots \\ y_m(a) = C_m \end{cases}$$

or

$$\begin{cases} \mathbf{y}' = \mathbf{f} \\ \mathbf{y}(\mathbf{a}) = \mathbf{C} \end{cases} \qquad \text{with } f_j = \begin{cases} y_{j+1}, & j = 1, 2, \dots\; m-1 \\ f, & j = m \end{cases}$$

which is a system of 1st order equations and thus can be solved using the methods in section 9.6.

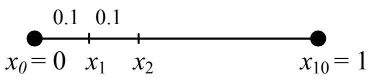
Example 9.6 Solve the 2nd order initial value problem. $\begin{cases} y'' + y = 0 & x \in (0,\, 1) \\ y(0) = 1, \;\; y'(0) = 0 \end{cases}$

using Euler's method with $h = 0.1$.

Solution As $h = 0.1$, the solution domain is divided into 10 sub-regions with 11 nodes x_i. The problem is to generate numerical solution $\{x_i, y_i\}(i=1,10)$ from the given initial data.

Firstly, we put $y' = z$, then the problem becomes

$$\begin{cases} y' = z & := f \\ z' = -y & := g \\ y(0) = 1, \;\; z(0) = 0 \end{cases}$$



Now, we can use Euler's method to solve the problem, i.e.,

$$\begin{cases} y_{n+1} = y_n + hf_n & = y_n + 0.1 z_n \\ z_{n+1} = z_n + hg_n & = z_n - 0.1 y_n \end{cases}$$

For $n = 0$: $\quad y_0 = 1, \quad z_0 = 0 \qquad$ (as given in the problem)

$\therefore \quad y_1 = y_0 + 0.1\, z_0 = 1 + 0.1*0 = 1$

$\quad z_1 = z_0 - 0.1\, y_0 = 0 - 0.1*1 = -0.1$

For $n = 1$: $\quad y_2 = y_1 + 0.1\, z_1 = 1 + 0.1*(-0.1) = 0.99$

$\quad z_2 = z_1 - 0.1\, y_1 = -0.1 - 0.1*1 = -0.2$

$\vdots$

9.8 Solution of Initial Value Problems Using Maple/MATLAB

Solution of Initial Value Problems Using Maple

The Maple function "**dsolve**()" can be used to find the solution of initial value problems and evaluate the value of the solution at any given point. The syntax is

```
dsolve(ODEs, numeric,method=rkf45,vars,output=procedurelist)
```

where `ODEs` : a set of ordinary differential equations and initial conditions y(a)=b, D(y)(a)=c, etc.

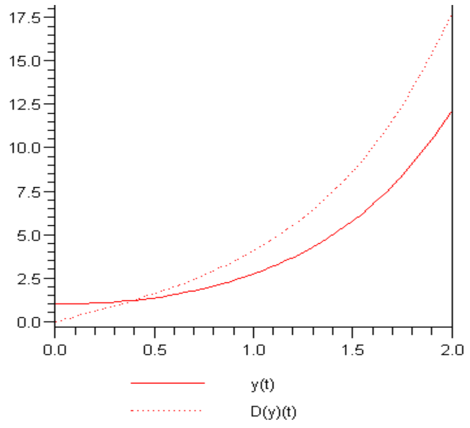
`vars` : (optional) unknowns of the ODE problem (one variable, or a set or list of them).

Example 1 Solve $\begin{cases} y'' = 2y + 1 \\ y(0) = 1, y'(0) = 0 \end{cases}$

```
> dsys := {diff(y(x),x,x) = 2*y(x) + 1, y(0)=1, D(y)(0)=0};
> dsol := dsolve(dsys, numeric,method=rkf45,
          output=procedurelist);
> dsol(0.4);
> soly:=t->rhs(dsol(t)[2]);
> solDy:=t->rhs(dsol(t)[3]);
> with(plots): setoptions(axes=BOXED);
> p1:= plot(soly, 0..2, linestyle = SOLID, legend = "y(t)");
> p2:= plot(solDy, 0..2, linestyle = DOT, legend = "D(y)(t)");
> plots[display]({p1,p2});
```

yields

$$\left[t = 0.4, y(t) = 1.24646863729600832, \frac{d}{dt} y(t) = 1.26503188892540352 \right]$$

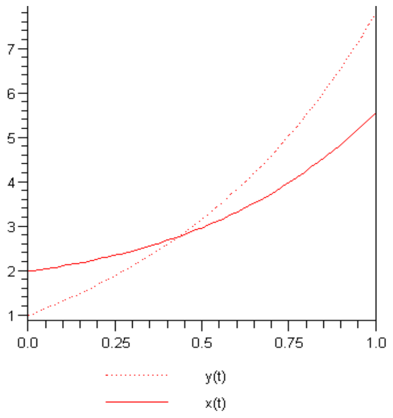


Example 2 Solve $\begin{cases} x' = y \\ y' = x + y \\ x(0) = 2, y(0) = 1 \end{cases}$

```
> dsys:={diff(x(t),t)=y(t), diff(y(t),t)=x(t)+y(t), x(0)=2,
        y(0)=1};
> dsol:=dsolve(dsys, numeric, method=rkf45,
        output=procedurelist):dsol(0.4);
> solx:=t->rhs(dsol(t)[2]);
> soly:=t->rhs(dsol(t)[3]);
> with(plots): setoptions(axes=BOXED);
> p1:= plot(solx, 0..2, linestyle = SOLID, legend = "x(t)");
> p2:= plot(soly, 0..2, linestyle = DOT, legend = "y(t)");
> plots[display]({p1,p2});
```

yields

$$\left[\, t{=}0.4,\, x(t){=}2.69118459002265676,\, y(t){=}2.60811749186656749 \right]$$



Solution of Initial Value Problems Using MATLAB

There are 7 built-in routines, `ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t` and `ode23tb`, which solve initial value problems for ordinary differential equations.   The syntax is

```
[T,Y] = solver(odefun, tspan, y0)
[T,Y] = solver(odefun, tspan, y0, option)
[T,Y,TE,YE,IE] = solver(odefun, tspan, y0, option)
sol = solver(odefun, [t0 tf], y0…)
```

where *solver* should be replaced by one of the 7 built-in routine names listed above.

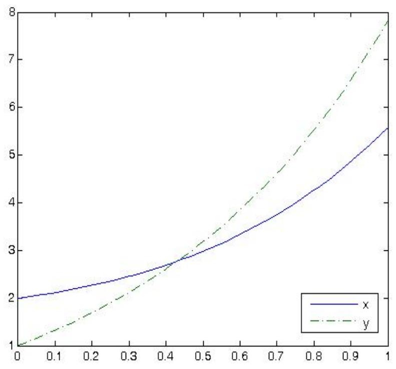
Example Solve $\begin{cases} x' = y \\ y' = x + y \\ x(0) = 2, y(0) = 1 \end{cases}$

To solve this system, first create a function "**`dsys()`**" containing the equations.

```
function  dy = dsys(t, y)
dy = zeros(2,1);   % a column vector
dy(1)  = y(2);
dy(2)  = y(1)+y(2);
```

Then solve the above system on a time interval [0 1] with an initial vector [2 1] at time *t*=0 and plot the results.

```
>> [T,Y]=ode45(@dsys, [0 1], [2 1]);
>> plot(T,Y(:,1),'-',T,Y(:,2),'-.')
```



EXERCISES 9

Q9.1 Use Euler's method to approximate the solution for each of the following initial value problems.

(a) $y' = \sin x + e^{-x}, \quad 0 \le x \le 1, \quad y(0) = 0$ with $h = 0.5$.

(b) $y' = \left(\dfrac{y}{x}\right)^2 + \dfrac{y}{x}, \quad 1 \le x \le 1.2, \quad y(1) = 1$ with $h = 0.1$.

Q9.2 Use Taylor's method of order four with $h = 0.1$ to approximate the solution to

$y' = -y + x + 1, \quad x \in [0, 0.2], \quad y(0) = 1.$

(Ans: y(0.1) = 1.00484, y(0.2) = 1.01873)

Q9.3 Use Taylor's method of order two with h = 0.1 to approximate the solution to the initial value problem in Q9.1(b).

(Ans: y(1.1) = 1.214999, y(1.2) = 1.465250)

Q9.4 Given $y' = \left(\frac{y}{x}\right)^2 + \frac{y}{x}, \quad y(1) = 1.$

(a) Determine y(0.2) using h = 0.1 by Runge-Kutta 4th order method.
(b) Determine y(0.2) using h = 0.2 by Runge-Kutta 4th order method.
(c) Use the results obtained with the two different step sizes to estimate the error committed in (a).

(Ans: (a) 1.4675, (b) 1.4674, (c) 0.000067

Q9.5 Describe how the Runge-Kutta (4th order) method can be used to produce a table of values for the function $f(x) = \int_0^x e^{-t^2}\,dt$ at 100 equally spaced points in [0, 1]

(Hint: find an approximate initial value problem whose solution is $f(x)$)

Q9.6 Show that the Runge-Kutta (4th order) formula reduces to a simple form when applied to an ordinary differential equation of the form $y' = f(x)$.

***Q9.7** Derive the Runge-Kutta 4th order formula.

***Q9.8** The Runge-Kutta order two method for $y' = f(x)$, $y(a) = y^0$ is given by

$$y_{n+1} = y_n + h\phi(x_n, y_n, h)$$

where $\phi(x, y, h) = a_1 f(x, y) + a_2 f\left[x + p_1 h, y + p_2 h f(x, y)\right].$

Show that the local truncation error term is

$$h^3\left(\frac{1}{6} - \frac{p_1}{4}\right)\left(\frac{\partial}{\partial x} + f\frac{\partial}{\partial y}\right)^2 f + \frac{h^3}{6} f_y\left(\frac{\partial}{\partial x} + f\frac{\partial}{\partial y}\right) f .$$

Q9.9 Derive the following multistep formula

(a) $y_{n+1} = y_{n-3} + \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2})$ Milne's formula

(b) $y_{n+1} = y_{n-1} + \frac{h}{3}(8f_n - 5f_{n-1} + 4f_{n-2} - f_{n-3})$

(Hint: (a) p = 3, m = 3; (b) p = 1, m = 3)

Q9.10 Solve the initial-value problem

$$xy' = x - y, \qquad y(2) = 2$$

from $x = 2$ to $x = 2.3$ using $h = 0.05$ and Adam-Bashforth method. Obtain the starting values from exact solution $y(x) = \frac{x}{2} + \frac{2}{x}$

Q9.11 Solve the initial-value problem

$$xy' = x - y, \qquad y(2) = 2$$

from $x = 2$ to $x = 2.3$ using $h = 0.05$ and Milne's formula in Q9.9(a). Obtain the starting values using Euler's formula.

Q9.12 Derive the following Predictor-Corrector formula and corresponding truncation errors

$$y_{n+1}^{p} = y_n + \frac{h}{2}(3f_n - f_{n-1}), \qquad E_1 = \frac{5}{12}h^3 y'''(\xi_1)$$

$$y_{n+1}^{c} = y_n + \frac{h}{2}(f_n + f_{n+1}), \qquad E_2 = -\frac{1}{12}h^3 y'''(\xi_2)$$

Q9.13 Derive the local error estimate for the Milne Predictor-Corrector formulae

$$y_{n+1}^{p} = y_{n-3} + \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}), \qquad E_1 = \frac{14}{45}h^5 y^{(5)}(\xi_1)$$

$$y_{n+1}^{c} = y_{n-1} + \frac{h}{3}(f_{n+1} + 4f_n + f_{n-1}), \qquad E_2 = -\frac{1}{90}h^5 y^{(5)}(\xi_2)$$

Q9.14 Use the Predictor-Corrector in Q9.4(a) to find a solution for $y' = 1 - y$ at $x = 0.2$ given $y(0) = 0$ and $y(0.1) = 0.09515$. Estimate the error.

Q9.15 Solve $y' = y + x^2$, $y(0) = 1$, from $x = 0$ to $x = 2$ with $h = 0.1$ using Adams-Moulton Predictor-Corrector method. The starting values, corrected to six decimal places are $y(0) = 1.000000$, $y(0.1) = 1.105513$, $y(0.2) = 1.224208$, $y(0.3) = 1.359576$. Compute the discretization error D_{n+1}, and estimate the error at $x = 2$.

Q9.16 Find the region of values for h over which

$y_{n+1}^{(k)} = y_n + \frac{h}{24}\left(9f\left(x_{n+1}, y_{n+1}^{(k-1)}\right) + 19f_n - 5f_{n-1} + f_{n-2}\right)$, $k = 1, 2, \ldots$, and x_n fixed, converges.

Q9.17 Show that the correct formula

$y_{n+1} = y_n + \frac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, y_{n+1})\right]$ is stable for equations of the form $y' = \lambda y$.

Q9.18 Determine the interval of absolute stability for Euler's method.

Q9.19 Consider $\begin{cases} y' = yz + x \\ z' = xz + y \\ y(0) = 1, \quad z(0) = -1 \end{cases}$

Find y(0.1), $z(0.1)$, $y(0.2)$ and $z(0.2)$ using Runge-Kutta method (order 4) with $h = 0.1$.

Programming

Q9.20 Write a subroutine for solving the 1st order initial value problem $\begin{cases} y' = f(x, y), & x \in [a, b] \\ y(a) = y_0 \end{cases}$ using Taylor series method of order four. Then test your subroutine by solving $y' = -y + x + 1, \quad 0 \le x \le 1, \ y(0) = 1.$

Compare your result with the exact solution $y(t) = t + e^{-t}$.

Hint: 1. Taylor series method of order 4:

$$y_{n+1} = y_n + hy_n' + \frac{h^2}{2!}y_n'' + \frac{h^3}{3!}y_n''' + \frac{h^4}{4!}y_n^{(4)} = y_n + hT(x_n, y_n, h).$$

2. Define $T(x, y, h)$ as an external function.

3. Solve the initial value problem using a subroutine with header.

SUBROUTINE TAYLOR4 (A, B, N, Y0, XI, YI)

where	A, B, N, Y0	– input
	XI(N), YI(N)	– output of the numerical solution $(x_i, y_i), \ i = 1, N$.

<u>Algorithm</u>

```
H = (B–A )/N
X = A
Y = Y0
FOR I = 1 TO N DO
    Y = Y+H*T (X, Y, H)
    X = X+H
    YI(I)=Y
    XI(I)=X
RETURN
```

4. Write a main program which reads A, B, N, Y0, calls Taylor4 to find the numerical solution, and prints the result).

Q9.21 Using the program developed in Q9.20 as a guide, write a program based on the improved Euler method to solve the 1st order initial value problem in Q9.4.

Q9.22 Using the program developed in Q9.20 as a guide, write a program based on the Runge-Kutta method (order 4) to solve the 1st order initial value problem in Q9.4.

Q9.23 Write a program which use the predictor-corrector pair in Q9.12 to solve the 1st order initial value problem

$$\begin{cases} y' = 1 - y, & 0 \le x \le 1 \\ y(0) = 0 \end{cases}$$

Obtain the starting values using the Euler's or Runge-Kutta methods. Use step size $h = 0.1$.

Hint: 1) Write an external function $F(x, y)$ to define $f(x, y) = 1 - y$.

2) Write a subroutine PCorrector_to solve an initial-value problem using the pair in Q9.16 given (x_0, y_0), (x_1, y_1), H, N, Tol

SUBROUTINE PCORRECTOR (X0, Y0, XI, YI, H, N, Tol, Status)
where Status_– output. Status = 'hlarge' if step size is too large.

```
Algorithm (PECE process)
    Set f0 = f(x0, y0)
        f1 = f(x1, y1)
        x = x1
        yc = y1
    For  i =1  to N – 1
        Set x = x+h
        yp = yc+h/2*(3f1 – f0)  ——— P
        f2 = f(x, yp)           ——— E
        yc = yc+h/2*(f1 + f2)   ——— C
        f2 = f(x, yc)           ——— E
        error =…
        if (error > Tol) then
            status= ‘hlarge’
            return
        else
            call xyout(x, yc)
        f0 = f1
        f1 = f2
    Return
```

3) Write a main program which reads a, b, y0, Tol, N; calls a subroutine EULER to calculate the starting values (x_0, y_0), (x_1, y_1) for the sub PCorrector, and calls the subroutine PCorrector to calculate (xi, yi), i = 1 to N.

```
Algorithm:
    Input a, b, y0, Tol, N
    h = (b – a)/N
    x0 = a
    call Euler(a, b, h, y0, xi, yi)
    x1 = xi(1)
    y1 = yi(1)
    call Pcorrector(x0, y0, x1, y1, h, N, Tol, Status)
    if (status = ‘hlarge’) then
        output (‘h too large’)
        stop
    end
```

Q9.24 Write a program based on the Runge-Kutta method (order 4) to solve the system of 1st order equations in Q9.23. Choose $a = 0$, $b = 1$, $h = 0.1$.

SUBROUTINE RK4TWO (a, b, y0, z0, N)

Q9.25 Solve the initial value problems in Q9.1, Q9.4 and Q9.20 using Maple/Matlab built-in functions

CHAPTER 10

Solution of Boundary Value Problems for Ordinary Differential Equations

For differential equations of order *m* (greater than 1), *m* conditions must be given to specify the values of the unknown function and/or its derivatives in order to obtain a unique solution. In the problems discussed so far, all the conditions are given at the same value of *x*, generally at the start of the interval of *x*. This kind of problems is called initial value problem.

If the conditions are given to specify the function and/or its derivatives at more than one point (usually end points), the problem is referred to as a boundary value problem.

In this chapter, we study three types of methods for solving second order two point boundary value problems

$$\begin{cases} y'' = f(x,y,y'), \\ y(a) = A, y(b) = B. \end{cases} \tag{10.1}$$

10.1 Shooting Method for Two Point Boundary Value Problems

Let z = y', then the two point boundary value problem (10.1) becomes

$$\begin{cases} y' = z \\ z' = f(x,y,z) \\ y(a) = A, \quad y(b) = B \end{cases} \tag{10.2}$$

Idea Behind the Shooting Method

By putting aside the condition *y*(*b*)=*B* and imposing a condition *z*(*a*)=α estimating the value of *z*(*a*) (i.e. y'(a)), we can form an initial value problem:

$$\begin{cases} y' = z \\ z' = f(x,y,z) \\ y(a) = A, \quad z(a) = \alpha \end{cases} \tag{10.3}$$

which can be solved using the methods in section 9.6 to yield a numerical solution $\{(x_i, y_i, z_i)\}_{i=1}^{N}$ starting from (x_0, A, α) as shown below

$$\begin{array}{c|cccc} (a=)\, x_0 & x_1 & x_2 & \cdots & x_n (=b) \\ \hline A & y_1 & y_2 & \cdots & \boxed{y_n} \\ \boxed{\alpha} & z_1 & z_2 & \cdots & z_n \end{array} \tag{10.4}$$

Obviously, the y_n value in (10.4) depends on the α value chosen and so we can write

$$y_n = y_n(\alpha) \approx y(b)$$

to indicates that it is the numerical approximation of $y(b)$ obtained by using $z(a) = \alpha$.

It is also clearly that the solution (10.4) of the initial value problem (10.3) automatically satisfies all the equations/conditions of the boundary value problem (10.2) except for the condition $y(b)$=B which was put aside before. However, if the α value is so chosen that

$$y_n(\alpha) \approx B$$

then the solution (10.4) satisfies all the equations and conditions of (10.2) and hence is the solution of the given boundary value problem.

Hence, the problem becomes to determine the α value which leads to $y_n(\alpha) \approx B$, then solve the equivalent IVP (10.3) using the α value determined, and then use the solution as the solution of the boundary value problem.

The shooting method is a method for finding the α value which leads to $y_n(\alpha) \approx B$ through an iteration process.

Algorithm and Formulation

1) Guess $z(a) = \alpha_1$, then solve the initial value problem

$$\begin{cases} y' = z \\ z' = f(x,\ y,\ z) \\ y(a) = A, \quad z(a) = \alpha_1 \end{cases} \tag{10.5}$$

to yield the solution $\{(x_i, y_i, z_i)\}_{i=1}^{N}$ with $y_n(\alpha_1) = B_1$ as shown in Figure 10.1

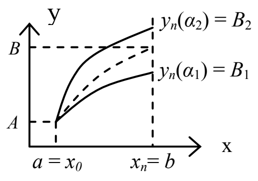


Figure 10.1

2) Guess $z(a) = \alpha_2$, then solve the initial value problem

$$\begin{cases} y' = z \\ z' = f(x,\ y,\ z) \\ y(a) = A, \quad z(a) = \alpha_2 \end{cases} \tag{10.6}$$

to yield the solution $\{(x_i, y_i, z_i)\}_{i=1}^{N}$ with $y_n(\alpha_2) = B_2$.

.

3) Find a new estimate to $z(a)$ (say α_3) such that $y_n(\alpha_3) \approx B$

The given conditions are

$$y_n(\alpha_1) = B_1 \neq B \ , \quad y_n(\alpha_2) = B_2 \neq B$$

which represent two points (α_1, B_1) and (α_2, B_2) in the $y_n(\alpha)$ curve as shown in Figure 10.2.

The problem is to find α where $y_n(\alpha) \approx B$. For this, we approximate the curve by a straingt line passing through the two given points and use α_3 to approximate α. From the diagram, we have

$$\tan\theta = \frac{B_2 - B_1}{\alpha_2 - \alpha_1} = \frac{B - B_1}{\alpha_3 - \alpha_1} \quad \Rightarrow \quad \alpha_3 = \alpha_1 + (\alpha_2 - \alpha_1)\frac{B - B_1}{B_2 - B_1} \tag{10.7}$$

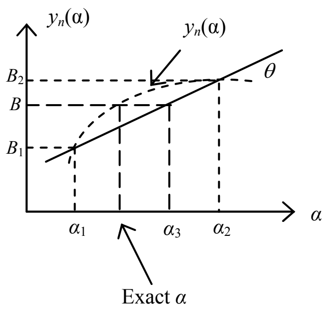


Figure 10.2 Linear interpolation of the function $y_n(\alpha)$

4) Solve initial value problem

$$\begin{cases} y' = z, \\ z' = f(x, y, z), \\ y(a) = A, \quad z(a) = \alpha_3, \end{cases} \tag{10.8}$$

to yield the solution $\{(x_i, y_i, z_i)\}_{i=1}^{N}$ with $y_n(\alpha_3) = B_3$.

5) If $|B_3 - B| < Tol$ or $|B_3 - B_2| < Tolerance$ then

take $\{(x_i, y_i, z_i)\}_{i=1}^{N}$ as the solution of the BVP and stop

else

update $\alpha_1 \leftarrow \alpha_2, \ B_1 \leftarrow B_2$

$\alpha_2 \leftarrow \alpha_3, \ B_2 \leftarrow B_3$

and goto step 3

Example 10.1 Solve the following two-point boundary value problem

$$\begin{cases} y'' = \dfrac{1}{50} y^3 & x \in (0,\ 2) \\ y(0) = 1,\ \ \mathrm{y}(2) = \dfrac{5}{6} \end{cases}$$

Using the shooting method starting with $\alpha_1 = -0.09$, $\alpha_2 = -0.1$ and $h = 1$.
Iterate until $\left| y_n(\alpha) - y(2) \right| \le 0.001$.

Solution

As $h = 1$, the solution domain is divided into 2 regions with 3 nodes. The problem is:

Given y_0 and y_2, find the numerical solution of y at other nodes, in this case we only need to find y_1.

Firstly, we put $y' = z$, then the problem becomes to solve the boundary value problem

$$\begin{cases} y' = z, \\ z' = 0.02 y^3, \\ y(0) = 1,\ \ \mathrm{z}(2) = \dfrac{5}{6}. \end{cases} \tag{10.9}$$

To solve this problem, we consider the differential equations and the first boundary condition but put aside the second boundary condition and add in one additional initial condition, i.e., consider

$$\begin{cases} y' = z, \\ z' = 0.02 y^3, \\ y(0) = 1,\ \ \mathrm{z}(0) = \alpha. \end{cases} \tag{10.10}$$

This problem can now be solved to yield numerical solution $\{(x_i,\ y_i)\}_{i=1}^{2}$ from (x_0, y_0). If α is such chosen that the solution of (10.10) satisfies the boundary condition which we put aside before, then this solution is the solution of the boundary value problem, as it satisfies all conditions in (10.9).

However, in general for an arbitrary chosen α, the solution from the initial value problem (10.10) will not satisfy the boundary condition which has been put aside. Therefore, we need an iterative process to determine the approximate α, such that the solution obtained from the initial value problem satisfies

$\left| y_N(\alpha) - y(b) \right| \le Tol$. For the current problem we require $\left| y_2(\alpha) - \dfrac{5}{6} \right| \le 0.001$.

Iteartion 1: Choose $\alpha = \alpha_1 = -0.09$, then solve (10.10) with $z_0 = \alpha = \alpha_1$

Using Euler's method $\begin{cases} y_{n+1} = y_n + hf_n & = y_n + z_n \\ z_{n+1} = z_n + hg_n & = z_n + 0.02 y_n^3 \end{cases}$

$y_0 = 1,\ z_0 = -0.09$
$y_1 = y_0 + z_0 = 1 - 0.09 = 0.91$
$z_1 = z_0 + 0.02\ y_0^3 = -0.09 + 0.02*1^3 = -0.07$
$y_2 = y_1 + z_1 = 0.91 - 0.07 = 0.84 \quad (=:B_1)$

As $\left|y_2(\alpha_1) - y(x_2)\right| = \left|0.84 - \frac{5}{6}\right| > 0.001$, continue iteration.

Iteartion 2: Choose α = α$_2$ = – 0.11 and then solve (10.10) with α = α$_2$

$y_0 = 1,\ z_0 = -0.11$
$y_1 = y_0 + z_0 = 1 - 0.11 = 0.89$
$z_1 = z_0 + 0.02\ y_0^3 = -0.11 + 0.02*1^3 = -0.09$
$y_2 = y_1 + z_1 = 0.89 - 0.09 = 0.80 \quad (=:B_2)$

As $\left|y_2(\alpha_1) - y(x_2)\right| = \left|0.80 - \frac{5}{6}\right| > 0.001$, go to the next iteration

Iteartion 3: Determine α$_3$ using the formula 10.7,

$$\alpha_3 = \alpha_1 + (\alpha_2 - \alpha_1)\frac{B - B_1}{B_2 - B} = -0.09 + \frac{\frac{5}{6} - 0.84}{0.80 - 0.84} = -0.093$$

Then solve (10.10) with α = α$_3$ = z_0

$y_1 = y_0 + z_0 = 1 - 0.093 = 0.907$
$z_1 = z_0 + 0.02\ y_0^3 = -0.093 + 0.02*1^3 = -0.073$
$y_2 = y_1 + z_1 = 0.907 - 0.073 = 0.834$

Now, $\left|y_2(\alpha_1) - y(x_2)\right| = \left|0.834 - \frac{5}{6}\right| < 0.001$.

Hence the solution of the initial value problem (10.10) with α=α$_3$ is the solution of the given boundary value problem and hence $y(1)= y_1= 0.907$is the solution of the given boundary value problem .

10.2 Finite Difference Methods

This method enables us to replace a differential equation by a system of algebraic equatoins. If the differential equation is nonlinear, the algebraic equaltions will also be non-linear.

Consider

$$\begin{cases} y'' + p(x)y' + q(x)y = r(x) \\ y(a) = A, \quad y(b) = B \end{cases} \tag{10.11}$$

To solve the equation, we divide [a, b] into N subintervals with (N+1) equally spaced nodes $a = x_0$, x_1, ..., $x_n = b$. Then the problem becomes to determine y_1, y_2, ..., y_{n-1} from the given boundary values y_0=A and y_n=B. To find the numerical solution of the (n-1) unknowns, we require equation (10.11) to be satisfied at all nodes where the function value is to be determined, i.e.

$$y''_i + \rho(x_i)y'_i + g(x_i)y_i = r(x_i), \quad i = 1,2,\dots,n-1. \tag{10.12}$$

Using then central finite difference scheme

$$y'(x_i) = \frac{y_{i+1} - y_{i-1}}{2h} + O(h^2), \qquad y''(x_i) = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + O(h^2). \tag{10.13}$$

Equation (10.12) becomes

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + p_i \frac{y_{i+1} - y_{i-1}}{2h} + q_i y_i = r_i, \tag{10.14}$$

or

$$c_i y_{i+1} + d_i y_i + a_i y_{i-1} = b_i, \quad (i = 1, 2, \dots, n-1), \tag{10.15}$$

where
$$c_i = 1 + \frac{1}{2}hp_i, \;\; d_i = -2 + h^2 q_i, \;\; a_i = 1 - \frac{1}{2}hp_i, \;\; b_i = h^2 r_i. \tag{10.16}$$

In matrix form, equations (10.15) can be written as

$$\begin{bmatrix} d_1 & c_1 & & & O \\ a_2 & d_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-2} & d_{n-2} & c_{n-2} \\ O & & & a_{n-1} & d_{n-1} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-2} \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} b_1 - a_1 A \\ b_2 \\ \vdots \\ b_{n-2} \\ b_{n-1} - c_{n-1}B \end{bmatrix} \quad \text{i.e} \quad A\mathbf{y} = b \tag{10.17}$$

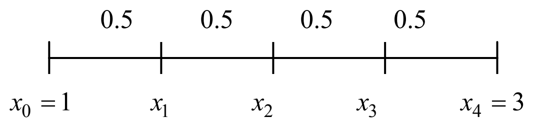
Algorithm:

1) Construct the tri-diagonal system
2) Solve the tri-diagonal system

Example 10.2 Solve
$$\begin{cases} y'' = y & x \in (1,3) \\ y(1) = 1.1752, & y(3) = 10.0179 \end{cases} \tag{10.18}$$
using the Finite Difference method with h=0.5.

Solution

As h=0.5, the solution domain (1,3) is divided into 4 subregions as shown.



Now the problem is: Given y_0 and y_4, find y_1, y_2 and y_3 respectively representing the value of $y(x)$ at x_1, x_2 and x_3.

For this purpose, we require equation (10.18) to be satisfied at the node x_i, i=1, 2, 3:

$$y''_i = y_i, \quad (i = 1,2,3) \tag{10.19}$$

Approximating y_i'' by $y_i'' = \dfrac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$, we obtain $\dfrac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = y_i$

that is $\quad y_{i+1} - (2+h^2)y_i + y_{i-1} = 0 \quad (i=1, 2, 3)$

or in matrix form

$$\begin{bmatrix} -(2+h^2) & 1 & 0 \\ 1 & -(2+h^2) & 1 \\ 0 & 1 & -(2+h^2) \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} -y_0 \\ 0 \\ -y_4 \end{bmatrix}. \tag{10.20}$$

Now, $h=0.5$, $y_0 = 1.1752$, $y_4 = 10.0179$, we thus have

$$\begin{bmatrix} -2.25 & 1 & 0 \\ 1 & -2.25 & 1 \\ 0 & 1 & -2.25 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} -1.1752 \\ 0 \\ -10.0179 \end{bmatrix}$$

Solving the above system yeilds,

$$y_1 = 2.1467,$$
$$y_2 = 3.6549,$$
$$y_3 = 6.0768.$$

Note: It is common to normalise a function to interval [0,1] by substitution $x = (b - a)\,t + a.$

Error Analysis - Richardson's Extrapolation

$$y_h(x) = y(x) + Ah^2 + Bh^4 + \tag{10.21}$$

$$y_{h/2}(x) = y(x) + A\frac{h^2}{4} + B\frac{h^4}{16} + \ldots \tag{10.22}$$

4*(10.22) – (10.21):

$$y(x) = \frac{1}{3}\left[4y_{h/2}(x) - y_h(x)\right] + O(h^4) \tag{10.23}$$

By neglecting the higher order terms of h, we obtain the Richardson's extrapolation formula

$$y_r(x) = \frac{1}{3}[4y_{h/2}(x) - y_h(x)].$$

Remark 1. The errors in $y_h(x)$ and $y_{h/2}(x)$ are of order two of h, while the error in Richardson's extrapolation formula is of order 4 of h.

Remark2. In practice, it is convenient to calculate and represent the improved results by Richardson's extrapolation formula using table form as shown below.

x_i	y_{hi}	$y_{h/2i}$	improved y_{ri}
		$\overline{y}_1$	
$a+h$	y_1	$\overline{y}_2$	$(4\overline{y}_2 - y_1)/3$
		$\overline{y}_3$	
$a+2h$	y_2	$\overline{y}_4$	$(4\overline{y}_4 - y_2)/3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$a+ih$	y_i	$\overline{y}_{2i}$	$(4\overline{y}_{2i} - y_i)/3$

Algorithm

(1) Generate $\{y_1, y_2, ..., y_n\}$ using step size h;

(2) Generate $\{\overline{y}_1, \overline{y}_2, ..., \overline{y}_{2n}\}$ using step size $2h$;

(3) Calculate $y_{ri} \approx y(a+ih) = \dfrac{4\overline{y}_{2i} - y_i}{3}$.

10.3 Collocation Methods

In collocation methods, a linear combination of a set of linearly independent functions is used to approximate the solution function via the following steps:

(i) Express the solution in terms of a set of 'basis' functions $\{\varphi_i(x),\ i = 1,\ 2,\ ......,\ \mathrm{N}\}$, i.e., seak a solution of the form $y_N(x) = \sum_{i=1}^{N} c_i\varphi_i(x)$

(ii) Determine c_i by making $y_N(x)$ to satisfy all the boundary conditions and the differential equation at some values of x.

Remark: The solution determined will satisfy all boundary conditions and the differential equation at some points in $[a, b]$. To obtain more accurate result, simply increase the number of basis functions.

Example 10.3 Solve

$$\begin{cases} y''(x) = y(x) & (10.24) \\ y(0) = 0, \quad y(1) = 1 & (10.25) \end{cases}$$

Solution Firstly we approximate the solution by the following form

$$y_N(x) = c_1 + c_2 x + c_3 x^2 + c_4 x^3. \qquad (10.26)$$

As there are four coefficients to determine, we need to construct 4 equations, which can be obtained by making $y_N(x)$ to satisfy the two boundary conditions and the differential equation at two chosen points in [0,1].

For $y_N(x)$ to satisfy the boundary conditions (10.25), we require

$$c_1 = 0 \tag{10.27}$$

$$c_2 + c_3 + c_4 = 1. \tag{10.28}$$

For $y_N(x)$ to satisfy the differential equation, we substitute (10.26) into (10.24) to yield

$$2c_3 + 6c_4 x = c_1 + c_2 x + c_3 x^2 + c_4 x^3$$

or
$$c_1 + c_2 x + c_3(x^2 - 2) + c_4(x^3 - 6x) = 0\,.$$

Then we choose two points in [0,1], namely ¼ and ¾ , such that the above differential equation is satisfied at this points. Thus,

$$c_1 + c_2\frac{1}{4} + c_3\left(\frac{1}{16} - 2\right) + c_4\left(\frac{1}{64} - \frac{6}{4}\right) = 0 \tag{10.29}$$

$$c_1 + c_2\frac{3}{4} + c_3\left(\frac{9}{16} - 2\right) + c_4\left(\frac{27}{64} - \frac{18}{4}\right) = 0\,. \tag{10.30}$$

Solving the system of equations (10.27)-(10.30) yields

$$\begin{aligned} c_1 &= 0,\\ c_2 &= 0.852237,\\ c_3 &= -0.0138527,\\ c_4 &= 0.161616. \end{aligned}$$

Therefore, $y = y_N = 0.852237x - 0.0138527x^2 + 0.161616x^3$.

To obtain more accuracy, simply increase the number of basis functions.

10.4 Solution of Boundary Value Problems for ODEs using Maple/MATLAB

Solution of Boundary Value Problem for ODEs using Maple

Maple uses the function “**dsolve**()” to find the numerical solution of boundary value problems for ordinary differential equations. The syntax is

```
dsolve(odesys, numeric, vars, options)
```

where `odesys` : set of ordinary differential equations, and initial/boundary conditions
`numeric` : name, instruct dsolve to find a numrical solution
`vars` : (optional) the unknowns of the ODE system
`options`: (optional) `method = bvp(Submethod)` indicates that a specific method is to be used to solve BVP. Available submethods are a combination of the base

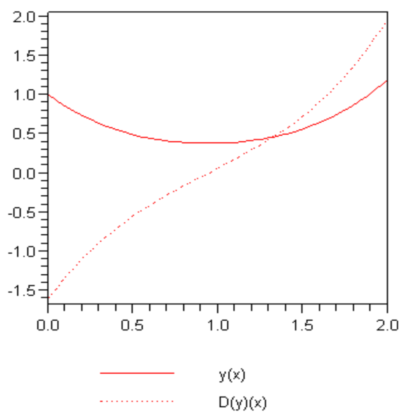
scheme (trapezoid or midpoint), and a method enhancement scheme (Richardson extrapolation or deferred correction). These are specified as traprich, trapdefer,midrich, or middefer. The default method is traprich.

Example 1 Solve $\begin{cases} \dfrac{d^2y}{dx^2} = 3y(x) \\ y(0)=1, \quad y(2)=1.2 \end{cases}$

```
>dsol:=dsolve([diff(y(x),x,x)=3*y(x),y(0)=1, y(2)=1.2],
        numeric);
>dsol(0.5);
>soly:=x->rhs(dsol(x)[2]);
>solDy:=x->rhs(dsol(x)[3]);
>with(plots): setoptions(axes=BOXED);
>p1:= plot(soly, 0..2, linestyle = SOLID, legend= "y(x)");
>p2:= plot(solDy, 0..2, linestyle = DOT, legend=
"D(y)(x)");
>plots[display]({p1,p2});
```

gives

$$\left[x=0.5, y(x)=0.492273627127113700, \frac{d}{dx}y(x) = -.551072761778112596\right]$$



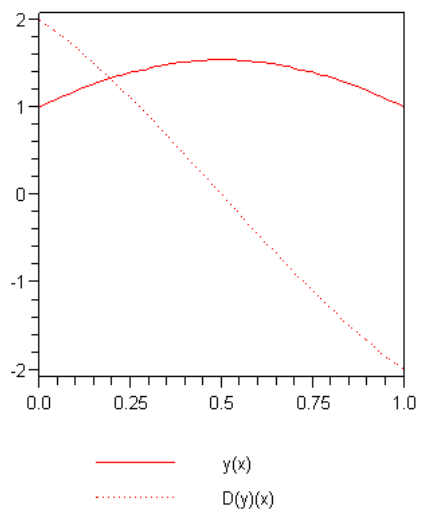
Notes: `soly:=x->rhs(dsol(x)[2])` is to extract the 2nd column of `dsol` and assign it to `soly` as a function of `x`.

Example 2 Solve $\begin{cases} \dfrac{d^2}{dx^2}y(x) - ay(x) = 0, \\ y(0)=1, y(1)=1, y'(0)=2 \end{cases}$

```
> dsys:={diff(y(x), x, x)-a*y(x)=0, y(0)=1, y(1)=1,
         D(y)(0)=2};
> dsol:=dsolve(dsys, numeric, method=bvp[midrich]);
> dsol(1);
> soly:=x->rhs(dsol(x)[2]);
> solDy:=x->rhs(dsol(x)[3]);
> with(plots):
setoptions(axes=BOXED);
> p1:= plot(soly, 0..2, linestyle=SOLID, legend= "y(x)");
> p2:= plot(solDy, 0..2, linestyle=DOT, legend= "D(y)(x)");
> plots[display]({p1,p2});
```

gives

$$\left[\text{x=1., y(x)=1.}, \frac{d}{dx}y(x) = -2., \quad a = -2.96069553617247560\right]$$



Solution of Boundary Value Problem of ODEs using MATLAB

The function "**bvp4c**()" can be used to solve boundary value problems (BVPs) for ordinary differential equations and produces a solution that is continuous on [a,b] and has a continuous first order derivative. The syntax is

```
sol = bvp4c(odefun,bcfun,solinit)
sol = bvp4c(odefun,bcfun,solinit,options)
```

where odefun : A function handle that evaluates the differential equations . It can have the form

```
dydx = odefun(x,y)
dydx = odefun(x,y,parameters)
```

where `x` is a scalar corresponding to *x*, and `y` is a column vector corresponding to *y*. `parameters` is a vector of unknown parameters. The output `dydx` is a column vector.

bcfun : A function handle that computes the residual in the boundary conditions. For two-point boundary value conditions of the form $bc(y(a), y(b))$, "**bcfun**()" can have the form

```
res = bcfun(ya,yb)
res = bcfun(ya,yb,parameters)
```

where `ya` and `yb` are column vectors corresponding to $y(a)$ and $y(b)$. `parameters` is a vector of unknown parameters. The output `res` is a column vector.

`solinit` : A structure containing the initial guess for a solution. You create solinit using the function bvpinit. `solinit` has the following fields.

`x` - Ordered nodes of the initial mesh. Boundary conditions are imposed at a = `solinit.x(1)` and b = `solinit.x(end)`.

y - Initial guess for the solution such that `solinit.y(:,i)` is a guess for the solution at the node `solinit.x(i)`.

`parameters` : (Optional) A vector that provides an initial guess for unknown parameters.

The structure can have any name, but the fields must be named `x`, `y`, and parameters. You can form solinit with the helper function `bvpinit()`.

Example Solve $\begin{cases} y''+|y|=0 \\ y(0)=0, y(4)=-2. \end{cases}$

Prior to solving this problem with **bvp4c**(), you must write the differential equation as a system of two first order ODEs

$$y'_1 = y_2,$$
$$y'_2 = -|y_1|,$$

where $y_1 = y$ and $y_2 = y'$. This system has the required form

$$\begin{cases} y' = f(x,y) \\ bc(y(a), y(b)) = 0 \end{cases}$$

The function f and the boundary conditions are coded in MATLAB as functions `twoode()` and `twobc()`. We now create two M-files as follows.

```
function dydx = twoode(x,y)
dydx = [ y(2)
           -abs(y(1))];
```

```
function res = twobc(ya,yb)
  res = [ ya(1)
          yb(1) + 2];
```

Form a guess structure consisting of an initial mesh of five equally spaced points in [0,4] and a guess of constant values $y_1(x) = 1$ and $y_2(x) = 0$ with the command

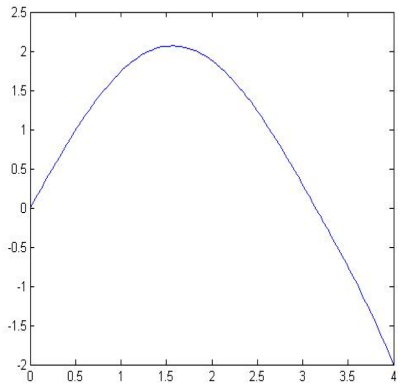
```
>> solinit = bvpinit(linspace(0,4,5),[1 0]);
```

Now solve the problem with

```
>> sol = bvp4c(@twoode,@twobc,solinit);
```

Evaluate the numerical solution at 100 equally spaced points and plot with

```
>> x = linspace(0,4);
>> y = deval(sol,x);
>> plot(x,y(1,:));
```



EXERCISES 10

Q10.1 Solve the boundary value problem

$$\begin{cases} y'' = y \\ y(0) = 0, \quad y(1) = 1 \end{cases}$$

Using the shooting method. Starts with $a_1 = 0.3$, $a_2 = 0.4$ and $h = 0.5$. Use Euler's method for solving systems of differential equations Perform several (say 4) iterations.

Q10.2 Consider the boundary value problem

$$y'' + p(x)\, y' + q(x) y = r(x)\,, \qquad a < x < b$$

with $y(a) = A$ and $y(b) = B$

(a) Use Taylor series to show that the finite differences

$$y''(x_n) = \frac{y_{n+1} - 2y_n + y_{n-1}}{h^2} \quad \text{and} \quad y'(x_n) = \frac{y_{n+1} - y_{n-1}}{2h}$$

have errors of the form $A\,h^2 + B\,h^4 + C\,h^6 + \ldots$

(b) Use the finite differences in (a) to set up the necessary eqs (in matrix form $A\mathbf{y} = \mathbf{b}$) for the numerical solution of the boundary value problem using N iterations.

(c) Solve the boundary value problem

$$y'' + x\,y' - 2y + x = 0, \qquad y(0) = 1, \qquad y(1) = 3$$

using the finite difference method with $h = 0.25$. Compare your results with analytical solution $y = 1 + x + x^2$.

(d) Explain (without actually doing) how you go about improving the accuracy of the solution obtained in (b).

(Ans: (c) 1.3125, 1.7500, 2.3125)

PROGRAMMING

Q10.3 Write a F95 program to solve a linear two-point boundary-value problem using the shooting method. Hence solve the problem

$$y'' + x\,y' - 2y + x = 0, \quad y(0) = 1, \; y(1) = 3 \quad \text{with } N = 4.$$

<u>Hint:</u>

(1) Rewrite the B.V.P in the form of $\begin{cases} y' = z \\ z' = f(x, y, z) \\ y(a) = AA, \quad y(b) = BB. \end{cases}$

(2) Define $f(x, y, z)$ using a function subprogram.

(3) Write a subroutine for solving the I.V.P $\begin{cases} y' = z \\ z' = f(x, y, z) \\ y(a) = Y_0, \quad z(a) = Z_0 \end{cases}$

using Euler's method to obtain B_k (estimate of $y(b)$).

SUBROUTINE EulerTwo (a, b, y0, z0, N, Bk)
COMMON Xi(100), Yi(100), Zi(100)

a, b, y0, z0	*= Input.*
N	*= Input. Number of intervals*
Bk	*= Output. The estimate of $y(b)$.*

(4) Write a subroutine to implement the shooting method

SUBROUTINE Shooting (a, b, AA, BB, N, alf1, alf2, Tol, MaxNit, Ierr)

a, b	=	*Interval of x, i.e, $[a, b]$.*
AA, BB	=	*As defined in (1).*
alf1, alf2	=	*Initial guesses of z(a)*
Tol	=	*Tolerance. If $\lvert B_{k+1} - B_k \rvert < Tol$* **or** *$\lvert B_k - B \rvert < Tol$, convergence.*
MaxNit	=	*Maximum number of iterations. If the* ***number of iterations >MaxNit,*** *Let* ***Ierr = 1*** *and then return.*

Ierr = *Output. Return '0' if the iteration converges.*

Algorithm:

```
h = (b – a) / N
alf(1) = alf1
alf(2) = alf2
k = 0
Do while (k < MaxNit)
    k = k +1
    Call  EulerTwo (a, b, AA,  alf(k), N, B(k) )
    If  k ≥ 2  Then
        if  $(|B(k) - B(k-1)| < Tol$  or  $|B(k) - BB| < Tol)$  then
            Ierr = 0
            return
        else
            alf(k+1) = alf(k)+...
        endif
    Endif
EndDo
Ierr =1
Return
```

(5) Write a main program which reads input data, calls the subroutine **Shooting** and prints the numerical result (or error message).

Algorithm:

```
Input  a, b, .....
Call  Shooting ( ... )
    If ( Ierr = 0) Then
        Print  {xi, yi, zi}
    Else
        Print ('Not converges')
    End
```

Q10.4 Write a F95 program to solve a linear two-point boundary-value problem using the finite difference method. Hence solve the problem defined in Q10.2(c) with step size h=0.25.

Hint:

(1) Rewrite the differential equation given in the form of $y'' + p(x)\,y' + q(x)y = r(x)$ and then define p, q, r using function subprograms.

(2) Write a subroutine FTriD to construct the tri-diagonal system (see formula 10.16 and 10.17)

$$\begin{bmatrix} d_1 & c_1 & & \\ a_2 & d_2 & \ddots & \\ & \ddots & \ddots & c_{n-2} \\ & & a_{n-1} & d_{n-1} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{bmatrix}$$

SUBROUTINE FTriD (a, b, AA, BB, N, Ai, Di, Ci, Bi)
Input: *a*, *b*, *AA*, *BB*, *as defined in the B.V.P.*
Output: *Ai*, *Bi*, *Ci*, *Di*, *arrays as defined in above eqs*.

Algorithm:

```
h=(b – a) / N
for i = 1 to N – 1
    Set Ai (i) = 1– 1/2 * h * pi
        Di (i) = – 2 + h**2*qi
        Ci (i) = 1 + 1/2 *h*pi
        Bi (i) = h**2*ri
set Bi (1) = Bi (1) – Ai (1) * AA
    Bi (n – 1) = Bi (n – 1) – Ci (n – 1) * BB
```

(3) Write a subroutine for solving the tri-diagonal system
SUBROUTINE SolTriDS (N, Ai, Di, Ci, Bi)
COMMON Y(100)
INPUT : *N*, *Ai*, *Di*, *Ci*, *Bi*, *as defined in above eqs.*
OUTPUT : *Y*, *solution of the system*

(4) Write a main program which reads input data, calls FTriD to form the tri-diagonal system, calls SolTriDS to solve the tri-diagonal system, and prints the results.

Algorithm:

```
Input a, b, ...
Call FTriD ( ... )
Call SolTriDS ( ... )
Print solution {xi, yi}
End
```

Q10.5. Modify your program in Q10.4 to solve the boundary value problem in Q10.2(c) with h = 0.25 and h = 0.125.

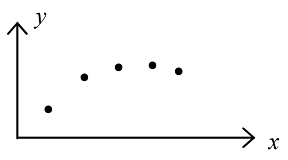
Q10.6 Solve the boundary value problems in Q 10.3 using Maple and Matlab built-in functions.

CHAPTER 11

Least Squares Approximation and Curve Fitting

The chapter concerns two types of approximation: curve fitting and function approximation. In curve fitting, we are given a discrete set of data points $\{(x_i, y_i)\}_{i=1}^{m}$ describing the relation between x and y which may arise from experimental investigation and is usually given in table form or as a set of data points in a x-y diagram such as those shown below

x_i	0	1	2	3	4
y_i	0	2.1	3.9	6.1	7.9



The purpose of curve fitting is to find a function (usually a polynomial) to approximately describe the relation between x and y, i.e $y = y(x)$, or in other words, find a curve which is as close as possible to the data points.

In function approximation, we are given a continuous function and our aim is to find the best polynomial to approximate the function. The need to approximate a continuous function by a polynomial arises from many cases, for example, when the function is an integrand which is not integrable analytically such as evaluating

$$\int \sqrt{1+x^3}\,dx$$

In summary, no matter in curve fitting or function approximation, our aim is to find the best function (usually a polynomial function) to fit the given data or function. To achieve this objective, we need to answer/tackle the following two essential problems.

What do we mean by the best?

The answer to this question is not so straight forward. One would expect that the best curve must be the one which is closest to the data points. However, the problem is how to measure the ' distance' between a curve and data points. To answer this problem, we need to define an error measure.

Denote the error at each point by

$$e_i = y(x_i) - y_i \qquad \text{(error = estimate –actual data)}$$

where $y(x_i)$ and y_i denote respectively the value of the approximation function and the actual value of y corresponding to x_i, as shown in figure 11. 2.

Now we consider the following quantities to determine which can be chosen as error measures :

i) $E_1 = e_1 + e_2 + \ldots + e_n$

ii) $E_2 = \|e\|_\infty = \max\limits_{1 \le i \le n} |e_i|$

iii) $E_3 = \|e\|_2 = \sum\limits_{i=1}^{n} e_i^2$

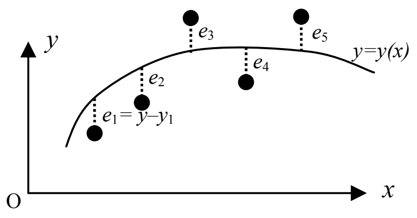


Figure 11.2 Diagram showing local errors

Obviously, both (ii) and (iii) can be used as an error measure as the value of E_2 and E_3 reflect the accuracy of approximation. However, (i) cannot be used as an error measure, as the value of $E_1 = \sum\limits_{i=1}^{n} e_i$ does not reflect the accuracy of approximation. For example, even when local errors at all the measure points are very large, $\sum\limits_{i} e_i$ may still has zero value if some of the local errors are positive and some are negative.

How to Find the Best Approximation ?

First, we need to determine the form of the approximate function with certain arbitrary constants. Then we determine the arbitrary constants which minimise the error in some sense such as Minimax or least squares.

Remark: Different error measures lead to different approximation methods. The following is two widely used approximation methods.

Minimax Approximation

In this method, the distance from $P_n(x)$ to $y(x)$ is measured via the l_∞ norm

$$E = \|y - P_n\|_\infty = \max_{a \le x \le b} |y(x) - P_n(x)|.$$

Let π_n be the set of polynomials of degree $\le n$, then the Best Approximation P_n^* would satisfy

$$\|y - P_n^*\|_\infty = \min_{P \in \pi_n} \|y - P\|$$

Least Squares Approximation

In this method, the distance from $P_n(x)$ to $y(x)$ is measured via the squares sum of errors

$$E = \int_a^b [y(x) - P_n(x)]^2 dx \quad \text{– for continuous function}$$

$$E = \sum_{i=1}^{m} [y_i - P_n(x_i)]^2 \quad \text{– for discrete data.}$$

Let P_n be the set of polynomials of degree $\le n$, then the best approximation P_n^* would be the one that minimises the above E.

11.1 Least Squares Fit

Problem: Given a set of data $\{(x_i, y_i)\}_{i=1}^m$, find a best polynomial $P_n(x) = \sum_{j=0}^n a_j x^j$ to fit the data in the least squares sense.

Error Measure: In least square fit, the distance from $P_n(x)$ to $\{(x_i, y_i)\}_{i=1}^m$ is measured via the squares sum of errors at the data points :

$$E = \sum_{i=1}^m [y_i - P_n(x_i)]^2 = \sum_{i=1}^m \left[y_i - \sum_{j=0}^n a_j x_i^j \right]^2 .$$

Derivation of Best Approximation:

Obviously, the squares sum of errors depends only on the coefficients a_j above. To find the best $P_n(x)$ is to choose a_j $(j = 0, 1, \ldots n)$ to minimise the error measure E defined above.

As $E = E(\alpha_0, \alpha_1, \ldots, \alpha_n)$, for a minimum of E, we must have

$$\frac{\partial E(a_0, a_1, \ldots, a_n)}{\partial a_k} = 0, \quad (k = 0, 1, \ldots, n),$$

$$\Rightarrow \quad 2\sum_{i=1}^m \left[y_i - \sum_{j=0}^n a_j x_i^j \right] \left(-x_i^k \right) = 0$$

$$\Rightarrow \quad \sum_{i=1}^m \sum_{j=0}^n a_j x_i^j x_i^k = \sum_{i=1}^m y_i x_i^k$$

$$\Rightarrow \quad \sum_{j=0}^n a_j \left(\sum_{i=1}^m x_i^{k+1} \right) = \sum_{i=1}^m y_i x_i^k \quad (k = 0, 1, \ldots\ldots, n)$$

which lead to the following normal equations

Normal Equations for Least Squares Polynomial Fit

$$C\mathbf{a} = \mathbf{b}$$

where $$c_{kj} = \sum_{i=1}^m x_i^{k+j}, \quad b_k = \sum_{i=1}^m y_i x_i^k . \quad (k, j = 0, 1, \ldots, n).$$

Let $s_l = \sum_{i=1}^m x_i^l$, then $c_{kj} = s_{k+j}$ and thus

$$C = \begin{bmatrix} c_{00} & c_{01} & \cdots & c_{0n} \\ c_{10} & c_{11} & \cdots & c_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n0} & c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} s_0 & s_1 & \cdots & s_n \\ s_1 & s_2 & \cdots & s_{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_n & s_{n+1} & \cdots & s_{2n} \end{bmatrix}.$$

Algorithm:

1) Construct C and **b**,

2) Solve C**a** = **b** for **a**, then $y(x) \approx P_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$.

Example 11.1 Fit a quadratic to the data (0, 0), (1, 4), (-1, 1), (-2,5).

Solution The problem is to fit $y_2(x) = a_0 + a_1 x + a_2 x^2$ to $\{x_i, y_i\}_{i=1}^{4}$.

For $m = 4$, $n = 2$, we have

$$s_\ell = \sum_{i=1}^{m} x_i^\ell \quad (\ \ell = 0, 1, 2, \ldots, 2n) \quad \text{giving}$$

$$s_0 = \sum_{i=1}^{4} i = 4$$

$$s_1 = \sum_{i=1}^{4} x_i = 0 + 1 - 1 - 2 = -2$$

$$s_2 = \sum_{i=1}^{4} x_i^2 = 0 + 1^2 + (-1)^2 + (-2)^2 = -6$$

$$s_3 = \sum_{i=1}^{4} x_i^3 = 0 + 1^3 + (-1)^3 + (-2)^3 = -8$$

$$s_4 = \sum_{i=1}^{4} x_i^4 = 0 + 1^4 + (-1)^4 + (-2)^4 = 18,$$

and $b_k = \sum_{i=1}^{m} y_i x_i^k \quad (k = 0,1,2)$ giving

$$b_0 = \sum_{i=1}^{4} y_i = 0 + 4 + 1 + 5 = 10$$

$$b_1 = \sum_{i=1}^{4} y_i x_i = 0 + 1 \times 4 + (-1) \times 1 + (-2) \times 5 = -7$$

$$b_2 = \sum_{i=1}^{m} y_i x_i^2 = 25.$$

$$\therefore \quad \begin{bmatrix} s_0 & s_1 & s_2 \\ s_1 & s_2 & s_3 \\ s_2 & s_3 & s_4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 4 & -2 & 6 \\ -2 & 6 & -8 \\ 6 & -8 & 18 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 10 \\ -7 \\ 25 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 1.6 \\ 2 \end{bmatrix}$$

Hence,

$$y_2(x) = 0.3 + 1.6x + 2x^2.$$

11.2 Weighted Least Squares Fit

For experimental data, usually we associate a weight w_i with each point (x_i, y_i). The purpose of weighting is to assign varying degree of importance to approximations on certain portions of the interval. For the portion which is more important, a larger weight must be assigned to it.

Error Measure: The distance between the approximation function $P_n(x)$ and the data y_i is measured by

$$E = \sum_{i=1}^{m} w_i \left[y_i - P_n(x_i)\right]^2 = \sum_{i=1}^{m} w_i \left[y_i - \sum_{j=0}^{n} a_j x_i^j\right]^2$$

Derivation of Best Approximation

Following the same procedure as in section 11.1, we obtain the following normal equations for the weighted least squares fit

Normal Equations for Weighted Least Squares Polynomial Fit

$$C'\mathbf{a} = \mathbf{b}'$$

where $$c'_{kj} = \sum_{i=1}^{m} w_i x_i^{k+j}, \quad b'_k = \sum_{i=1}^{m} w_i y_i x_i^k .$$

Let $s'_l = \sum_{i=1}^{m} w_i x_i^l$,

then $c'_{kj} = s'_{k+j}$ and thus

$$C' = \begin{bmatrix} c'_{00} & c'_{01} & \cdots & c'_{0n} \\ c'_{10} & c'_{11} & \cdots & c'_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ c'_{n0} & c'_{n1} & \cdots & c'_{nn} \end{bmatrix} = \begin{bmatrix} s'_0 & s'_1 & \cdots & s'_n \\ s'_1 & s'_2 & \cdots & s'_{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ s'_n & s'_{n+1} & \cdots & s'_{2n} \end{bmatrix}$$

Example 11.2 Given

x_i	– 2	–1	0	1	2
y_i	0.3	0.5	1	2	4.1
w_i	0.2	0.5	1	0.5	0.2

Fit a quadratic to the data.

Solution For $m = 5$, $n = 2$, we have

$$s'_\ell = \sum_{i=1}^{m} w_i x_i^\ell \quad (\ell = 0, 1, 2, \ldots, 2n)$$

$$s_0' = \sum_{i=1}^{5} w_i x_i^0 = \sum_{i=1}^{5} w_i = 2.4$$

$$s_1' = \sum_{i=1}^{5} w_i x_i = 0.2\times(-2) + 0.5\times(-1) + 0 + 0.5\times(1) + 0.2\times(2) = 0$$

$$s_2' = \sum_{i=1}^{5} w_i x_I^2 = 2.6$$

$$s_3' = \sum_{i=1}^{5} w_i x_I^3 = 0$$

$$s_4' = \sum_{i=1}^{5} w_i x_i^4 = 7.4$$

$$b_k' = \sum_{i=1}^{m} w_i y_i x_i^k \quad (k = 0,1,\ldots,n)$$

$$b_0' = \sum_{i=1}^{5} w_i y_i x_i^0 = \sum_{i=1}^{5} w_i y_i = 0.2(0.3) + 0.5(0.5) + 1(1) + 0.5(2) + 0.2(4.1) = 3.13$$

$$b_1' = \sum_{i=1}^{5} w_i y_i x_i = 2.27, \quad b_2' = \sum_{i=1}^{5} w_i y_i x_i^2 = 4.77$$

$$\therefore \quad \begin{bmatrix} s_0' & s_1' & s_2' \\ s_1' & s_2' & s_3' \\ s_2' & s_3' & s_4' \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 2.4 & 0 & 2.6 \\ 0 & 2.6 & 0 \\ 2.6 & 0 & 7.4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 3.13 \\ 2.27 \\ 4.77 \end{bmatrix}$$

$$\Rightarrow \quad a_0 = 0.9782, \quad a_1 = 0.8710, \quad a_2 = 0.3009$$

$$\therefore \quad y_2(x) = 0.9782 + 0.8710x + 0.3009x^2 .$$

Remark (problems with least squares methods using normal equations) It can be shown that with the increase in n (degree of polynomial), the least squares approximation obtained by solving the normal equations becomes progressively worse as the normal equations becomes ill-conditioned.

11.3 Orthogonal Polynomials

Definition (Inner Product): Let $f(x)$, $g(x)$ be functions defined on $[a, b]$, and $w(x)$ be a weight function defined on (a, b). We define the inner product of f with g by

$$\langle f, g\rangle = \int_a^b w(x) f(x) g(x) dx.$$

If f and g are discrete (.i.e., defined on a discrete set of data points $\{x_i\}_{i=1}^m$), then

$$\langle f, g\rangle = \sum_{i=1}^{m} w(x_i) f(x_i) g(x_i) .$$

Definition (Orthogonal and Orthonormal): A family of polynomials $\{P_n(x) | n \geq 0\}$ is said to be:
Orthogonal for the interval $[a, b]$ with respect to the weight function $w(x)$ if

$$\langle P_r, P_s \rangle = 0 \quad \text{whenever} \quad r \neq s.$$

Orthonormal if, in addition, $\langle P_r, P_r \rangle = 1 \ \forall r$.

Example 11.3 Show $P_0(x) = 1$, $P_1(x) = x$ and $P_2(x) = 3x^2 - 1$ are orthogonal but not orthonormal on $[-1, 1]$.
Solution (exercise)

Properties of Orthogonal Polynomials

(1) Every polynomial can be written as a combination of orthogonal polynomials of no greater degree.

Let $\{P_n(x) | n \geq 0\}$ be an orthogonal family of polynomial on $[a, b]$ with weight function $w(x)$ and let $f(x)$ be a polynomial of degree n, then

$$f(x) = \sum_{r=0}^{n} a_r P_r(x), \qquad \text{with} \qquad a_r = \frac{\langle f, P_r \rangle}{\langle P_r, P_r \rangle}.$$

Proof To find a_s, take inner product of f with P_s:

$$\begin{aligned} \langle f, P_s \rangle &= \sum_{r=0}^{n} a_r \langle P_r, P_s \rangle \\ &= a_0 \langle P_0, P_s \rangle + a_1 \langle P_1, P_s \rangle + \ldots + a_s \langle P_s, P_s \rangle + \ldots + a_n \langle P_n, P_s \rangle \\ &= a_s \langle P_s, P_s \rangle \end{aligned}$$

$$\therefore \quad a_s = \frac{\langle f, P_s \rangle}{\langle P_s, P_s \rangle}.$$

(2) Let $\{P_n(x) | n \geq 0\}$ be a set of orthogonal polynomials. If $P(x)$ is a polynomial of degree $< k$, then $P(x)$ is orthogonal to $P_k(x)$.

Proof

Let $P(x)$ be a polynomial of degree $m < k$, and
$\{P_0, P_1, \ldots, P_m, \ldots, P_k\}$ be a set of orthogonal polynomials.

Then from property (1)

$$P(x) = \sum_{s=0}^{m} a_s P_s(x).$$

Hence

$$\begin{aligned} \langle P, P_k \rangle &= \sum_{s=0}^{m} a_s \langle P_s, P_k \rangle \\ &= 0 \qquad \text{(as all } s \neq k\text{)}. \end{aligned}$$

(3) (Recurrence formulae for generation of orthogonal polynomials) If we set the coefficient of x_r in $P_r(x)$ to be 1, and define $P_{-1}= 0$, $P_0= 1$, then we can generate a unique family of orthogonal polynomials using the following formulae:

$$P_{r+1} = (x - \alpha_r)P_r - \beta_r P_{r-1},$$

where

$$\begin{cases} \alpha_r = \dfrac{\langle xP_r, P_r\rangle}{\langle P_r, P_r\rangle}, & r = 0,\ 1,\ 2,\ \ldots \\ \beta_r = \dfrac{\langle P_r, P_r\rangle}{\langle P_{r-1}, P_{r-1}\rangle}, & r = 1,\ 2,\ 3,\ \ldots \end{cases} \qquad (11.1)$$

Proof We prove this by mathematical induction. Firstly, we prove that the property is true for P_1 ($r = 0$). Then we assume that it is true for P_n ($r = n$-1) and prove that it is also true for P_{n+1} ($r = n$).

(1) Check the property is true for P_1 ($r = 0$).

From the recurrence formulae, for $r = 0$,

$$\alpha_0 = \frac{\langle xP_0, P_0\rangle}{\langle P_0, P_0\rangle},$$

$$P_1 = (x - a_0)P_0.$$

Hence

$$\begin{aligned} \langle P_0, P_1\rangle &= \langle P_0, xP_0 - \alpha_0 P_0\rangle \\ &= \langle P_0, xP_0\rangle - \alpha_0 \langle P_0, P_0\rangle \\ &= \langle P_0, xP_0\rangle - \frac{\langle xP_0, P_0\rangle}{\langle P_0, P_0\rangle}\langle P_0, P_0\rangle \quad = \quad 0 \end{aligned}$$

and so P_1 is orthogonal to P_0.

(2) Suppose that the property is true for P_n ($r = n$-1) , i.e., the P_n constructed from the recurrence formulae is orthogonal to P_0, P_1, …, P_{n-1}. We will show here that the P_{n+1} (r=n) generated from the recurrence formulae will be orthogonal to P_0, P_1, …, P_n, as detailed below.

From the recurrence formulae,

$$\alpha_n = \frac{\langle xP_n, P_n\rangle}{\langle P_n, P_n\rangle}, \quad \beta_n = \frac{\langle P_n, P_n\rangle}{\langle P_{n-1}, P_{n-1}\rangle}, \quad P_{n+1} = (x - \alpha_n)P_n - \beta_n P_{n-1},$$

we have

$$\begin{aligned}\left\langle P_{n+1}, P_j\right\rangle &= \left\langle (x-\alpha_n)P_n - \beta_n P_{n-1}, P_j\right\rangle \\ &= \left\langle xP_n, P_j\right\rangle - \alpha_n\left\langle P_n, P_j\right\rangle - \beta_n\left\langle P_{n-1}, P_j\right\rangle \\ &= \left\langle xP_n, P_j\right\rangle - \frac{\left\langle xP_n, P_n\right\rangle}{\left\langle P_n, P_n\right\rangle}\left\langle P_n, P_j\right\rangle - \frac{\left\langle P_n, P_n\right\rangle}{\left\langle P_{n-1}, P_{n-1}\right\rangle}\left\langle P_{n-1}, P_j\right\rangle. \end{aligned} \qquad (11.2)$$

For *j* < *n*-1,

$$\left\langle xP_n, P_j\right\rangle = \left\langle P_n, xP_j\right\rangle = 0 \quad \text{(as } xP_j \text{ is a polynomial of degree less than } n\text{)}$$

Hence from (11.2), we have

$$\left\langle P_{n+1}, P_j\right\rangle = 0 \qquad \text{for all} \quad j < n\text{-}1.$$

For *j* = *n*-1, from (11.2), we have

$$\left\langle P_{n+1}, P_{n-1}\right\rangle = \left\langle xP_n, P_{n-1}\right\rangle - \frac{\left\langle xP_n, P_n\right\rangle}{\left\langle P_n, P_n\right\rangle}\left\langle P_n, P_{n-1}\right\rangle - \frac{\left\langle P_n, P_n\right\rangle}{\left\langle P_{n-1}, P_{n-1}\right\rangle}\left\langle P_{n-1}, P_{n-1}\right\rangle. \quad (11.3)$$

As from the recurrence formulae

$$P_n = (x-\alpha_{n-1})P_{n-1} - \beta_{n-1}P_{n-2},$$

we have

$$xP_{n-1} = P_n + \alpha_{n-1}P_{n-1} + \beta_{n-1}P_{n-2}$$

and thus

$$\begin{aligned}\left\langle xP_n, P_{n-1}\right\rangle = \left\langle P_n, xP_{n-1}\right\rangle &= \left\langle P_n, P_n\right\rangle + \alpha_{n-1}\left\langle P_n, P_{n-1}\right\rangle + \beta_{n-1}\left\langle P_n, P_{n-2}\right\rangle \\ &= \left\langle P_n, P_n\right\rangle.\end{aligned}$$

Hence from (11.3), we have

$$\left\langle P_{n+1}, P_{n-1}\right\rangle = \left\langle P_n, P_n\right\rangle - 0 - \left\langle P_n, P_n\right\rangle = 0.$$

For *j* = *n*, from (11.2), we have

$$\begin{aligned}\left\langle P_{n+1}, P_n\right\rangle &= \left\langle xP_n, P_n\right\rangle - \frac{\left\langle xP_n, P_n\right\rangle}{\left\langle P_n, P_n\right\rangle}\left\langle P_n, P_n\right\rangle - \frac{\left\langle P_n, P_n\right\rangle}{\left\langle P_{n-1}, P_{n-1}\right\rangle}\left\langle P_{n-1}, P_n\right\rangle \\ &= \left\langle xP_n, P_n\right\rangle - \left\langle xP_n, P_n\right\rangle = 0.\end{aligned}$$

Hence, the P_{n+1} generated for the recurrence formulae is orthogonal to P_0, P_1, …, P_n. The proof is completed. □

Example 11.4 Generate P_0, P_1 and P_2 in [-1,1] for $w = 1$

Solution $P_{-1} = 0,\ P_0 = 1$.

For $r = 0$,

$$P_1 = (x - \alpha_0) \quad \text{where } \alpha_0 = \frac{\langle xP_0, P_0\rangle}{\langle P_0, P_0\rangle} = \frac{\int_{-1}^{1} x dx}{\int_{-1}^{1} 1 dx} = 0$$

$\therefore P_1 = x.$

For $r=1$

$$P_2 = (x - \alpha_1)P_1 \text{-} \beta_1 \quad \text{where } \alpha_1 = \frac{\langle xP_1, P_1\rangle}{\langle P_1, P_1\rangle} = \frac{\int_{-1}^{1} x^3 dx}{\langle P_1, P_1\rangle} = 0,$$

$$\beta_1 = \frac{\langle P_1, P_1\rangle}{\langle P_0, P_0\rangle} = \frac{\int_{-1}^{1} x^2 dx}{\int_{-1}^{1} 1 dx} = \frac{1}{3}.$$

$$\therefore P_2 = x^2 - \frac{1}{3}.$$

11.4 Least Squares Approximation Using Orthogonal Polynomials

11.4.1 Curve Fitting

Problem: Given a set of data $\{(x_i, y_i)\}_{i=1}^{m}$ and a set of orthogonal polynomials $\{P_n(x) | n \geq 0\}$ with inner product

$$\langle P_r, P_s\rangle = \sum_{i=1}^{m} w(x_i) P_r(x_i) P_s(x_i)$$

find a best approximating polynomial $f(x) = \sum_{r=0}^{n} a_r P_r(x)$ to fit the data.

Error Measure

Measure the 'distance' from $f(x)$ to (x_i, y_i) (i=1,m) via the squares sum of errors at the points

$$E = \sum_{i=1}^{m} w_i [y_i - f(x_i)]^2 = \sum_{i=1}^{m} w_i \left[y_i - \sum_{r=0}^{n} a_r P_r(x_i) \right]^2.$$

Derivation of Best Approximation

Find the best $f(x)$ by choosing a_r (r = 0, 1, ..., n) which minimise the E defined above.

Notice that $E = E(\alpha_0, \alpha_1, \ldots, \alpha_n)$. For a minimum of E, we must have

$$\frac{\partial E(a_0, a_1, \ldots, a_n)}{\partial a_k} = 0, \quad (k = 0, 1, \ldots, n),$$

$$\Rightarrow \quad 2\sum_{i=1}^{m} w_i \left(y_i - \sum_{r=0}^{n} a_r P_r(x_i) \right) P_k(x_i) = 0$$

$$\Rightarrow \quad \sum_{i=1}^{m} w_i y_i P_k(x_i) = \sum_{i=1}^{m} w_i P_k(x_i) \sum_{r=0}^{n} a_r P_r(x_i) = \sum_{r=0}^{n} a_r \sum_{i=1}^{m} w_i P_r(x_i) P_k(x_i)$$

$$= \sum_{r=0}^{n} a_r \langle P_k, P_r \rangle = a_k \langle P_k, P_k \rangle \quad \text{as} \quad \langle P_r, P_k \rangle = \begin{cases} \neq 0 & \text{if } r = k \\ 0 & \text{if } r \neq k \end{cases}$$

$$\therefore \quad a_k = \frac{\sum_{i=1}^{m} w_i y_i P_k(x_i)}{\sum_{i=1}^{m} w_i \left[P_k(x_i) \right]^2} = \frac{\langle y, P_k \rangle}{\langle P_k, P_k \rangle}. \tag{11.4}$$

Algorithm:

1) Construct orthogonal polynomials (P_{-1}), P_0, P_1, P_2, and P_3 using property (1) of the orthogonal polynomials;
2) Construct $f(x) = \sum_{r=0}^{n} a_r P_r(x)$ with a_r determined by formula (11.4) above.

Example 11.5 Use orthogonal polynomials to obtain a least squares polynomial of degree 3 to fit the data (w_i=1)

x_i	–2	–1	0	1	2
y_i	–1	–1	0	1	1

Solution

Construct orthogonal polynomials (P_{-1}), P_0, P_1, P_2, and P_3 using the recurrence relation (11.1)

$P_{-1} = 0$, $P_0 = 1$

$P_1 = (x - \alpha_0) P_0 = x$, as $\alpha_0 = \dfrac{\langle xP_0, P_0 \rangle}{\langle P_0, P_0 \rangle} = \dfrac{\sum_{i=1}^{5} x_i}{\sum_{i=1}^{5} 1} = \dfrac{x_1 + x_2 + x_3 + x_4 + x_5}{5} = 0$

$P_2 = (x - \alpha_1) P_1 - \beta_0 P_0$, where $\alpha_1 = \dfrac{\langle xP_1, P_1 \rangle}{\langle P_1, P_1 \rangle} = \dfrac{\sum_{i=1}^{5} x_i P_1(x_i) P_1(x_i)}{\sum_{i=1}^{5} P_1(x_i) P_1(x_i)} = \dfrac{\sum_{i=1}^{5} x_i^3}{\sum_{i=1}^{5} x_i^2} = 0$

$$\beta_0 = \frac{\langle P_1, P_1 \rangle}{\langle P_0, P_0 \rangle} = \frac{\sum_{i=1}^{5} x_i^2}{5} = 2$$

$\therefore \; P_2 = x^2 - 2$

Similarly, $\alpha_2 = 0$, $\beta_2 = \frac{14}{10}$, $P_3 = x^3 - 2x - \dfrac{14}{10} x = x^3 - \dfrac{17}{5} x$.

Hence, the least squares polynomial of degree 3 is

$$f(x) = \sum_{k=o}^{3} a_k P_k(x) \qquad \text{with } a_k = \frac{\langle y, P_k \rangle}{\langle P_k, P_k \rangle} \tag{11.5}$$

Now
$$a_0 = \frac{\langle y, P_0 \rangle}{\langle P_0, P_0 \rangle} = \frac{\sum_{i=1}^{5} y_i P_0(x_i)}{\sum_{i=1}^{5} P_0^2(x_i)} = \frac{\sum_{i=1}^{5} y_i}{\sum_{i=1}^{5} 1} = \frac{0}{5} = 0$$

$$a_1 = \frac{\langle y, P_1 \rangle}{\langle P_1, P_1 \rangle} = \frac{\sum_{i=1}^{5} y_i x_i}{\sum_{i=1}^{5} x_i^2} = \frac{6}{10} = \frac{3}{5}$$

$$a_2 = \frac{\langle y, P_2 \rangle}{\langle P_2, P_2 \rangle} = \frac{\sum_{i=1}^{5} y_i P_2(x_i)}{\sum_{i=1}^{5} [P_2(x_i)]^2} = \frac{\sum_{i=1}^{5} y_i (x_i^2 - 2)}{\sum_{i=1}^{5} (x_i^2 - 2)^2} = 0$$

$$a_3 = \frac{\langle y, P_3 \rangle}{\langle P_3, P_3 \rangle} = \frac{\sum_{i=1}^{5} y_i P_3(x_i)}{\sum_{i=1}^{5} [P_3(x_i)]^2} = \frac{\sum_{i=1}^{5} y_i \left(x_i^3 - \frac{17}{5} x_i\right)}{\sum_{i=1}^{5} \left(x_i^3 - \frac{17}{5} x_i\right)^2} = -\frac{1}{6}$$

$$\therefore \quad f(x) = \frac{3}{5} P_1 - \frac{1}{6} P_3$$
$$= \frac{3}{5} x - \frac{1}{6}\left(x^3 - \frac{17}{5} x\right) = \frac{1}{6}\left(7x - x^3\right).$$

11.4.2 Approximation of Functions

Problem: Given a continuous function *f*(*x*) and a set of orthogonal polynomials $\{P_r | r \geq 0\}$, find a best least squares polynomial $P(x) = \sum_{r=0}^{m} a_r P_r(x)$ to fit the data.

Error Measure: $E(a_0, a_1, ..., a_n) = \int_a^b w(x) \left[f(x) - \sum_{r=0}^{n} a_r P_r(x) \right]^2 dx.$

Derivation of the Best Approximation

Find the best approximation by choosing a_r ($r = 0, 1, ..., n$) which minimise the E defined above.

To minimize E, let $\frac{\partial E(a_0, a_1, ..., a_n)}{\partial a_s} = 0, \quad (s = 0, 1, ..., n),$

$$\Rightarrow \quad 2\int_a^b w(x)\left[f(x)-\sum_{r=0}^{n} a_r P_r(x)\right]P_s(x)dx = 0$$

$$\Rightarrow \quad 2\int_a^b w(x)f(x)P_s(x)dx = \sum_{r=0}^{n} a_r \int_a^b w(x)P_r(x)P_s(x)dx = \sum_{r=0}^{n} a_r \langle P_r, P_s\rangle = a_s\langle P_s, P_s\rangle$$

$$\text{as } \langle P_r, P_s\rangle = \begin{cases} \neq 0 & \text{if } r = s \\ 0 & \text{if } r \neq s \end{cases}$$

Hence, we obtain

$$a_s = \frac{\int_a^b w(x)f(x)P_s(x)dx}{\int_a^b w(x)[P_s(x)]^2\,dx} = \frac{\langle f, P_s\rangle}{\langle P_s, P_s\rangle} \tag{11.6}$$

Algorithm:

1) Construct orthogonal polynomials (P_{-1}), P_0, P_1, P_2, and P_3 using property 1 of orthogonal polynomials;

2) Construct $f(x) = \sum_{r=0}^{n} a_r P_r(x)$ with the a_r determined by formula (11.6) above.

11.5 Least Squares Approximation using Legendre Polynomials

Definition (Legendre Polynomials): The Legendre polynomials $\{P_r(x)\}$ is a set of orthogonal polynomials defined on [– 1, 1] with weight function $w(x)$ =1.

Sequence of Legendre Polynomials

The sequence of Legendre polynomials can be derived by the recurrence relations (property 3 for orthogonal polynomials).

Exercise *Show that the first few terms in the sequence are as follows*

$$P_0 = 1, \quad P_1 = x, \quad P_2 = x^2 - \frac{1}{3}, \quad P_3 = x^3 - \frac{3}{5}x, \quad P_4 = x^4 - \frac{6}{7}x^2 + \frac{3}{35}, \quad \ldots\ldots$$

Normalized Legendre Polynomials

It is customary to normalize the Legendre polynomials so that $P_k(1) = 1$ for all k. Thus, the normalized Legendre polynomials are

$$P_0 = 1, \quad P_1 = x, \quad P_2 = \left(3x^2 - 1\right)/2, \quad P_3 = \left(5x^3 - 3x\right)/2, \quad P_4 = \left(35x^4 - 30x^2 + 3\right)/8, \ \ldots\ldots$$

Example 11.6 Find the least squares polynomial approximation of order 3 for e^x on [– 1, 1] using orthogonal polynomials.

Solution Use Legendre polynomials $\{P_r(x)\}$, the least squares polynomial approximation is

$$P(x)=\sum_{r=0}^{3}a_rP_r(x)$$

with a_r determined by

$$a_r=\frac{\langle f,P_r\rangle}{\langle P_r,P_r\rangle},$$

where $f(x)=e^x$ and hence

$$a_0=\frac{\langle f,P_0\rangle}{\langle P_0,P_0\rangle}=\frac{\int_{-1}^{1}f\,P_0dx}{\int_{-1}^{1}P_0^2dx}=\frac{\int_{-1}^{1}e^xdx}{\int_{-1}^{1}1dx}=\frac{e^1-e^{-1}}{2}=1.1752.$$

Similarly, we can obtain $a_1=1.1036$, $a_2=0.3578$, $a_4=0.0704$.

Remark: It can be shown that for the Legendre polynomials with $P_r(1)=1$,

$$\langle P_r,\,P_r\rangle=\int_{-1}^{1}P_r^2(x)dx=\frac{2}{2r+1},\qquad r=0,1,2,\ldots$$

Hence the Legendre polynomials fit (degree *n*) to *f*(*x*) can be determined by

$$f(x)\sim\sum_{r=0}^{n}a_rP_r(x)\quad,$$

where $$a_r=\frac{2r+1}{2}\langle f,\,P_r\rangle=\frac{2r+1}{2}\int_{-1}^{1}f(x)P_r(x)dx.$$

Example 11.7 Find the least squares approximation of order two to $f(t)=\sin t$ on $[0,\pi]$.

Solution. Remark: We can solve this problem by the following two steps

1^0 Constract orthogonal polynomial by formulae (11. 1);

2^0 Constract the least squares approximation $f(x)=\sum_{r=0}^{2}a_rP_r(x)$ with a_r determined by (11. 4).

Alternatively, we can solve the problem by using Legendre polynomials as follows

Firstly, we need to transform [0, π] to [– 1, 1] in order to use Legendre polynomials. As $t=\frac{a(1-x)+b(1+x)}{2}$ transforms $t\in[a,b]$ to $x\in[-1,1]$, let $t=\frac{\pi}{2}(x+1)$ ($a=0,\,b=\pi$ in this example). So the problem is equivalent to approximate

$$g(x)=f(t)=\sin\frac{\pi}{2}(x+1)\quad\text{for } x \text{ on } [-1,1].$$

Thus, the Legendre polynomial fit (degree 2) to $g(x)$ is

$$g(x) \sim \sum_{r=0}^{2} a_r P_r(x) \quad \text{with } a_r \text{ determined by } a_r = \frac{2r+1}{2}\int_{-1}^{1} g(x)P_r(x)dx.$$

$$\therefore \quad \sin\frac{\pi}{2}(x+1) \sim \frac{2}{\pi}+\frac{10}{\pi}\left(1-\frac{12}{\pi^2}\right)\frac{3x^2-1}{2}, \qquad x\in[-1,1]$$

$$\Rightarrow \quad \sin t \sim \frac{2}{\pi}+\frac{10}{\pi}\left(1-\frac{12}{\pi^2}\right)\left[\frac{3}{2}\left(\frac{2t}{\pi}-1\right)^2-\frac{1}{2}\right], \qquad t\in[0,\pi].$$

11.6 Least Squares Approximation using Chebyshev Polynomials

Definition : The Chebyshev polynomials $\{T_r(x)\}$ is a set of orthogonal polynomials defined on $[-1, 1]$ with weight function

$$w(x)=\frac{1}{\sqrt{1-x^2}}.$$

Theorem 11.1 A sequence on $[-1, 1]$ given by $T_n(x)=\cos(n\, ar\cos x)$ for $n\ge 0$, can define the Chebyshev polynomials.

Proof (Hint) To prove the theorem, we only need to show that

$$\langle T_n, T_m\rangle = \int_{-1}^{1}\frac{T_n(x)T_m(x)}{\sqrt{1-x^2}}dx \quad \begin{cases} =0 & n\ne m \\ \ne 0 & n=m. \end{cases}$$

Generation of Chebyshev Polynomials

a) By using general recurrence relations for orthogonal polynomials – See property (3) of orthogonal polynomials.

b) By using recurrence relations for Chebyshev polynomials

$$T_0(x)=1, \quad T_1(x)=x, \quad T_{n+1}(x)=2xT_n(x)-T_{n-1}(x) \text{ for } n\ge 1.$$

Proof (exercise)

Sequence of Chebyshev Polynomials

$$T_0=1, \quad T_1=x, \quad T_2=2xT_1-T_0=2x^2-1, \quad T_3=2xT_2-T_1=2x\left(2x^2-1\right)-x=4x^3-3x,$$

Exercise Show that

$$T_4=8x^4-8x^2+1, \quad T_5=16x^5-20x^3+5x, \quad T_6=32x^6-48x^4+18x^2-1, \ \ldots$$

Expansion of a Function in terms of Chebyshev Polynomials

A continuous function $f(x)$ defined on $[-1, 1]$ may be approximated in the weighted least squares sense, with weight function $w(x)=\dfrac{1}{\sqrt{1-x^2}}$, by

$$f(x)\sim\sum_{r=0}^{n} a_r T_r(x)$$

where
$$a_r=\frac{\langle f,\,T_r\rangle}{\langle T_r,\,T_r\rangle}=\begin{cases}\dfrac{1}{\pi}\displaystyle\int_{-1}^{1}\frac{f(x)T_r(x)}{\sqrt{1-x^2}}dx & r=0\\ \dfrac{2}{\pi}\displaystyle\int_{-1}^{1}\frac{f(x)T_r(x)}{\sqrt{1-x^2}}\mathrm{dx} & r=1,2,\ldots.\end{cases}$$

So we write
$$f(x)\sim\sum_{r=0}^{n}{}' a_r T_r(x)=\frac{1}{2}a_0+a_1T_1+a_2T_2+\ldots\ldots+a_nT_n$$

with
$$a_r=\frac{2}{\pi}\int_{-1}^{1}\frac{f(x)T_r(x)}{\sqrt{1-x^2}}dx \qquad (\,'\text{ signifies } a_0 \text{ is to be halved}).$$

Example 11.7 Find the least squares linear fit to $f(t)=t^2$ on $[0, 1]$ using $w(x)=\dfrac{1}{\sqrt{1-x^2}}$.

Solution

As $t=\dfrac{a(1-x)+b(1+x)}{2}$ takes $t\in[a,\,b]\;\rightarrow\; x\in[-1,\,1]$ and $a=0,\; b=1,$

let $t=\dfrac{x+1}{2}$. Then $f(t)$ define a function $g(x)$ on $[-1, 1]$.

i.e.,
$$g(x(t))=f(t)=\left(\frac{x+1}{2}\right)^2 \quad\text{for } x\in[-1,\,1].$$

The least squares fit to $g(x)$ using Chebyshev polynomials is

$$g(x)\sim\frac{1}{2}a_0T_0(x)+a_1T_1(x)$$

with a_r determined by

$$a_r=\frac{2}{\pi}\int_{-1}^{1}\frac{g(x)T_r(x)}{\sqrt{1-x^2}}dx$$

$$\therefore\; a_0=\frac{2}{\pi}\int_{-1}^{1}\frac{(x+1)^2}{4}\frac{1}{\sqrt{1-x^2}}dx$$

$$=\frac{1}{2\pi}\int_{-1}^{1}\frac{x^2+1}{\sqrt{1-x^2}}dx+\frac{1}{\pi}\int_{-1}^{1}\frac{x}{\sqrt{1-x^2}}dx=\frac{1}{\pi}\int_{0}^{1}\frac{x^2+1}{\sqrt{1-x^2}}dx.$$

Let $x = \sin\theta$, then

$$a_0 = \frac{1}{\pi}\int_0^{\pi/2}\frac{\left(1+\sin^2\theta\right)\cos\theta}{\cos\theta}d\theta = \frac{1}{\pi}\int_0^{\pi/2}\left(1+\sin^2\theta\right)d\theta = \int_0^{\pi/2}\left(\frac{3}{2}+\frac{\cos 2\theta}{2}\right)d\theta = \frac{3}{4}$$

Similarly we obtain

$$a_1 = \frac{2}{\pi}\int_{-1}^{1}\frac{(x+1)^2}{4}\frac{x}{\sqrt{1-x^2}}dx = \frac{1}{2}$$

$$\therefore\quad g(x) \sim \frac{1}{2}\left(\frac{3}{4}\right)T_0(x) + \frac{1}{2}T_1(x)$$

$$= \frac{3}{8}+\frac{1}{2}x.$$

As $f(t) = g(x(t))$, $f(t) = \frac{3}{8}+\frac{1}{2}(2t-1) = t-\frac{1}{8}.$

Advantage of Chebyshev Least Squares Approximation

The Chebyshev least squares approximation

$$C_n(x) = \sum_{j=0}^{n} a_j T_j(x)$$

to the function $f(x)$ on $[-1, 1]$ is more useful than the Legendre least squares approximation, which is established by the following theorem

Theorem 11.2 Let $f(x)$ have r continuous derivatives on $[-1, 1]$ with $r \ge 1$, then

$$\max_{x\in[-1,\,1]}\left|f(x)-C_n(x)\right| \le \frac{B\ln n}{n^r}, \qquad n \ge 2$$

for a constant B dependent on f and r. $C_n(x)$ converges uniformly to $f(x)$ as $n \to \infty$ provided $f(x)$ is continuously differentiable.

Proof See T. Rivlin, The Chebyshev Polynomials, John Wiley, 1974.

Numerical Evaluation of a_j

The coefficient a_j is defined by

$$a_j = \frac{2}{\pi}\int_{-1}^{1}\frac{f(x)T_j(x)}{\sqrt{1-x^2}}dx.$$

To evaluate a_j numerically, we firstly remove singularity in the integrand by letting $x = \cos\theta$ and thus

$$a_j = \frac{2}{\pi}\int_0^{\pi} f(\cos\theta)\cos(j\theta)d\theta.$$

The mid-point rule or the trapezoidal rule can then be used for the evaluation of the above integral.

Chebyshev Algorithm for the Evaluation of $C_n(x)$

Without converting to the form using x^j, the Chebyshev algorithm evaluates $C_n(x) = \sum_{j=0}^{n} a_j T_j(x)$ through an iteration process as follows

Chebyshev Algorithm for Computing $C_n(x)$

```
Set $b_{n+2} = 0$
    $b_{n+1} = 0$
    $z = 2x$
For $j = n, n-1, \ldots 0$ Do
    $b_j = zb_{j+1} - b_{j+2} + a_j$
$C_n(x) = (b_0 - b_2)/2$
```

Proof. Hint: 1) $a_j = b_j - 2xb_{j+1} + b_{j+2}$

2) $C_n(x) + \frac{a_0}{2} = \sum_{j=1}^{n} a_j T_j$

3) Find $\sum_{j=1}^{n} a_j T_j$ and use $T_{j+1}(x) - 2xT_j(x) + T_{j-1}(x) = 0$

Economization of Power Series

Chebyshev polynomials can also be used to reduce the degree of an approximating polynomial with a minimal loss of accuracy. This is particularly useful when the approximating polynomial is a Taylor polynomial.

A Taylor polynomial is very accurate near a point about which it is expanded, but poor when farther away from this point. For this reason, a high-degree Taylor Polynomial may be needed to achieve a prescribed error tolerance. Chebyshev polynomial may be used to reduce the degree of the Taylor polynomial without exceeding the error tolerance.

Example 11.8 Approximate $f(x) = e^x$ on [– 1, 1] with error tolerance 0.05.

Solution Taylor Polynomial:

degree 3: $$P_3 = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} \tag{11.5}$$

$$R_3 \le \frac{\left|f^{(4)}(\xi)x^4\right|}{24} \le \frac{e}{24} \approx 0.11 \quad \text{(Too large!)}$$

degree 4: $$P_4 = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} \tag{11.6}$$

$$R_4 \le \frac{e}{120} \approx 0.023 \quad \text{(OK.)}$$

Can we reduce P_4 to degree 3 without exceeding the error tolerance? Yes, as shown below

Since $$T_4(x) = 8x^4 - 8x^2 + 1$$

we have $$\frac{x^4}{24} = \frac{1}{24}\left(\frac{1}{8}T_4 + x^2 - \frac{1}{8}\right). \qquad (11.7)$$

Thus substituting (11.7) into (11.6), we get

$$P_4 = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{1}{24}\left(\frac{1}{8}T_4 + x^2 - \frac{1}{8}\right)$$

$$= \frac{191}{192} + x + \frac{13}{24}x^2 + \frac{x^3}{6} + \frac{1}{192}T^4$$

$$\therefore\ f = P_4 + R_4 = \frac{191}{192} + x + \frac{13}{24}x^2 + \frac{x^3}{6} + \left(\frac{1}{192}T^4 + R_4\right)$$

If we neglect the last term, then the error will be

$$E_{\text{total}} \le |R_4| + \left|\frac{1}{192}T_4\right| \le 0.023 + \frac{1}{192} = 0.0283 < 0.05$$

Hence, on neglecting the last term, we obtain the 3rd degree polynomial

$$P_3(x) = \frac{191}{192} + x + \frac{13}{24}x^2 + \frac{x^3}{6}$$

with maximum error 0.0283 on [– 1, 1].

11.7 Least Squares Approximation using Maple/MATLAB

Least Squares Approximation using Maple

The Maple function "`LeastSquares()`" constructs a least-squares approximation to the points $\{(x_i, y_i)\}_{i=1}^{N}$. The syntax is

```
LeastSquares([x1,x2,…,xn],[y1,y2,…,yn], v, option)
```

Notes: A linear function in variable `v` is returned unless the '`curve = f`' option is provided (the unknown parameters to determine are the indeterminates in `f` different from `v`). A weight can be assigned to each data point by using the '`weight = weightlist`' option where weightlist is a list containing n non-negative values.

Example 1

```
> with(curveFitting):
> y:=LeastSquares([0, 1, 2, 3], [1, 2, 3, 10], x);
```

returns a default function type with the form a*x+b:

$$\boxed{y := -\frac{1}{5} + \frac{14}{5}x}$$

Example 2

```
> with(curveFitting):
> y:=LeastSquares([0,1,2,3], [1,2,3,10], x, weight=[1,1,1,10]);
```

gives

$$\boxed{y := -\frac{47}{73} + \frac{253}{73}x}$$

Example 3

```
> with(curveFitting):
> y:=LeastSquares([0,1,3,5,6],[2,-1,-3,6,8],x,curve=a*x^2+b*x+c);
```

returns a function in the form of a*x^2+b*x+c as follows:

$$\boxed{y := \frac{487}{273} - \frac{330}{91}x + \frac{71}{21}x^2}$$

Least Squares Approximation using MATLAB

The MATLAB "polyfit( )" function finds the best coefficients of a polynomial that fits a set of data in a least-squares sense. The syntax is

```
p = polyfit(x,y,n)
```

where `x` and `y` are vectors containing the `x` and `y` data to be fitted, and n is the degree of the polynomial to return. For example,

```
  >> x = [0   1   3   5    6];
  >> y = [2  -1  -3   6   8];
  >> p = polyfit(x, y, 3)
```

yields

```
p =
    -0.1500    2.1595   -6.5571    2.4762
```

and thus the least square polynomial is

$$p = -0.1500x^3 + 2.1595x^2 - 6.5571x + 2.4762.$$

Note: The result `p` (output of polyfit) is a row vector of length `n+1` containing the polynomial coefficients in descending powers of `x`.

EXERCISES 11

Q11. 1 The average scores reported by golfers of various handicaps on a par-four hole were as follows:

Handicap	6	8	10	12	14	16	18	20	22	24
Average	4.6	4.8	4.6	4.9	5.0	5.4	5.1	5.5	5.6	6.0

Find the least-squares line for this data using normal equations. (Ans: y = 0.07x + 4.07)

Q11. 2 Derive the normal equations for finding the coefficients of the least squares fit

$$f(x) = a + b\,e^{-x}, \quad \text{to the data} \quad \{x_i, y_i\}_{i=1}^{N}.$$

In general, how to determine the coefficients c_i of the least squares fit

$$f(x) = \sum_{i=1}^{N} c_i P_i(x)$$

where P_i are linearly independent functions.

Q11. 3 Find a function of type $P = Ae^{Mx}$ for the data

x_i	1	2	3	4
y_i	60	30	20	15

using normal equations. (Ans: A = 84.8, M = – 0.456)

Q11. 4 Use orthogonal polynomials to find a least squares quadratic fit to

x_i	– 2	– 1	0	1	2
y_i	0.3	0.5	1	2	4.1

(Ans: y = 0.96 + 0.91x + 0.31x^2)

Q11. 5 Find the least-squares polynomial of degree two to fit

x_i	– 3	– 2	– 1	0	1	2	3
y_i	– 0.71	–0.01	0.51	0.82	0.88	0.81	0.49

(Ans: y = 0.806 + 0.200x – 0.102x^2).

Q11. 6 Find the least-squares parabola for $y(x) = x^3$ on the interval (– 1, 1). (Ans: 3x/5)

Q11. 7 Approximate $y(x) = 4/(2 + x)$ in the interval (2, 6) by a least squares polynomial of degree five. Transform to the interval (– 1, 1) and use Legendre polynomials.

Q11.8 Find the least squares parabola for $y(x) = x^3$ on (– 1, 1) with weight function $w(x) = 1/(1 - x^2)^{1/2}$. (Ans: 3x/4)

Q11.9 Represent $y(x)$=exp(–x) by terms of its power series through x^7. The error will be in the fifth decimal place for x near one. Rearrange the sum into Chebyshev polynomials. How many terms can then be dropped without seriously affecting the fourth decimal place? Rearrange the truncated polynomial into standard form. (this is another example of economisation of a polynomial). ($1.266T_0 - 1.1303T_1 + 0.2715T_2 - 0.0444T_3 + 0.0055T_4 - 0.0005T_5$)

Q11. 10 Economise the result $\ln(1+x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \frac{1}{5}x^5$ by rearranging into Chebyshev polynomials and then retaining only the quadratic terms. Show that the final result

$$\ln(1+x) = \frac{1}{32} + \frac{11}{8}x - \frac{3}{4}x^2$$

has about the same accuracy as the fourth degree part of the original approximation.

Programming

Q11. 11 Write a program to establish a least squares polynomial of degree n to fit a set of data using normal equations. Hence solve Q11.1 and Q11.3 using the program.

Program Structure

1) Write a subroutine to establish *Pn = a*0 + *a*1 * *x* + ... + *an* * *x****n*

SUBROUTINE LSPDEGN(M, X, Y, N, A)

M = before entry, M must be set to the number of data points.
X, Y = before entry, (X, Y) must contain the data (xi, yi).
N = before entry, N must be set to the required degree of poly.
A = on exit, A contains the coef. of polynomial A(0) = a0, A(1)= a1 ....

Algorithm:

Call FormCoef (M, X, Y, N, C, B) – *form normal equations C* ***a*** = ***b***
Call LUFACT – *factor C into L and U matrices*
Call SUBST – *substitution to find solution* ***a***.

Subroutine ***FormCoef*** **(*M, X, Y, N, C, B*)**

```
For $l = 0$ to $2n$ Do
        $s_l = \sum_{i=1}^{m} x_i^l$
End Do
For $k = 0$ to $n$ Do
        $b_k = \sum_{i=1}^{m} y_i x_i^k$
    For $j = k$ to $n$ Do
        Set $c_{kj} = s_{k+j}$
        if($k \neq j$) set $c_{jk} = c_{kj}$
    End Do
End Do
```

2) Write a subroutine ReadDat to read data.
3) Write a subroutine OutPDat to print the results.

Q11.12. Solve problems Q11.1 and Q11.5 using Maple/MATLAB built-in functions.

Chapter 12
The Eigenvalue Problem

The solution to many physical and mathematical problems requires the calculation of eigenvalues and corresponding eigenvectors of certain matrices. For matrices of order two or three, the eigenvalues and eigenvectors of the matrices can be easily determined analytically. However, for higher order matrices, one usually needs to determine the eigenvalues of the matrices numerically.

In this chapter, we introduce various numerical methods for the determination of eigenvalues and eigenvectors of matrices. The rest of the chapter is organized as follows.

Section 12.1	introduces some basic definitions and theorems providing the preliminaries for other sections.
Section 12.2	introduces the Power method for finding the eigenvalue with maximum magnitude.
Section 12.3	introduces various techniques for accelerating the convergence of the Power method.
Section 12.4	presents the inverse Power method for finding the eigenvalue closest to any given value.
Section 12.5	and Sections 12.6 presents the Jacobi's method and Given's method for finding all eigenvalues and eigenvectors of matrices.
Section 12.7	shows how to find eigenvalues and eigenvectors using Maple/Matlab.

Definition: (eigenvalue λ and eigenvector **x**)

Let A be an $n\times n$ matrix, $\mathbf{x}$ be an $n\times 1$ vector.

If there exists $\mathbf{x}\neq 0$, such that $A\mathbf{x}=\lambda\mathbf{x}$, then $\begin{cases}\lambda = \text{the eigenvalue of } A,\\ \mathbf{x} = \text{ the eigenvector of } A \text{ belonging to } \lambda.\end{cases}$

Direct Method for Calculating λ and x

Let λ be eigenvalue and $\mathbf{x}$ be an associated eigenvector, then

$$A\mathbf{x}=\lambda\mathbf{x} \quad\Leftrightarrow\quad (A-\lambda I)\mathbf{x}=0 .$$

As $\mathbf{x}\neq 0$, the above homogeneous system holds if and only if

$$P(\lambda)=\det(A-\lambda I)=0,$$

which is a polynomial equation of degree n and so it will have n roots $\lambda_1,\ \lambda_2,\ \ldots\ ,\ \lambda_n$ (real or complex). Hence,

to calculate λ:	solve $\det(\boldsymbol{A}-\lambda I)=0$ for λ;
to calculate **x** associated with each λ:	solve $(A-\lambda I)\mathbf{x}=0$ for **x**.

Example 12.1 Find eigenvalues and their associated eigenvector of matrix $A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix}$.

Solution

$$P(\lambda) = \det(A - \lambda I) = \begin{pmatrix} 1-\lambda & 4 \\ 3 & 2-\lambda \end{pmatrix} = \lambda^2 - 3\lambda - 10 = 0, \quad \Rightarrow \lambda = 5, \ -2.$$

Hence there are two eigenvalues $\lambda_1 = 5$ and $\lambda_2 = -2$. For each eigenvalue, we can determine an associated eigenvector by solving

$$(A - \lambda I)\mathbf{x} = 0. \tag{12.1}$$

Here we determine the eigenvector for $\lambda = 5$. Substituting $\lambda = 5$ into (12.1) yields

$$\begin{pmatrix} -4 & 4 \\ 3 & -3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \Rightarrow x_1 = x_2$$

So, any vector of the form $C\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ is the eigenvactor of $\boldsymbol{A}$ associated with $\lambda = 5$.

For ***uniqueness,*** we can normalise the eigenvector such that

$$\begin{cases} \text{either it has unit length in Euclidean norm} \sqrt{\sum x_i^2} & \left(\mathbf{x} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right); \\ \text{or the element of largest modulus is unity} & \left(\mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}\right). \end{cases}$$

12.1 Basic Definitions and Theorems for Eigenvlaue and Eigenvector

Some basic theorems for eigenvalues and eigenvectos.

(1) The eigenvalues of a symmetric matrix are real.

(2) A symmetric $n \times n$ matrix with n distinct eigenvalues has n linearly independent eigenvectors.

(3) The number of linearly independent eigenvectors associated with an eigenvalue of multiplicity m is m or less.

Gerschgorin's Discs Theorem (for approximation of eigenvalues)

Let $A = \{a_{ij}\}$ be an $n \times n$ matrix and C_i be the disc in the complex plane with centre at $(a_{ii}, 0)$ and radius

$$r_i = \sum_{j=1(j \neq i)}^{n} |a_{ij}|. \tag{12.2}$$

If D is the union of all the discs C_i (i =1 to n), then all the eigenvalues of A lie within D. Moreover, the union of any k of the these discs that do not intersect the remaining $(n - k)$ must contain precisely k eigenvalues.

Remark: From the data on each row of A, one can draw a Gerschgorin's disc.

Example 12.2 Use Gerschgorin's Discs Theorem to find bounad for the eigenvalues of

$$A = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 7 & 0 \\ -1 & 0 & -5 \end{bmatrix}$$

Solution 1^0 As A is symmetric, all the eigenvalues of A are real.

2^0 The Gerschgorin's Discs are

$$|\lambda - 1| = |2| + |-1| = 3$$
$$|\lambda - 7| = |2| + |0| = 2$$
$$|\lambda + 5| = |-1| + |0| = 1$$

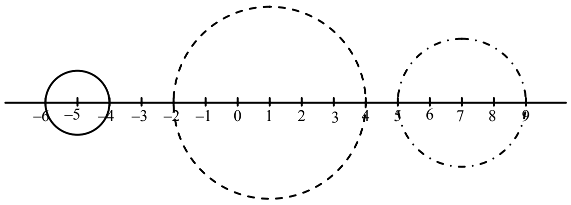


Figure 12.1 Gerschgorin's discs for example 12.2

From 1^0 and 2^0, each of the following intervals contains one eigenvalue :

[– 6, – 4], [– 2, 4] [5, 9].

12.2 The Power Method

In many cases, we require only one eigenvalue or eigenvector of the matrix A. For example, to analyse the convergence of the iterative methods for solving $A\mathbf{x} = \mathbf{b}$, we only need to calculate the eigenvalue with maximum modulus of the matrix A.

In this section, we introduce the Power Method, which is an iterative technique for finding the dominant eigen-solution (λ_1, $\mathbf{x}_1$) of a matrix, where λ_1 is the eigenvalue with maximum magnitude and $\mathbf{x}_1$ is the associated eigenvector, i.e.

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \text{......} \geq |\lambda_n| . \tag{12.3}$$

It is also assumed that the eigenvectors of A are linearly independent and that they are normalised so that their largest element is unity.

Numerical Algorithm of the Power Method

Let A be an $n \times n$ matrix. To find λ_1 and $\mathbf{x}_1$,

(1) Choose an $n \times 1$ column vector $\mathbf{z}^{(0)}$ whose largest element is unity;

(2) Perform iterations $\begin{cases} \mathbf{y}^{(k)} = A\mathbf{z}^{(k-1)} \\ \mathbf{z}^{(k)} = \dfrac{\mathbf{y}^{(k)}}{\mu_k} \end{cases}$ for $k = 1, 2, \ldots$ (12.4)

until convergence is achieved, where μ_k is the element of $\mathbf{y}(k)$ with largest modulus.

Remark: As $k \to \infty$, $\begin{cases} \mu_k \to \lambda_1 \\ \mathbf{z}^{(k)} \to \mathbf{x}_1 \end{cases}$

where λ_1 is the dominant eigenvlaue, and $\mathbf{x}_1$ is the associated eigenvector with the element of largest modulus being one.

Example 12.3 Find the dominant eigen-solution of the following matrix using the power method.

$$A = \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix}.$$

Solution

1^0 Choose $\mathbf{z}^{(0)} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

2^0 Perform iterations for k=1, 2, 3,…

$$\mathbf{y}^{(k)} = A\mathbf{z}^{(k-1)}$$

$$\mathbf{z}^{(k)} = \frac{\mathbf{y}^{(k)}}{\mu_k}$$

where μ_k := element of $\mathbf{y}^{(k)}$ with largest modules.

For $k = 1$, $\mathbf{y}^{(1)} = A\mathbf{z}^{(0)} = \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$

$\mu_1 = -2$

$\mathbf{z}^{(1)} = \frac{\mathbf{y}^{(1)}}{\mu_1} = \begin{bmatrix} -\frac{1}{2} \\ 1 \\ -\frac{1}{2} \end{bmatrix}$

For $k = 2$, $\mathbf{y}^{(2)} = A\mathbf{z}^{(1)} = \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix}\begin{bmatrix} -\frac{1}{2} \\ 1 \\ -\frac{1}{2} \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 2 \end{bmatrix}$

$\mu_2 = -3$

$\mathbf{z}^{(2)} = \frac{\mathbf{y}^{(2)}}{\mu_2} = \begin{bmatrix} -\frac{2}{3} \\ 1 \\ -\frac{2}{3} \end{bmatrix}$

For $k = 3$, $\quad \mathbf{y}^{(3)} = A\mathbf{z}^{(2)} = \begin{bmatrix} \frac{7}{3} \\ -\frac{1}{3} \\ \frac{7}{3} \end{bmatrix}, \quad \mu_3 = -\frac{10}{3}$

Continuing on, we obtain the following sequences of μ_k and $z^{(k)}$:

$$\mu_k : \quad -2, -3, -3.3333, -3.4000, -3.4118, -3.4138, -3.4142;$$

$$\mathbf{z}^{(k)} : \quad \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -\frac{1}{2} \\ 1 \\ -\frac{1}{2} \end{bmatrix}, \begin{bmatrix} -\frac{2}{3} \\ 1 \\ -\frac{2}{3} \end{bmatrix}, \ldots, \begin{bmatrix} -0.7071 \\ 1 \\ -0.7071 \end{bmatrix}, \begin{bmatrix} -0.7071 \\ 1 \\ -0.7071 \end{bmatrix}.$$

As the sequences have converged, we have

$$\lambda_1 = -3.4142, \quad \mathbf{x}_1 = \begin{bmatrix} -0.7071 \\ 1 \\ -0.7071 \end{bmatrix}.$$

Convergence Test

We usually stop computation and let $\lambda_1 = \mu_k$, $\quad \mathbf{x}_1 = \mathbf{z}^{(k)}$ if

1^0 The size of λ_1 converge, i.e.,

$$\frac{|\mu_k - \mu_{k-1}|}{|\mu_{k-1}|} < \text{Tol}. \tag{12.5}$$

2^0 The element with maximum modules occurs at the some position of the eigenvector at any two consecutive iterations.

Analysis of the Power Method

In the following we prove $\begin{cases} \mu_k \to \lambda_1 \\ \mathbf{z}^{(k)} \to \mathbf{x}_1 \end{cases}$ as $k \to \infty$.

Proof

It has been assumed that the normalised eigenvectors of A, $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, are linearly independent, so

$$\mathbf{z}^{(0)} = \sum_i^N \alpha_i \mathbf{x}_i$$

From (12.4),

$$\mathbf{y}^{(1)} = A\mathbf{z}^{(0)} = \sum_i \alpha_i A\mathbf{x}_i = \sum_i \alpha_i \lambda_i \mathbf{x}_i, \qquad \mathbf{z}^{(1)} = \frac{\mathbf{y}^{(1)}}{\mu_1} = \frac{\sum \alpha_i \lambda_i \mathbf{x}_i}{\mu_1}.$$

$$\mathbf{y}^{(2)} = A\mathbf{z}^{(1)} = \frac{\sum \alpha_i \lambda_i A\mathbf{x}_i}{\mu_1} = \frac{\sum \alpha_i \lambda_i^2 \mathbf{x}_i}{\mu_1}, \qquad \mathbf{z}^{(2)} = \frac{\mathbf{y}^{(2)}}{\mu_2} = \frac{\sum \alpha_i \lambda_i^2 \mathbf{x}_i}{\mu_1 \mu_2}.$$

In general $$\mathbf{z}^{(k)} = \frac{\sum \alpha_i \lambda_i^k \mathbf{x}_i}{\mu_1\mu_2 \dots \mu_k} = \frac{\lambda_1^k \left[\alpha_1 \mathbf{x}_1 + \alpha_2 (\lambda_2/\lambda_1)^k \mathbf{x}_2 + \dots + \alpha_n (\lambda_n/\lambda_1)^k \mathbf{x}_n\right]}{\mu_1\mu_2 \dots \mu_k}.$$

As $|\lambda_i| < |\lambda_1| \;\; \forall\, i \neq 1$, $$\mathbf{z}^{(k)} \to \frac{\alpha_1 \lambda_1^k}{\mu_1\mu_2\dots\mu_k}\mathbf{x}_1 \text{ as } k \to \infty. \tag{12.6}$$

Further as both $\mathbf{z}^{(k)}$ and $\mathbf{x}_1$ are normalised vector with largest element unity, we have from (12.6)

$$\mathbf{z}^{(k)} \to \mathbf{x}_1\,, \text{ as } k \to \infty. \tag{12.7}$$

$$\frac{\alpha_1 \lambda_1^k}{\mu_1\mu_2\dots\mu_k} \to 1\,, \text{ as } k \to \infty. \tag{12.8}$$

Equation (12.8) implies that

$$\begin{cases} \mu_1\mu_2\dots\mu_k = \alpha_1\lambda_1^k \\ \mu_1\mu_2\dots\mu_k\mu_{k+1} = \alpha_1\lambda_1^{k+1} \end{cases} \quad \therefore \quad \mu_{k+1} = \lambda_1 \text{ as } k \to \infty. \tag{12.9}$$

12.3 Acceleration Techniques for the Power Method

In the power method, rate of convergence is governed mainly by λ_2/λ_1. Thus, if $\lambda_2 \approx \lambda_1$, the rate of convergence is very slow. Fortunately, various techniques are available to accelerate the rate of convergence.

Aitken's Δ^2 Process

This process can be used to accelerate the convergence of a linearly convergent iteration scheme

$$x_{n+1} = g(x_n) \tag{12.10}$$

Method: generate the following sequence using both (12.10) and Aitken's formula (12.11):

$$x_1, x_2, x_3, \hat{x}_4, x_5, x_6, \hat{x}_7, x_8, x_9, \hat{x}_{10}, \dots$$

where x_i is obtained from (12.10) and $\hat{x}_i$ is obtained from (12.11)

$$\hat{x}_{n+3} = x_{n+2} - \frac{(\Delta x_{n+1})^2}{\Delta^2 x_n} \tag{12.11}$$

in which

$$\Delta x_{n+1} = x_{n+2} - x_{n+1}\,, \quad \Delta^2 x_n = x_{n+2} - 2x_{n+1} + x_n\,.$$

Remark: The sequence generated by (12.11) converges faster than the sequence generated by (12.10). More details of Aitken's Δ^2 process can be found in Section 2.7

Application to The Power Method

Starting from $\mathbf{z}^{(0)}$, generate $\{\mathbf{y}^{(k)}, \mu_k, \mathbf{z}^{(k)}\}$ using both the power method (12.4) and the Δ^2 process (12.12) in the order as shown below.

k	$\mathbf{y}^{(k)}$	μ_k	$\mathbf{z}^{(k)}$	Formulae for $\mathbf{y}^{(k)}$
0			$\mathbf{z}^{(0)}$	
1	$\mathbf{y}^{(1)}$	μ_1	$\mathbf{z}^{(1)}$	P.M.
2	$\mathbf{y}^{(2)}$	μ_2	$\mathbf{z}^{(2)}$	P.M.
3	$\mathbf{y}^{(3)}$	μ_3		P.M.
4	$\mathbf{y}^{(4*)}$	μ_4	$\mathbf{z}^{(4)}$	Δ^2 process
5	$\mathbf{y}^{(5)}$	μ_5	$\mathbf{z}^{(5)}$	P.M.
6	$\mathbf{y}^{(6)}$	μ_6		P.M.
7	$\mathbf{y}^{(7*)}$	μ_7	$\mathbf{z}^{(7)}$	Δ^2 process

where $\mathbf{y}^{(k)}$ are calculated from the power method (P.M), while $\mathbf{y}^{(k*)}$ are determined by the Δ^2 process, eg.

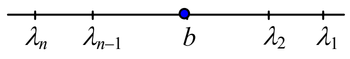
$$y_i^{(4*)} = y_i^{(3)} - \frac{\left[y_i^{(3)} - y_i^{(2)}\right]^2}{y_i^{(3)} - 2y_i^{(2)} + y_i^{(1)}}. \qquad (12.12)$$

Shift of Origin

From the definition of eigenvalue and eigenvector, it can be proved that if λ is an eigenvalue of A and $\mathbf{x} \neq 0$ is the associated eigenvector, then $\lambda - b$ is an eigenvalue of $A - bI$ and $\mathbf{x}$ is its associated eigenvector.

Now suppose that the eigenvalues of A are $\lambda_1 > \lambda_2 \geq \lambda_3 \geq \ldots \geq \lambda_n$, (12.13)
then the eigenvalues of $A - bI$ are $\lambda_1 - b, \lambda_2 - b, \ldots, \lambda_n - b$.

As the modulus of $\lambda_i - b$ is the distance from λ_i to b (as shown in the figure below), the dominant eigenvalue (with maximum magnitude) of $A - bI$ must be either $\lambda_1 - b$ or $\lambda_n - b$. We study these two cases in the following.



1) For the case $|\lambda_1 - b| > |\lambda_n - b|$, following the same procedure as that for deriving (12.6), we obtain

$$\mathbf{z}^{(k)} = \frac{(\lambda_1 - b)^k \left[\alpha_1 \mathbf{x}_1 + \alpha_1 \left(\frac{\lambda_2 - b}{\lambda_1 - b}\right)\mathbf{x}_2 + \ldots + \alpha_n \left(\frac{\lambda_n - b}{\lambda_1 - b}\right)\mathbf{x}_n\right]}{\mu_1 \mu_2 \mu_3 \ldots \mu_k}$$

$$\mathbf{z}^{(k)} \to \frac{(\lambda_1 - b)^k \alpha_1}{\mu_1 \mu_2 \ldots \mu_k} \mathbf{x}_1 \quad \text{as} \quad k \to \infty.$$

$$\therefore \quad \mathbf{z}^{(k)} \to \mathbf{x}_1, \quad \mu_k \to (\lambda_1 - b).$$

Thus, the rate of convergence depends mainly on $v = \max\left\{\frac{\lambda_2 - b}{\lambda_1 - b}, \frac{\lambda_n - b}{\lambda_1 - b}\right\}$.

It can be noted that the minimum of v occurs when b is at the mid point of the interval $[\lambda_n, \lambda_2]$, i.e.

$$b = (\lambda_n + \lambda_2)/2 \tag{12.14}$$

which is the optimal choice of b for calculating λ_1.

2) For the case $|\lambda_n - b| > |\lambda_1 - b|$, similar to the analysis in 1), we obtain

$$\mathbf{z}^{(k)} \to \mathbf{x}_1, \quad \mu_k \to (\lambda_n - b) \quad \text{as} \quad k \to \infty.$$

Similarly, the fastest convergence rate is achieved if

$$b = (\lambda_{n-1} + \lambda_1)/2 \tag{12.15}$$

which is the optimal choice b for calculating λ_n.

Remark: Combined with the shift of origin, the power method can be used to determine λ_1 and λ_n where $\lambda_1 > \lambda_2 > \ldots > \lambda_n$. To determine λ_1, choose b=(λ_n+ λ_1)/2; to determine λ_n, choose b=(λ_{n-1}+ λ_1)/2.

Example 12.3 Given $A = \begin{pmatrix} 4 & -6 & 5 \\ -6 & 3 & 4 \\ 5 & 4 & -3 \end{pmatrix}$,

apply the power method to $A - bI$ using an 'optimal' value for b to find the dominant eigenvalue of A.

Solution

Firstly, use Gerschgorin's discs theorem to get a rough estimate of the eigenvalues of A. As the matrix is symmetric, all eigenvalues are real. Together with the Gerschgorin's discs theorem, we have 3 real eigenvalues respectively in the range [-7,15], [-7, 13], [-12,6] as shown in Figure 12.2.

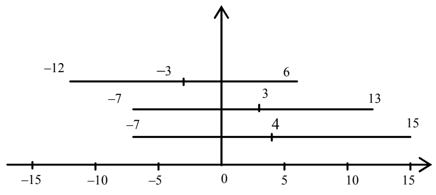


Figure 12.2 Estimate of eigenvalues using Gerschgorin's discs theorem

As an approximation, let $\lambda_1 = 15, \quad \lambda_2 = 6, \quad \lambda_3 = -12.$

To find the actual value of λ_1 , choose $b = \frac{1}{2}(\lambda_2 + \lambda_3) = \frac{1}{2}(6-12) = -3$

Construct $B = A - bI = A + 3I = \begin{pmatrix} 7 & -6 & 5 \\ -6 & 6 & 4 \\ 5 & 4 & 0 \end{pmatrix}$

The eigenvalues of B will then be

$$\lambda_1 - b = \lambda_1 + 3 \approx 18$$
$$\lambda_2 - b = \lambda_2 + 3 \approx 9$$
$$\lambda_3 - b = \lambda_3 + 3 \approx -9.$$

Thus the dominant eigenvalue of B is expected to be λ_1+3 and so, if we apply the Power method to B, it is expected that

$$\mu_k \rightarrow \lambda_1 + 3$$

$\mathbf{z}^{(k)} \rightarrow \mathbf{x}_1$ (where $\mathbf{x}_1$ is the eigenvector of A associated with λ_1)

Now apply the Power method to B.

Choose $\mathbf{z}^{(0)} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

$$k = 1, \quad \mathbf{y}^{(1)} = B\mathbf{z}^{(0)} = \begin{bmatrix} 7 & -6 & 5 \\ -6 & 6 & 4 \\ 5 & 4 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -6 \\ 6 \\ 4 \end{bmatrix}$$

$$\mu_1 = -6$$

$$\mathbf{z}^{(1)} = \frac{\mathbf{y}^{(1)}}{\mu_1} = \begin{bmatrix} 1 \\ -1 \\ -0.6667 \end{bmatrix}$$

$k = 2,$

$\vdots$

$k = 16, \quad \mu_{16} = 12.6219, \qquad \mathbf{z}^{(16)} = (1, -0.8247, 0.1348)^T$

$k = 17, \quad \mu_{17} = 12.6221, \qquad \mathbf{z}^{(17)} = (1, -0.8247, 0.1348)^T$

Hence, $\lambda_1 + 3 = \mu_{17} = 12.6221 \Rightarrow \lambda_1 = 9.6221$

$\mathbf{x}_1 = \mathbf{z}^{(17)} = (1, -0.8247, 0.1348)^T.$

Rayleigh Quotient

When the eigenvalue alone is required, the formula

$$\mathbf{q}^{(k+1)} = \frac{\mathbf{y}^{(k+1)T}\mathbf{z}^{(k)}}{\mathbf{z}^{(k)T}\mathbf{z}^{(k)}} \quad (k = 0, 1, 2, \ldots). \tag{12.16}$$

can be used in conjunction with the power method to give a sequence of approximations $\mathbf{q}^{(1)}, \mathbf{q}^{(2)}, \ldots,$ which converges more rapidly to λ_1.

12.4 The Inverse Power Method

The method presented in this section can be used to find

(a) the eigenvalue of A with least modulus and its associated eigenvector;

(b) any eigenvalue of A closest to a given value p and its associated eigenvector.

The idea behind the method is to construct a new matrix B such that the dominant eigenvalue of B is related to the eigenvalue of A to be determined. Then apply the power method to B to find its dominant eigenvalue and consequently the related eigenvalue of A.

Find the Eigenvalue with Least Modulus and its Associated Eigenvector

Construct $B = A^{-1}$. Then it can be shown, from the definition of eigenvalue and eigenvector, that

$$\begin{cases} \text{eigenvalue of } B = \dfrac{1}{\text{eigenvalue of } A}, \quad \text{i.e} \quad \lambda(B) = \dfrac{1}{\lambda(A)}. \\ \text{eigenvector of } B = \text{eigenvector of } A. \end{cases}$$

Suppose that the eigenvalues of A (λ_i) are such that

$$|\lambda_1| \geq |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_{n-1}| > |\lambda_n|,$$

then the eigenvalues of $B = A^{-1}$ ($1/\lambda_i$) are such that

$$\left|\frac{1}{\lambda_1}\right| \leq \left|\frac{1}{\lambda_2}\right| \leq \left|\frac{1}{\lambda_3}\right| \leq \ldots \leq \left|\frac{1}{\lambda_{n-1}}\right| < \left|\frac{1}{\lambda_n}\right|.$$

Obviously, if the eigenvalue of A with least modulus and its associated eigenvector are λ_n and $\mathbf{x}_n$, then the dominant eigen solution of B is $\{1/\lambda_n, \mathbf{x}_n\}$. Thus applying the power method to B, we can find its dominant eigen-solution,

$$\boxed{\frac{1}{\lambda_n} = \mu_k, \quad \mathbf{x}_n = \mathbf{z}^{(k)}, \qquad \text{(for } k \text{ sufficiently large)}}$$

Then it is a simple matter to find $\{\lambda_n, \mathbf{x}_n\}$ from the above formulae.

Find the Eigenvalue Closest to a Given Value *p* and its Associated Eigenvector

To find a single eigenvalue λ^* closest to p and its associated eigenvector $\boldsymbol{x}^*$, construct $B = A - p\mathrm{I}$. It can be shown that

$$\begin{cases} \text{eigenvalue of } B^{-1} = \dfrac{1}{(\text{eigenvalue of } A) - p}, \quad \text{i.e} \quad \lambda_i(B^{-1}) = \dfrac{1}{\lambda_i(A) - p}. \\ \text{eigenvector of } B^{-1} = \text{eigenvector of } A. \end{cases}$$

As λ^* is the eigenvalue closest to p, we have $|\lambda^* - p| < |\lambda_i - p|$ (for $\lambda_i \neq \lambda^*, i = 1, 2, ..., n$). Hence, the dominant eigen solution of B^{-1} is $\{1/(\lambda^* - p), \mathbf{x}^*\}$, which can be determined by the power method applied to B^{-1}, i.e.

$$\frac{1}{\lambda^* - p} = \mu_k, \quad \mathbf{x}^* = \mathbf{z}^{(k)} \qquad \text{(for } k \text{ sufficiently large)}$$

From the above formulae, it is then a simple matter to determine λ^* and $\mathbf{x}^*$. To find the dominant eigen-solution of B^{-1}, $\left(\dfrac{1}{\lambda^* - p}, \mathbf{x}^*\right)$, we may directly apply the power method to B^{-1}, i.e.

$$\text{perform iteration} \begin{cases} \mathbf{y}^{(k)} = B^{-1}\mathbf{z}^{(k-1)} \\ \mathbf{z}^{(k)} = \mathbf{y}^{(k)} / \mu_k \end{cases} \quad (k = 1, 2, ...), \qquad (12.17)$$

where μ_k is the element of $\mathbf{y}$(k) with largest modulus.

$$\text{As} \quad k \to \infty, \quad \begin{cases} \mu_k \to \dfrac{1}{\lambda^* - p} \\ \mathbf{z}^{(k)} \to \mathbf{x}^*. \end{cases}$$

To avoid actually calculating B^{-1}, we can use

$$\begin{cases} B\mathbf{y}^{(k)} = \mathbf{z}^{(k-1)} \\ \mathbf{z}^{(k)} = \mathbf{y}^{(k)} / \mu_k \end{cases} \quad (k = 1, 2, ...).$$

As the first set of equations in the above system needs to be solved many times, the *LU* decomposition method is usually used for the solution. Let $B=LU$, then

$$\begin{cases} L\mathbf{v}^{(k)} = \mathbf{z}^{(k-1)} \\ U\mathbf{y}^{(k)} = \mathbf{v}^{(k)} \\ \mathbf{z}^{(k)} = \mathbf{y}^{(k)} / \mu_k \end{cases} \quad (k = 1, 2, ...). \qquad (12.18)$$

Example 12.4 Given $A = \begin{pmatrix} 2 & -3 & 0 \\ -3 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$, find the eigenvalue closest to –2 and its associated eigenvector, take $\mathbf{z}^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$.

Solution For $p = -2$,

Construct $B = A - pI = A + 2I = \begin{pmatrix} 4 & -3 & 0 \\ -3 & 3 & 1 \\ 0 & 1 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{3}{4} & 1 & 0 \\ 0 & \frac{4}{3} & 1 \end{pmatrix} \begin{pmatrix} 4 & -3 & 0 \\ 0 & \frac{3}{4} & 1 \\ 0 & 0 & \frac{8}{3} \end{pmatrix}$.

Suppose the eigenvalues of A are $\lambda_i(A)$, $i = 1, 2, 3$. Then, the eigenvalues of B^{-1} are

$$\frac{1}{\lambda_i(A)+2} \quad (i = 1, 2, 3).$$

Thus, if λ_ℓ is the eigenvalue of A closest to –2, then the eigenvalue of B with maximum modulus is $\dfrac{1}{\lambda_\ell + 2}$. Hence if we apply the power method to B^{-1}, it is expected that

$$\mu_k \to \frac{1}{\lambda_\ell + 2}$$
$$\mathbf{z}^{(k)} \to \mathbf{x}_\ell .$$

Now, apply the Power method to B^{-1} to find μ_k and $\mathbf{z}^{(k)}$ by using

$$\text{either} \quad \begin{cases} \mathbf{y}^{(k)} = B^{-1}\mathbf{z}^{(k-1)} \\ \mathbf{z}^{(k)} = \mathbf{y}^{(k)} / \mu_k \end{cases} \quad \text{or} \quad \begin{cases} L\mathbf{v}^{(k)} = \mathbf{z}^{(k-1)} \\ U\mathbf{y}^{(k)} = \mathbf{v}^{(k)} \\ \mathbf{z}^{(k)} = \mathbf{y}^{(k)} / \mu_k \end{cases} \quad (k = 1, 2, \ldots).$$

In the following, we use the 2nd set of formulae

$$k = 1, \qquad \begin{bmatrix} 1 & 0 & 0 \\ -\frac{3}{4} & 1 & 0 \\ 0 & \frac{4}{3} & 1 \end{bmatrix} \begin{bmatrix} v_1^{(1)} \\ v_2^{(1)} \\ v_3^{(1)} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{v}^{(1)} = \begin{pmatrix} 1 \\ \frac{7}{4} \\ -\frac{4}{3} \end{pmatrix}$$

$$\begin{bmatrix} 4 & -3 & 0 \\ 0 & \frac{3}{4} & 1 \\ 0 & 0 & \frac{8}{3} \end{bmatrix} \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \\ y_3^{(1)} \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{7}{4} \\ -\frac{4}{3} \end{bmatrix} \quad \Rightarrow \quad \mathbf{y}^{(1)} = \begin{pmatrix} \frac{5}{2} \\ 3 \\ -\frac{1}{2} \end{pmatrix}$$

$$\therefore \quad \mu_1 = 3$$

$$\mathbf{z}^{(1)} = \frac{\mathbf{y}^{(1)}}{\mu_1} = \begin{pmatrix} \frac{5}{6} \\ 1 \\ -\frac{1}{6} \end{pmatrix}$$

$k = 2, \qquad \mathbf{v}^{(2)} \to \mathbf{y}^{(2)} \to \mu_2 \to \mathbf{z}^{(2)}$

$\vdots$

$k = 4, \qquad \mu_4 = 3.3507, \qquad \mathbf{z}^{(4)} = (0.0815,\ 1,\ -0.2701)^{\mathrm{T}}$

$k = 5, \qquad \mu_5 = 3.3508\ , \qquad \mathbf{z}^{(5)} = (0.0815,\ 1,\ -0.2701)^{\mathrm{T}}$

The sequence μ_k and $\mathbf{z}^{(k)}$ both converge and so

$$\frac{1}{\lambda_\ell + 2} = \mu_5 = 3.3508 \qquad \Rightarrow \quad \lambda_\ell = -1.702\,,$$

$$\mathbf{x}_\ell = \mathbf{z}^{(5)} = (0.0815,\ 1,\ -0.2701)^{\mathrm{T}}\,.$$

12.5 Methods for a Complete Eigen System

When all the eigenvalues of a matrix A are required, methods based on **similarity transformations** are usually employed. These methods make use of an **orthogonal matrix** P to transform A into a **similar matrix** B which is of simpler form. In this section, we first introduce some basic concepts, then present a method based on similarity transformation, namely the Jacobi's method, for finding eigenvalues and eigenvectors.

Basic Concepts of Similarity Transformation

Definition (Orthogonal Vectors): A set of vectors { $\mathbf{y}_1$, $\mathbf{y}_2$, ….., $\mathbf{y}_n$ } is said to be

$$\begin{cases} \textbf{orthogonal} & \text{if } \mathbf{y}_i^T\mathbf{y}_j = 0 \ \ \forall i \neq j \\ \textbf{orthonormal} & \text{if, in addition, } \mathbf{y}_i^T\mathbf{y}_i = 1. \end{cases}$$

Definition (Orthogonal Matrices): An $n \times n$ matrix P is said to be **orthogonal** if

$$P^T P = I \quad \Leftrightarrow \quad P^T = P^{-1}.$$

Definition (Similar matrices): Two $n \times n$ matrices, A and B, are said to be **similar** if there exists a non-singular matrix S, such that

$$S^{-1}AS = B.$$

Definition (Similarity transformation): The process which transforms A to a similar matrix B (via $B = S^{-1}AS$) is called **similarity transformation**.

Property of Similarity Transformation

The eigenvalues of a matrix are **invariant** under a similarity transformation. More specifically, suppose B is obtained from A by a similarity transformation $B = S^{-1}AS$, then if $\{\lambda, \mathbf{x}\}$ is an eigen solution of A, $\{\lambda, S^{-1}\mathbf{x}\}$ is the eigen solution of B.

Methods based on Similarity Transformations

1) Transform A into a similar matrix B which is of simpler form $B = P^{-1}AP$ using an orthogonal matrix P.
2) Determine the eigenvalues and eigenvectors of B and consequently of A using the property of similarity transformation.

The Theoretical Basis of Jacobi's Method

Theorem 12.1 If A is symmetric ($A^T = A$), then there exists an orthogonal matrix P such that $P^TAP=D$ where D is a diagonal matrix. A and D are similar and hence

(1) eigenvalues of A (λ_i) = eigenvalues of D [$\lambda_i(D)$] $=d_i$

(2) eigenvectors of A = Columns of P .

Proof

(1) First prove $\lambda_i(A)] = d_i$.
As A and D are similar, $\lambda_i(A) = \lambda_i(D)$. The characteristic equation for D is

$$p(\lambda) = |D - \lambda I| = \prod_{i=1}^{n} (d_i - \lambda_i) = 0 \quad \text{which gives } \lambda_i(D) = d_i .$$

(2) Now prove that the eigenvectors of A are the columns of P.

$$\text{As } P^T AP = D, \text{ we have } AP = PD \Leftrightarrow A[P_1 \ P_2 \ \ldots P_n] = [P_1 \ P_2 \ \ldots P_n]\begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}.$$

$$\therefore \quad [AP_1 \ AP_2 \ \ldots AP_n] = [\lambda_1 P_1 \ \lambda_2 P_2 \ \ldots \lambda_n P_n] \quad \Rightarrow \quad AP_i = \lambda_i P_i$$

which implies that P_i (the ith column of P) is the eigenvector associated with λ_i.
The proof is completed. □

The **Jacobi's method** is based on the above theorem and can be used to find all the eigenvalues and eigenvectors of a symmetric matrix A. The method transforms A ($n \times n$) into diagonal form by annihilating its off-diagonal elements one by one through a finite number of similarity transformations using plane rotation matrices Rpq ($n \times n$), i.e. $A \xrightarrow{\text{via } \hat{R}^T AR} D$. Thus, according to the above theorem, the eigenvalues of A are the diagonal elements of the diagonal matrix, i.e. $\lambda_i = d_i$, and the eigenvector associated with λ_i is the ith column of $\hat{R}$. The key question is how to eliminate the off diagonal elements of A.

In the following, we introduce the plane rotation matrix followed by its properties, then present a theorem which describes how to eliminate an off-diagonal element of a matrix by using the plane rotation matrices followed by the proof which makes use of the properties of the plane rotation matrices.

Definition (Plane rotation matrices): A plane rotation matrix is basically an unit matrix except for the elements $r_{pp} = r_{qq} = \cos\theta = c, \quad r_{qp} = \sin\theta = s, \quad r_{pq} = -\sin\theta,$ i.e.,

$$R_{pq} = \begin{pmatrix} 1 & & 0 & & 0 & & 0 \\ & & \vdots & & \vdots & & \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ & & \vdots & & \vdots & & \\ 0 & & s & & c & & 0 \\ & & \vdots & & \vdots & & \\ 0 & & 0 & & 0 & & 1 \end{pmatrix} \tag{12.19}$$

Properties of Plane Rotation Matrices $R_{pq}(\theta)$

Premultiplication of A by R_{pq}^T affects only the elements in rows p and q

$$R_{pq}^T A = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_{11} & \cdots & a_{1p} & \cdots & a_{1q} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots & & \vdots \\ a_{p1} & \cdots & a_{pp} & \cdots & a_{pq} & \cdots & a_{pn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ a_{q1} & \cdots & a_{qp} & \cdots & a_{qq} & \cdots & a_{qn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ a_{n1} & \cdots & a_{np} & \cdots & a_{nq} & \cdots & a_{nn} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & \cdots & a_{1p} & \cdots & a_{1q} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots & & \vdots \\ \overline{a}_{p1} & \cdots & \overline{a}_{pp} & \cdots & \overline{a}_{pq} & \cdots & \overline{a}_{pn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ \overline{a}_{q1} & \cdots & \overline{a}_{qp} & \cdots & \overline{a}_{qq} & \cdots & \overline{a}_{qn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ a_{n1} & \cdots & a_{np} & \cdots & a_{nq} & \cdots & a_{nn} \end{bmatrix},$$

where

$$\overline{a}_{pj} = ca_{pj} + sa_{qj}, \ \overline{a}_{qj} = -sa_{pj} + ca_{qj}. \tag{12.20}$$

Postmultiplication of *a matrix* by R_{pq} affects only the elements in columns p and q of the matrix.

$$\left(R_{pq}^T A\right) R_{pq} = \begin{bmatrix} a_{11} & \cdots & a_{1p} & \cdots & a_{1q} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots & & \vdots \\ \overline{a}_{p1} & \cdots & \overline{a}_{pp} & \cdots & \overline{a}_{pq} & \cdots & \overline{a}_{pn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ \overline{a}_{q1} & \cdots & \overline{a}_{qp} & \cdots & \overline{a}_{qq} & \cdots & \overline{a}_{qn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ a_{n1} & \cdots & a_{np} & \cdots & a_{nq} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & \cdots & a^*_{1p} & \cdots & a^*_{1q} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots & & \vdots \\ \bar{a}_{p1} & \cdots & a^*_{pp} & \cdots & a^*_{pq} & \cdots & \bar{a}_{pn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ \bar{a}_{q1} & \cdots & a^*_{qp} & \cdots & a^*_{qq} & \cdots & \bar{a}_{qn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ a_{n1} & \cdots & a^*_{np} & \cdots & a^*_{nq} & \cdots & a_{nn} \end{bmatrix}$$

where $\left.\begin{aligned} a^*_{ip} &= ca_{ip} + sa_{iq} = \bar{a}_{pi} \\ a^*_{iq} &= -sa_{ip} + ca_{iq} = \bar{a}_{qi} \end{aligned}\right\} i \neq p,\, q,$

$$a^*_{pp} = c\bar{a}_{pp} + s\bar{a}_{pq} = c^2 a_{pp} + 2csa_{pq} + s^2 a_{qq}$$

$$a^*_{qq} = -s\bar{a}_{qp} + c\bar{a}_{qq} = s^2 a_{pp} - 2csa_{pq} + c^2 a_{qq}$$

$$a^*_{pq} = -s\bar{a}_{pp} + c\bar{a}_{pq}$$

$$= -csa_{pp} - s^2 a_{qp} + c^2 a_{qp} + csa_{qq} \xLeftrightarrow[\text{symmetric}]{a_{ij}=a_{ji}} \left(c^2 - s^2\right) a_{pq} + cs\left(a_{qq} - a_{pp}\right) = a^*_{qp} \tag{12.21}$$

Remark: In the transformation $A^* = R^T_{pq} A R_{pq}$, the new matrix A^* is essentially the same as A except for the elements

$$\left.\begin{aligned} a^*_{ip} &= ca_{ip} + sa_{iq} = \bar{a}_{pi} \\ a_{iq} &= -sa_{ip} + ca_{iq} = \bar{a}_{qi} \end{aligned}\right\} i \neq p, q$$

$$a^*_{pp} = c^2 a_{pp} + 2csa_{pq} + s^2 a_{qq}$$

$$a^*_{qq} = s^2 a_{pp} - 2csa_{pq} + c^2 a_{qq}$$

$$a^*_{pq} = a^*_{qp}.$$

Theorem 12.2 (Property of R_{pq})

For $\theta = \begin{cases} \dfrac{1}{2}\tan^{-1}\left(\dfrac{2a_{pq}}{a_{pp} - a_{qq}}\right) & \text{if } a_{pp} \neq a_{qq} \\ \pm\dfrac{\pi}{4} & \text{if } a_{pp} = a_{qq} \quad \text{(take its sign to be that of } a_{pq}\text{)} \end{cases}$

the similarity transformation $A^* = R^T_{pq} A R_{pq}$ will annihilate the element a_{pq}, i.e. $a^*_{pq} = a^*_{qp} = 0$.

Proof To annihilate the element in position (p, q) through similarity transformation $R^T_{pq} A\ R_{pq}$, let $a^*_{pq}(\theta) = 0$, which, from (12.21), requires

$$\left(c^2 - s^2\right)a_{pq} + cs\left(a_{qq} - a_{pp}\right) = 0 \quad \Rightarrow \quad 2\cos 2\theta a_{pq} = \sin 2\theta\left(a_{pp} - a_{qq}\right)$$

$$\theta = \begin{cases} \dfrac{1}{2}\tan^{-1}\dfrac{2a_{pq}}{a_{pp} - a_{qq}} & \text{if} \quad a_{pp} \neq a_{qq} \\ \pm\dfrac{1}{4}\pi & \text{if} \quad a_{pp} = a_{qq} \end{cases}$$

The proof is complete. □

Jacobi's Iteration Process

The Jacobi process is an iteration process which sets one off-diagonal element (with maximum modulus) to zero each cycle using the plane rotation matrix (12.19) with the angle determined by theorem 12.2.

Cycle 1
- ◦ Locate element $a_{pq}(p<q)$ with maximum modulus in A then determine $R_1 = R_{pq}$;
- ◦ Perform similarity transformation $A^{(1)} = R_1^T A R_1$ such that $a_{pq}^1 = 0$.

Cycle 2
- ◦ Locate element $a_{pq}(p<q)$ with maximum modulus in $A^{(1)}$ and then determine $R_2 = R_{pq}$;
- ◦ Perform similarity transformation $A^{(2)} = R_2^T A^{(1)} R_2 = R_2^T R_1^T A R_1 R_2$ such that $a_{pq}^{(2)} = 0$.

If convergence is reached after k iterations, i.e. $\max\limits_{i\neq j}\left|a_{ij}^{(k)}\right| \leq Tol$, we have

$$A^{(k)} = R_k^T ... R_2^T R_1^T A R_1 R_2 ... R_k = D \qquad \text{or} \qquad \hat{R}^T A \hat{R} = D,$$

where $\hat{R} = R_1 R_2 \cdots R_k$.

As $\hat{R}^T \hat{R} = R_k^T \ldots R_2^T R_1^T R_1 R_2 \ldots R_k = I$, A and D are similar and thus based on theorem 12.1

a) Eigenvalues of A = eigenvalue of D, i.e. $\lambda_i = d_i$

b) Eigenvectors of A = columns of $\hat{R}$, i.e. $\mathbf{x}_i = \hat{R}_i$, where $\hat{R}_i$ is the ith column vector of $\hat{R} = R_1 R_2 ... R_k$.

Example 12.5 Find the complete eigen-solution of $A = \begin{pmatrix} 10 & 3 & 2 \\ 3 & 5 & 1 \\ 2 & 1 & 0 \end{pmatrix}$.

Use 5 decimal places and terminate iteration when the magnitude of each off-diagonal element of $A^{(k)}$ does not exceed 0.0001.

Solution

For simplicity, throughout the computation, any element with magnitude less than 0.0001 are set to zero. Use Jacobi's iteration process to reduce A to a diagonal matrix.

Step 1. The element of $A^{(0)}(A)$ with maximum modulus and above the diagonal is in position $(1, 2)$, i.e., $p = 1$, $q = 2$,

$$\therefore \quad R_1 = R_{12} = \begin{bmatrix} c & -s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where θ is determined by

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2a_{pq}}{a_{pp}-a_{qq}}\right) = \frac{1}{2}\tan^{-1}\left(\frac{2a_{12}}{a_{11}-a_{22}}\right) = \frac{1}{2}\tan^{-1}\left(\frac{2\times 3}{10-5}\right) = 0.43803\,,$$

$c = \cos\theta = 0.90559, \qquad s = \sin\theta = 0.42416.$

Hence,

$$R_1^T A = \begin{bmatrix} 0.90559 & 0.42416 & 0 \\ -0.42416 & 0.90559 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{pmatrix} 10 & 3 & 2 \\ 3 & 5 & 1 \\ 2 & 1 & 0 \end{pmatrix} = \begin{bmatrix} 10.32838 & 4.83757 & 2.23534 \\ -1.52483 & 3.25547 & 0.05727 \\ 2 & 1 & 0 \end{bmatrix}$$

$$A^{(1)} = R_1^T A R_1 = \begin{bmatrix} 10.32838 & 4.83757 & 2.23534 \\ -1.52483 & 3.25547 & 0.05727 \\ 2 & 1 & 0 \end{bmatrix}\begin{bmatrix} 0.90559 & -0.42416 & 0 \\ 0.42416 & 0.90559 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 11.40518 & 0 & 2.23534 \\ 0 & 3.59489 & 0.05727 \\ 2.23534 & 0.05727 & 0 \end{bmatrix}$$

Step 2 The element with maximum modulus in $A^{(1)}$ is in position (1, 3), so we choose

$p = 1, q = 3,$

$$\therefore \quad R_2 = R_{13} = \begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix},$$ where θ is determined by

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2a_{pq}}{a_{pp}-a_{qq}}\right) = \frac{1}{2}\tan^{-1}\left(\frac{2a_{13}}{a_{11}-a_{33}}\right) = \frac{1}{2}\tan^{-1}\left(\frac{2\times 2.23534}{11.40518-0}\right) = 0.18679$$

Hence, $$R_2 = \begin{bmatrix} 0.98261 & 0.42416 & -0.18571 \\ 0 & 1 & 0 \\ 0.18571 & 0 & 0.98261 \end{bmatrix}$$

$$A^{(2)} = R_2^T A^{(1)} R_2 = \begin{bmatrix} 11.82777 & 0.01064 & 0 \\ 0.01064 & 3.59489 & 0.05627 \\ 0 & 0.05627 & -0.42246 \end{bmatrix}$$

Remark: The 2nd iteration has destroyed the zero element in position (1, 2).

Step 3 $p = 2, q = 3, \therefore \; R_3 = R_{23} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.99990 & -0.01401 \\ 0 & 0.01401 & 0.99990 \end{bmatrix}$

where $\theta = \frac{1}{2}\tan^{-1}\left(\frac{2\times 0.05627}{3.59481-(-0.42246)}\right) = 0.01401$

$$A^{(3)} = R_3^T A^{(2)} R_3 = \begin{bmatrix} 11.82777 & 0.01064 & 0 \\ 0.01064 & 3.59567 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Step 4 $p = 1$, $q = 2$

$$\theta = 0.00129, \quad A^{(4)} = \begin{bmatrix} 11.82780 & 0 & 0 \\ 0 & 3.59567 & 0 \\ 0 & 0 & -0.42325 \end{bmatrix},$$

That is, after four iterations, A has been transformed to a diagonal matrix:

$$R_4^T R_3^T R_2^T R_1^T A R_1 R_2 R_3 R_4 = \hat{R}^T A \hat{R} = A^{(4)}$$

Hence,

1° eigenvalue of A = eigenvalue of $A^{(4)}$.

$\therefore \; \lambda_1 = 11.82780, \; \lambda_2 = 3.59567, \; \lambda_3 = -0.42325$

2° eigenvector of A = column of $\hat{R}$ where $\hat{R} = R_1 R_2 R_3 R_4$, i.e.,

$$\hat{R} = \begin{bmatrix} 0.90559 & -0.42416 & 0 \\ 0.42416 & 0.90559 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.98261 & 0.42416 & -0.18571 \\ 0 & 1 & 0 \\ 0.18571 & 0 & 0.98261 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.99990 & -0.01401 \\ 0 & 0.01401 & 0.99990 \end{bmatrix} \begin{bmatrix} \cdots \end{bmatrix}$$

$$= \begin{bmatrix} 0.88929 & -0.42762 & -0.16222 \\ 0.41795 & 0.90386 & -0.09145 \\ 0.18573 & 0.01226 & 0.98251 \end{bmatrix}.$$

12.6 Reduction of Symmetric Matrices intoTridiagonal Matrices-Given's Method

Given's method uses orthogonal rotation matrices to reduce A (symmetric) to a tridiagonal form whose eigenvalues can be determined by standard means.

The tridiagonal form is achieved in a finite number of steps

$$A \xrightarrow{\text{Rotation Matrices}} \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{pmatrix} \quad \text{i.e., } Q_k^T \ldots Q_2^T Q_1^T A Q_1 Q_2 \ldots Q_k = \text{Tridiagonal.}$$

Process and Formulae

The process is to use rotation matrices $R_{pq}(\theta)$ in transformation $R_{pq}^T A R_{pq}$ to remove elements other than tridiagonal elements one by one in a systematic order. In the process, we use $R_{pq}(\theta)$ to eliminate the element $A_{(p\text{-}1)q}$. The sequence of rotation matrices $R_{pq}(\theta)$ used is shown as follows by (p, q).

(2, 3), (2, 4), (2, 5), ... , (2, n) – Remove elements other than tridiagonal in row 1 at locations (1, 3), (1, 4), ...

(3, 4), (3, 5), ... , (3, n) – Remove elements other than tridiagonal in row 2 at locations (2, 4), (2, 5) , ...

(4, 5),... , (4, n) – Remove elements other than tridiagonal in row 3 at locations (3, 5), (3, 6), ...

Step 1 Let $A_0 = A$

Step 2 For each (p, q), calculate $A_{k+1} = R_{pq}^T A_k R_{pq}$ $(k = 0,1, \ldots)$.

To determine the value of θ such that $(A_{k+1})_{(p-1)q} = 0$, we first calculate $(A_{k+1})_{(p-1)q}(\theta)$ and then set it to zero and solve for θ.

$$A_{k+1} = R_{pq}^T A R_{pq} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ & \vdots & & \\ 0 \cdots & c & \cdots s \cdots & 0 \\ & \vdots & & \\ 0 & -s & c & 0 \\ & \vdots & & \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{11} & \cdots a_{1p} & \cdots a_{1q} & \cdots a_{1n} \\ \vdots & & & \\ a_{p1} & a_{pp} & a_{pq} & a_{pn} \\ \vdots & & & \\ a_{q1} & a_{qp} & a_{qq} & a_{qn} \\ \vdots & & & \\ a_{n1} & a_{np} & a_{nq} & a_{nn} \end{pmatrix} \begin{pmatrix} 1 & \cdots 0 & \cdots 0 & \cdots 0 \\ \vdots & & & \\ 0 & c & -s & 0 \\ \vdots & & & \\ 0 & s & c & 0 \\ \vdots & & & \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ & \vdots & & \\ 0 \cdots & c & \cdots s \cdots & 0 \\ & \vdots & & \\ 0 & -s & c & 0 \\ & \vdots & & \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{11} & \cdots (a_{1p}c + a_{1q}s) & \cdots (-a_{1p}s + a_{1q}c) & \cdots a_{1n} \\ \vdots & & & \\ a_{p1} & (a_{pp}c + a_{pq}s) & (-a_{pp}s + a_{pq}c) & a_{pn} \\ \vdots & & & \\ a_{q1} & (a_{qp}c + a_{qq}s) & (-a_{qp}s + a_{qq}c) & a_{qn} \\ \vdots & & & \\ a_{n1} & (a_{np}c + a_{nq}s) & (-a_{np}s + a_{nq}c) & a_{nn} \end{pmatrix}$$

Hence, we let

$$(A_{k+1})_{(p-1)q} = -a_{(p-1)p}\sin\theta + a_{(p-1)q}\cos\theta = 0$$

which yields $$\tan\theta = \frac{a_{(p-1)q}}{a_{(p-1)p}}$$

and so

$$\sin\theta = \frac{a_{(p-1)q}}{\sqrt{a^2_{(p-1)q} + a^2_{(p-1)p}}}, \quad \cos\theta = \frac{a_{(p-1)p}}{\sqrt{a^2_{(p-1)q} + a^2_{(p-1)p}}}.$$

This will not alter any elements whose row and column index is i or j where $i \neq p$, q and $j \neq p, q$. If a_{ip} and a_{iq} are both zero, then they are not altered by the transformation.

Example 2.6 Reduce $A = \begin{pmatrix} 12 & 3 & 4 & 12 \\ 3 & -12 & 0 & 3 \\ 4 & 0 & -12 & 4 \\ 12 & 3 & 4 & 12 \end{pmatrix}$ to tridiagonal form.

Solution

$$R_{23} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c & -s & 0 \\ 0 & s & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad s = \sin\theta = \frac{a_{13}}{\sqrt{a_{13}^2 + a_{12}^2}} = \frac{4}{5}, \quad c = \cos\theta = \frac{a_{12}}{\sqrt{a_{13}^2 + a_{12}^2}} = \frac{3}{5}$$

$$A_1 = R_{23}^T A R_{23} = \begin{pmatrix} 12 & 5 & 0 & 12 \\ 5 & -12 & 0 & 5 \\ 0 & 0 & -12 & 0 \\ 12 & 5 & 0 & 12 \end{pmatrix}.$$

Now applying R_{24}: $$s = \frac{a_{14}}{\sqrt{a_{14}^2 + a_{12}^2}} = \frac{12}{\sqrt{12^2 + 5^2}} = \frac{12}{13}$$

$$c = \frac{a_{12}}{\sqrt{a_{14}^2 + a_{12}^2}} = \frac{5}{\sqrt{12^2 + 5^2}} = \frac{5}{13},$$

then $$A_2 = R_{24}^T A R_{24} = \begin{pmatrix} 12 & 13 & 0 & 0 \\ 13 & 12 & 0 & 5 \\ 0 & 0 & -12 & 0 \\ 0 & 5 & 0 & -12 \end{pmatrix}.$$

Lastly, applying R_{34}: $$s = \frac{a_{24}}{\sqrt{a_{24}^2 + a_{23}^2}} = \frac{5}{\sqrt{5^2 + 0}} = 1$$

$$c = \frac{a_{23}}{5} = 0\,,$$

then $$A_3 = R_{34}^T A R_{34} = \begin{pmatrix} 12 & 13 & 0 & 0 \\ 13 & 12 & 5 & 0 \\ 0 & 5 & -12 & 0 \\ 0 & 0 & 0 & -12 \end{pmatrix}.$$

Note: -12 is an eigenvalue.

Eigenvalues of a Symmetric Tridiagonal Matrix

The method in this section can be used to find all the eigenvalues of a symmetric tridiagonal matrix. Let A be an $n \times n$ symmetric matrix, then there are n real eigenvalues. The eigenvalues are the zeros of the characteristic polynomial of A. To compute $f_n(\lambda) = \det(A - \lambda I)$, introduce the sequence

$$f_o(\lambda) = 1$$

$$f_k(\lambda) = \begin{vmatrix} a_1-\lambda & b_1 & & & & \\ b_1 & a_2-\lambda & b_2 & & & \\ & b_2 & a_3-\lambda & b_3 & & \\ & & b_3 & & \ddots & \\ & & & \ddots & \ddots & b_{k-1} \\ & & & & b_{k-1} & a_k-\lambda \end{vmatrix}$$

By direct calculation,

$$f_o(\lambda) = 1$$

$$f_1(\lambda) = a_1 - \lambda$$

$$f_2(\lambda) = (a_1-\lambda)(a_2-\lambda) - b_1^2 = (a_2-\lambda) f_1(\lambda) - b_1^2 f_0(\lambda)$$

$$f_3(\lambda) = (a_3-\lambda) f_2(\lambda) - b_2^2 f_1(\lambda)$$

$$f_k(\lambda) = (a_k-\lambda) f_{k-1}(\lambda) - b_{k-1}^2 f_{k-2}(\lambda), \quad k \ge 1$$

which can be proved by expanding the determinant in its last row using minors.

At this point, we can consider the problem as solved since $f_n(\lambda)$ is a polynomial and there are many polynomial root finding methods. But the sequence f_k has special properties that make it a Sturm sequence and these properties make it comparatively easy to isolate the eigenvalues of A.

Theorem 12.3 The number of agreements in sign of consecutive members of the sequence $f_k(p)$ $(k = 0, n)$ is the number of eigenvalues greater than p. When the term in the sequence equals zero, the sign should be taken to be the opposite of the sign of the previous term.

Use the above theorem, we can find the eigenvalues of A.

Example 12.7 Given $A = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}$, find an interval, of length less than or equal 0.5, containing the smallest eigenvalue.

Solution

Gerschgorin discs $|\lambda\text{-}2| \le 2$, i.e. $0 \le \lambda \le 4$. Thus all eigenvalues (4 real eigenvalues) lie in [0, 4].

Sturm Sequence:

$$f_o(p) = 1$$
$$f_1(p) = a_1 - p = 2 - p$$
$$f_2(p) = (a_2 - p)f_1(p) - b_1^2 f_0(p) = (2-p)f_1(p) - 1$$
$$f_3(p) = (a_3 - p)f_2(p) - b_2^2 f_1(p) = (2-p)f_2(p) - f_1(p)$$
$$f_4(p) = (a_4 - p)f_3(p) - b_3^2 f_2(p) = (2-p)f_3(p) - f_2(p)$$

Range	p	f_0	f_1	f_2	f_3	f_4	# agreements	Remark
[0, 4]	2	1 (+)	0 (–)	–1 (–)	0 (+)	1 (+)	2	2 eigenvalues > 2
[0, 2]	1	1 (+)	1 (+)	0 (–)	–1 (–)	–1 (–)	3	3 eigenvalues > 1
[0, 1]	0.5	1 (+)	1.5 (+)	1.25 (+)	0.38 (+)	–.69 (–)	3	3 eigenvalues > 0.5

Hence, the interval containing the smallest eigenvalue is [0, 0.5].

– To refine the above estimate, continue applying the bisection method to [0, 0.5] (very slow process but stable).
– Although all roots of a symmetric tridiagonal matrix may be found in this way, it is generally faster to use other algorithms. With large matrices, we usually do not want all of the eigenvalues and then this method is ok.
– If we want only specific roots, for example, those in [1,3], then it is easy to locate them with this technique.

12.7 Computing Eigenvalues and Eigenvectors Using Maple/MATLAB

Computing Eigenvalue and Eigenvectors Using Maple

Maple povides functions “`eigenvals()`” and “`eigenvectors()`” to compute eigenvalues and eigenvectors of matrices.

Example 1 Given $A = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix}$. Find eigenvalues and eigenvectors of A.

```
> with(linalg);
> A:=matrix(3,3,[1, 0, 2, 0, 1, -1, -1, 1, 1]);
> e:=eigenvals(A);
> v:=eigenvectors(A);
```

gives

```
e := 1, 1+I√3 , 1-I√3
v  := [1+I√3 ,1,{[-2 1 - I√3 ]}], [1-I√3 ,1,{[-2 1  I√3 ]}],[1,1,{[1 1 0]}]
```

Note: the result returned by the function **eigenvectors** is a sequence of list of the form $[e_i, m_i, \{[\mathbf{v}_{1i}],[\mathbf{v}_{2i}],...\}]$ in which e_i are the eigenvalues, m_i their algebraic multiplicities and $\{[\mathbf{v}_{1i}],[\mathbf{v}_{2i}],...\}$ is a set of basis vectors for the eigenspace corresponding to e_i.

Computing Eigenvalue and Eigenvectors Using MATLAB

The MATLAB function "`eig()`" computes all the eigenvalues and their corresponding eigenvectors of a given matrix. The syntax is

```
[V, L] = eig(A)
e =  eig(A)
```

where V : (output) modal matrix ($\hat{R}$ as defined in section 12.5) composed of eigenvectors of A;
L : (output) diagonal matrix whose diagonal elements are eigenvalues of A;
e : (output) eigenvalues.

Example Given $A = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix}$. Find eigenvalues and eigenvectors of A.

```
>> A=[1 0 2; 0 1 -1; -1 1 1];
>> [V,L] = eig(A)
```

yields

```
V =

    0.7071              0.7071              0.7071
  -0.3536+0.0000i     -0.3536+0.0000i      0.7071
   0.0000+0.6124i      0.0000-0.6124i      0.0000

L =

  1.0000+1.7321i          0                 0
        0            1.0000-1.7321i         0
        0                 0               1.0000
```

The diagonal matrix L give 3 eigenvalues of A: $\lambda_1 = 1+1.7321i,\ \lambda_2 = 1-1.7321i,\ \lambda_3 = 1$; while the 3 columns of the matrix U are the eigenvectors corresponding respectively to λ_1, λ_2 and λ_3.

If only the eigenvalues of A are to be computed, one could use the following command

```
>> e = eig(A)
```

which returns the eigenvalues of A,

```
e =
   1.0000 + 1.7321i
   1.0000 - 1.7321i
   1.0000
```

EXERCISES 12

Q12. 1 Let λ be an eigenvalue of the $n \times n$ matrix A and $\mathbf{x} \neq 0$ be associated eigenvector. Show that

a) If A^{-1}exists, then $1/\lambda$ is an eigenvalue of A^{-1} with eigenvector $\mathbf{x}$.

b) $\lambda - p$ is an eigenvalue of $A - p$ with eigenvector $\mathbf{x}$.

c) If $(A - \alpha I)^{-1}$ exists ($\alpha \neq \lambda$), then $1/(\lambda - \alpha)$ is an eigenvalue of $(A - \alpha I)^{-1}$ with eigenvector $\mathbf{x}$.

Q12. 2 Describe in your words how to find

a) a bound for the eigenvalue using Gerschgorin's discs theorem;

b) the dominant eigenvalue and its associated eigenvector using the power method;

c) the smallest eigenvalue and its associated eigenvector using the power method (see Q12.1-b);

d) the eigenvalue closest to a given value p and its associated eigenvector using the power method (see Q12.1-c).

Q12. 3 Use Gershgorin's Theorem to find bounds for the eigenvalues of

$$\text{(a)} \begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & -1 \\ -1 & -1 & 2 \end{pmatrix} \quad \text{(b)} \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ -1 & -1 & 4 \end{pmatrix} \quad \text{(c)} \begin{pmatrix} 3 & 2 & 1 \\ 2 & 3 & 0 \\ 1 & 0 & 3 \end{pmatrix}$$

(Ans: (a) (*Ans*: (a) $|\lambda| \le 4$; (b) $2 \le \lambda \le 6$; (c) $0 \le \lambda \le 6$)

Q12. 4 Find the larger eigenvalue and the corresponding eigenvector by the power method:

$$\text{(a)} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad \text{(b)} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Q12. 5 Find the dominant eigenvalue and its corresponding eigenvector of $A = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 3 \\ 1 & 1 & 1 \end{pmatrix}$

(Ans: $\lambda = 3.49086$, $x^T = [1, 0.9362, 0.7733]$)

Q12. 6 Invert the matrices in Q12.4(b) and Q12.5; then use the power method to get the smallest eigenvalue of the original matrices. Repeat, except avoiding the inversion but using LU decomposition of the matrix. Compare the effort in the two cases.

Q12. 7 Find all the eigenvalues of the matrix in Q12.5. Apply INVERSE power iteration method to matrix $A - kI$ with appropriate values of k.
(Ans: 3.49086, –0.83424, 0.34338)

Q12. 8 After finding the dominant eigenvalue in Q12.5, subtract that value from each of the diagonal elements and use the power method. Compare the value obtained with the second largest eigenvalue as determined in Q12.7.

Q12. 9 Find the larger eigenvalue and corresponding eigenvector by the inverse power method: $\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$

Q12. 10 Describe in your words

a) How to obtain a complete e.solution using Jacobi's method;
b) How to reduce a symmetric matrix to a tridiagonal form;
c) How to calculate the eigenvalue of a tridiagonal matrix.

Q12. 11 Show that if B is obtained from A via a similarity transformation, then A and B have the same eigenvalues. Find a relationship between the eigenvectors of A and B.

Q12. 12 Use the Jacobi method to determine all the eigenvalues of the matrix $A = \begin{pmatrix} 3 & 0 & 1 \\ 0 & 3 & 2 \\ 1 & 2 & 3 \end{pmatrix}$.

Terminate iteration when all the off-diagonal elements have modulus not exceeding 0.001. Compare your results obtained from direct method (5.23607, 3, 0.76393).

Q12. 13 Apply Given's method to reduce the following matrix to tri-diagonal form $\begin{pmatrix} 2 & -1 & 4 \\ -1 & 2 & -1 \\ 4 & -1 & 1 \end{pmatrix}$

Q12. 14 Reduce A to tri-diagonal form C, where $A = \begin{pmatrix} 1 & \sqrt{2} & \sqrt{2} \\ \sqrt{2} & -\sqrt{2} & -1 \\ \sqrt{2} & -1 & \sqrt{2} \end{pmatrix}$.

If $\left(-\frac{1}{\sqrt{2}},\ 0,\ 1\right)^T$ is an eigenvector of C, write down the corresponding eigenvector of A. Find the number of non-negative eigenvalues of A.

Q12. 15 Use Gershgorin's discs to find bounds on the eigenvalues of $A = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 2 & 1 & 2 & 0 \\ 0 & 2 & 1 & 2 \\ 0 & 0 & 2 & 1 \end{pmatrix}$

Find the Sturm sequence for evaluating the eigenvalues of A. Determine the eigenvalue of the largest modulus using the Sturm sequence (to two decimal places).

Q12. 16 The eigenvalues of the tri-diagonal matrix $\begin{pmatrix} 1 & 2 & 0 \\ 2 & -1 & \sqrt{2} \\ 0 & \sqrt{2} & 1 \end{pmatrix}$ lie in the interval $(-a,\ a)$.

(a) Find the smallest value of a.

(b) Using $a = 6$ for simplicity, find an interval containing each eigenvalue by use of Sturm sequences.

Q12. 17 Reduce the matrix $\begin{pmatrix} 1 & -1 & 0 & 1 \\ -1 & 2 & -2 & 5 \\ 0 & -2 & 1 & 3 \\ 1 & 5 & 3 & 3 \end{pmatrix}$ to tri-diagonal form using Given's method.

PROGRAMMING

Q12. 18 Write a program to find the dominant eigenvalue and its associated eigenvector using the power method. Hence find the dominant eigen- solution of the matrix

$$\begin{pmatrix} 3 & 0 & 1 \\ 0 & 3 & 2 \\ 1 & 2 & 3 \end{pmatrix}$$

(Ans: 5.23607, [0.44721, 0.89443, 1])

Hint: **1)** Write a subroutine **ReadDat** to read data
2) Write a subroutine to implement the Power method

SUBROUTINE POWER(N, A, Z, Tol, MaxNit, NewMu, Ierr)

N	=	Before entry, N must be set to the dimension of A.
Z	=	Before entry, **z** must be set to $z^{(0)}$; On exit it contain the eigenvector $\mathbf{x}_1$.
Tol	=	Before entry, Tol must be set to a positive tolerance for controlling the error in the approx.
MaxNit	=	Before entry, MaxNit must be set to the Maximum number of iterations allowed.
NewMu	=	On exit, it contains the eigenvalue with maximum modulus, i.e., λ_1.
Ierr	=	On exit, Ierr = 0 indicates that the process converges to λ_1 .

Algorithm:

```
Set  Niter = 0
     NewMu = 0
Do While (Niter < MaxNit)
     Set  Niter = Niter+1
          OldMu = NewMu
     Call  FindYmult (N, A, Z, Y)              – $\mathbf{y}^{(k)} = A\mathbf{z}^{(k-1)}$
     Call  FindMu (N, Y, NewMu, NewI)          – Find $\mu_k$ from $\mathbf{y}^{(k)}$
     Call  FindZ (Y, NewMu, Z)                 – $\mathbf{z}^{(k)} = \mathbf{y}^{(k)}/\mu_k$
     If ( $|NewMu - OldMu| \le Tol * |OldMu|$ and $NewI = OldI$ ) Then
          Set  Ierr = 0
          Return
     EndIf
EndDo
Ierr = 1
Return
```

Note: Test for convergence:

$$\begin{cases}\text{The size of } \lambda_1 \text{ converges} \\ \text{The element with max. modulus occurs at the same position of the e.vector at any two consecutive iteration.}\end{cases}$$

3) Write a main program to control the process.

Q12. 19 Write a program to find the eigenvalue closest to p and the associated eigenvector. Hence find the eigenvalue closest to 2.5 and its associated eigenvector for the matrix in Q12.18. (Ans: 3, $[1, -0.5, 0]$).

Hint: A program plan for the inverse power method would be very similar to that for the direct power method except that the method used to evaluate $\mathbf{y}^{(k)}$ is different: The P.M. use matrix-vector multiplication whereas the I.P.M. requires the solution of a linear system $B\mathbf{y}^{(k)} = \mathbf{z}^{(k-1)}$ where $B = A - pI$.

Modification:

SUBROUTINE POWER(N, A, p, Z, Tol, MaxNit, NewMu, Ierr)

1) Form $B = A - pI$ and then factor B into $L\ U$ matrices before executing the while loop.
2) Replace the Subroutine ***FindYmult*** by a forward-backward substitution subroutine **FindYsubst** .
3) When convergence is reached, calculate (before exit) the eigenvalue closest to p by
   $NewMu = p + 1 / NewMu \quad (\lambda^* = p + 1 / \mu_k)$

Q12.20 Write a program to find the complete eigen solution of a $n \times n$ matrix. Hence find the complete eigen solution for the matrix in Q12.18.

Q12.21 Find the eigenvalues and eigenvectors of the matrices in Q12.12 and Q12.18 using Maple/Matlab built-in functions.

Chapter 13 Solution of Boundary Value Problems for Partial Differential Equations

The mathematical formulation of most real world problems in science and industry usually leads to a *boundary value problem*: a differential equation (or a set of differential equations) subject to certain initial and boundary conditions.

For example, the transient temperature field in a bounded domain Ω can be described by

$$\rho c \frac{\partial T}{\partial t} = k\nabla^2 T + Q(\mathbf{x}), \qquad \mathbf{x} \in \Omega$$

$$\frac{\partial T}{\partial n} = -h(T - T_\infty), \qquad \mathbf{x} \in \partial\Omega$$

$$T(0,\ \mathbf{x}) = T_0(\mathbf{x}), \qquad \mathbf{x} \in \Omega$$

where $Q(\mathbf{x})$ denotes heat source and ρ, c, k and h are constants.

In this chapter, we are concerned with

1) Classification of partial differential equations (PDEs).
2) Type of boundary conditions (BC).
3) An overview of the methods for solving boundary value problems (BVP).

13.1 Classification of Partial Differential Equations (2nd order)

Definition: A differential equation is said to be

linear – if it is a linear equation of the unknown function and its derivatives,

eg. $au_{xx} + bu_{yy} + cu_x + du = Q(x,y)$,

where a, b, c and d are constants, and Q is a function of x and y;

quasi-linear – if all the highest derivative terms are linear but some of the lower order derivatives are non-linear,

eg. $au_{xx} + bu_x^2 = f(x, y, u)$;

non-linear – if the equation is neither linear nor quasi-linear,

eg. $u_{xx} + 2u_{xy}^2 + bu = Q(x, y)$.

Most partial differential equations (PDE) arising from real world problems are second order. In this chapter we will focus only on second order quasi-linear equations which has the general form

$$au_{xx} + bu_{xy} + cu_{yy} + h(x, y, u, u_x, u_y) = 0,$$

and can be classified into three categories according to the value of $b^2 - 4ac$, namely

elliptic	if	$b^2 - 4ac < 0$;
parabolic	if	$b^2 - 4ac = 0$;
hyperbolic	if	$b^2 - 4ac > 0$.

Remark : If a, b and c are functions of x, y and u, the equation may change its type from one region to the other in the computation domain.

Example 13.1

Poisson equation $\nabla^2 u = \sigma$ is *elliptic* ($a = c = 1, b = 0$).

Diffusion equation $\dfrac{\partial u}{\partial t} = \dfrac{\partial^2 u}{\partial x^2}$ is *parabolic* ($a = 1\ b = c = 0$).

Wave equation $\dfrac{\partial^2 u}{\partial t^2} = \alpha^2 \dfrac{\partial^2 u}{\partial x^2}$ is *hyperbolic* ($a = \alpha^2$, $b = 0$, $c = -1$).

Example 13.2

Consider $\left[1 - M(x, y)\right]\dfrac{\partial^2 \phi}{\partial x^2} + \dfrac{\partial^2 \phi}{\partial y^2} = 0$.

The equation is	elliptic	if	$M(x, y) < 1$,
	parabolic	if	$M(x, y) = 1$,
	hyperbolic	if	$M(x, y) > 1$.

13.2 Boundary and Initial Conditions

If we do not distinguish between time and space as independent variables, an initial condition can also be regarded as a boundary condition.

For a real world problem, usually, we know the value of the unknown function and/or its derivatives on part of the boundary $\partial\Omega$. As the solution must satisfy the conditions on the boundary, we have to solve the partial differential equation in Ω subject to the boundary conditions on $\partial\Omega$.

Boundary conditions are usually of the following types:

(i) Dirichlet type (also called essential boundary condition)

$$u = \hat{u} \quad \text{on } \partial\Omega.$$

(ii) Neumann type (also called natural boundary condition)

$$\frac{\partial u}{\partial n} = \hat{\sigma} \quad \text{on } \partial\Omega\,.$$

(iii) Robin type (mixed or general boundary condition)

$$\alpha \frac{\partial u}{\partial n} + ku = f \quad (\alpha \neq 0,\ k \neq 0) \quad \text{on } \partial\Omega\,.$$

13.3 Methods for Solving Boundary Value Problems (BVPs) – an Overview

In general, a BVP can be written as

$$\begin{cases} L(u) = f & \text{on } \Omega \\ B(u) = g & \text{on } \partial\Omega \end{cases} \tag{13.1}$$

where f, g are known functions,
L is a linear or nonlinear differential operator,
B is a boundary operator.

The general problem is to find the unknown function u that satisfies the PDE in Ω and the boundary condition on $\partial\Omega$. There are many alternative approaches for solving linear and nonlinear boundary value problems, and they range from completely analytical to completely numerical and can be classified as

i) Direct Integration methods

eg. Separation of variables, similarity solutions,
Fourier and Laplace transformations.

ii) Approximate Solution methods

eg. Perturbation, Power series, Probability schemes (Monte Carlo)
The method of characteristics for hyperbolic equations
Finite difference technique,
Ritz method
Finite element method.
Boundary element method.

Remarks

a) Only for very simple problems, it is possible to obtain an exact solution by direct integration of the differential equation.

b) The Power series method is powerful, but since the method requires generation of a coefficient for each term in the series, it is relatively tedious.

c) The perturbation method is applicable primarily when the nonlinear terms in the equation are small in relation to the linear terms.

d) The probability schemes (Monte Carlo Method) are used for obtaining a statistical estimate of a desired quantity by random sampling. These methods work best when the desired quantity is a statistical parameter & sampling is done from a selective population.

e) With the advent of high-speed computers, it appears that the three currently outstanding methods for obtaining approximate solutions of high accuracy are the finite difference method, the finite element method and the boundary element method. The finite difference method usually is only applicable to problems with simple geometry. The boundary element method is a more efficient and accurate method, which usually reduces the dimensionality of the problem by one. However, the application of the boundary element method requires a singular solution to the problem, which limits its application. The finite element method is a more general and versatile method. In principle, any problem, which can be solved by the finite difference method or the boundary element method, can also be solved by the finite element method.

In this chapter, we will study briefly the finite difference method.

13.4 Finite Difference Method (FDM) for PDEs

To solve the BVP (13.1) with $L = \nabla^2$ using the FDM, we need to perform the following work

1) Discretize Ω into a mesh of discrete points called nodes.
2) Approximate all derivatives using the values of the unknown function at the nodes, and thus, the differential equation is approximated by a system of algebraic equations with the nodal values of the unknown function as basic unknowns.
3) Solve the linear or nonlinear system of algebraic equations.

In the following, we first show how to discretize Ω and approximate derivatives, and then demonstrate how to convert a boundary value problem into a system of algebraic equations and solve as such through an example.

Discretization and Derivative Approximation

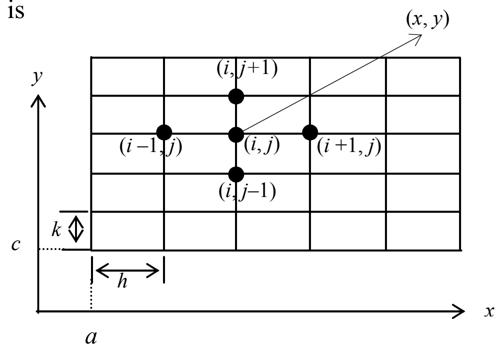
Firstly, the domain Ω, say $(a,b)\times(c, d)$ in the xy- plane, is subdivided into a set of equal rectangles of sides

$$\delta x = h=(b-a)/n, \quad \delta y = k=(d-c)/m.$$

Obviously, in the coordinate system chosen

$$(x_i, y_j) = (a+ih, c+jk), \quad (i=0,n; j=0, m).$$

Thus for convenience in presentation, we use (i,j) to denote the position (x_i,y_j) and $u_{i,j}$ to represent $u(x_i,y_j)$, namely



$$(x_i, y_j) = ij,$$
$$u(x_i, y_j) = u_{i,j}.$$

Now, we are ready to derive formulae for the approximation of derivatives. From Taylor's Theorem, we have

$$u\left(x_i + h,\ y_j\right) = u_{i+1,j} = u_{i,j} + hu_x + \frac{h^2}{2}u_{xx} + \ldots \tag{13.2}$$

$$u\left(x_i - h,\ y_j\right) = u_{i-1,j} = u_{i,j} - hu_x + \frac{h^2}{2}u_{xx} - \ldots \tag{13.3}$$

where u_x and u_{xx} are all evaluated at (x_i,y_j).

(13.2) – (13.3) and then rearranging yields

$$\left(\frac{\partial u}{\partial x}\right)_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2h} + O\left(h^2\right).$$

(13.2) + (13.3) and then rearranging yields

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j} = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + O\left(h^2\right).$$

In the same way , we have

$$\left(\frac{\partial^2 u}{\partial y^2}\right)_{i,j} = \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{k^2} + O\left(k^2\right).$$

By using the Taylor theorem in 2 variables, we have

$$\left(\frac{\partial^2 u}{\partial x \partial y}\right)_{i,j} = \frac{u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}}{4hk} + O\left(h^2 + k^2\right).$$

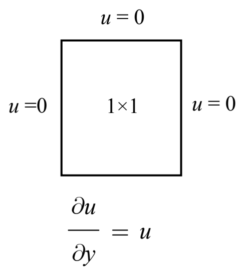
If we choose $h = k$, then

$$\nabla^2 u_{i,j} = \frac{1}{h^2}\left[u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}\right] + O\left(h^2\right),$$

which can be graphically displayed by

$$\nabla^2 u_{i,j} = \begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix} u_{i,j}$$

which is called a 5- points difference scheme.



Elliptic Partial Differential Equations

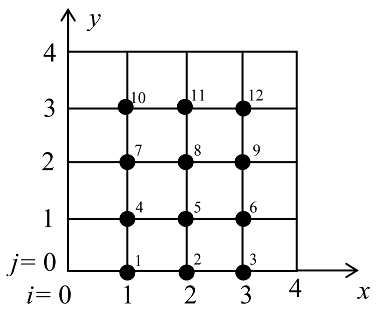
Consider $\nabla^2 u = f(x, y)$ on a square with BCs as shown. Firstly, we discretize the system. Let $N = 4$, $h = 1/4$, then the domain is discretized into a mesh with 5×5 grid points as shown. The nodes where u is to be determined are only those points

$(i,j) : i = 1$ to 3, $j = 0$ to 3.

At each of these nodes, we can set up an equation

$$\nabla^2 u_{i,j} = f_{i,j}$$

and thus the total number of equations equals to the number of unknowns, i.e., 12.



Now, we consider construction of the equations for the determination of $u_{i,j}\ (i = 1,3,\ j = 0,3)$.

By using the 5-point finite difference approximation

$$\nabla^2 u_{i,j} = \frac{1}{h^2}\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix} u_{i,j}\,,$$

the given PDE becomes

$$\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix} u_{i,j} = h^2 f_{i,j} \tag{13.4}$$

or

$$u_{i,j-1} + u_{i-1,j} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1} = h^2 f_{i,j}\ \ (i = 1 \text{ to } 3,\ j = 0 \text{ to } 3) \tag{13.5}$$

For each grid point, where u is to be determined, we have one equation. Usually we would construct these equations row by row . For j=0, as the above equation involves $u_{i,-1}$, we need to approximate $u_{i,-1}$ by using the boundary condition and this will be considered later; while for i, j = 1, 2, 3, we can immediately obtain the following nine equations

$$\begin{array}{lr|cccccccccccc|}
j{=}0 & \text{Col } i=1 & \\
 & 2 & \multicolumn{12}{c|}{\text{equations for } j{=}0 \text{ are to be constructed later}} \\
 & 3 & \\
j{=}1 & i=1 & 1 & & & -4 & 1 & & 1 & & & & & \\
 & 2 & & 1 & & 1 & -4 & 1 & & 1 & & & & \\
 & 3 & & & 1 & & 1 & -4 & & & 1 & & & \\
j{=}2 & i=1 & & & & 1 & & & -4 & 1 & & 1 & & \\
 & 2 & & & & & 1 & & 1 & -4 & 1 & & 1 & \\
 & 3 & & & & & & 1 & & 1 & -4 & & & 1 \\
j{=}3 & i=1 & & & & & & & 1 & & & -4 & 1 & \\
 & 2 & & & & & & & & 1 & & 1 & -4 & 1 \\
 & 3 & & & & & & & & & 1 & & 1 & -4
\end{array}
\begin{bmatrix} u_{10} \\ u_{20} \\ u_{30} \\ u_{11} \\ u_{21} \\ u_{31} \\ u_{12} \\ u_{22} \\ u_{32} \\ u_{13} \\ u_{23} \\ u_{33} \end{bmatrix}
= h^2 \begin{bmatrix} f_{10} \\ f_{20} \\ f_{30} \\ f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} \tag{13.6}$$

or in matrix form

$$\left[\begin{array}{c|c|c|c} ? & ? & ? & ? \\ \hline I & B & I & \\ \hline & I & B & I \\ \hline & & I & B \end{array}\right] \mathbf{u} = \mathbf{F} \tag{13.7}$$

where

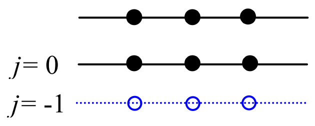
$$B = \begin{bmatrix} -4 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & -4 \end{bmatrix}.$$

Remark 1 (Dirichlet boundary condition): On $x = 0$ ($i = 0$), $x = 1$ ($i = 4$) and $y = 1$ ($j = 4$), $u = 0$. So the BCs never enter the calculation. If $u = \hat{u}$ on these sides then one must move these known values to the right hand side of the equations.

Remark 2 (Neumann type boundary condition): For $y = 0$ ($j = 0$), (13.5) becomes

$$u_{i,\,-1} + u_{i-1,\,0} - 4u_{i,\,0} + u_{i+1,\,0} + u_{i,\,1} = h^2 f_{i,\,0} \tag{13.8}$$

Obviously, $u_{i,-1}$ is not defined as the point $(i, -1)$ is outside the region Ω considered. So we need to eliminate the term $u_{i,-1}$ using the Neumann boundary condition, as given below. As we know the value of $\dfrac{\partial u}{\partial y}$ on $y = 0$, we introduce a fictitious set of grid points $(i, -1)$ $(i = 1, 2, 3)$ as shown below.



Then at the boundary point $(i, 0)$, we can approximate the BC by

$$\left.\frac{\partial u}{\partial y}\right|_{j=0} = \frac{u_{i,1} - u_{i,-1}}{2h} = u_{i,0} \tag{13.9}$$

which gives

$$u_{i,-1} = u_{i,1} - 2hu_{i,0} \quad . \tag{13.10}$$

Substituting (13.10) into (13.8) yields

$$u_{i-1,\,0} - (4+2h)u_{i,\,0} + u_{i+1,\,0} + 2u_{i,\,1} = h^2 f_{i,\,0} \quad (i = 1, 2, 3) \tag{13.11}$$

or

$$\left[\begin{array}{ccc} & 0 & \\ 1 & -(4+2h) & 1 \\ & 2 & \end{array}\right] u_{i,\,0} = h^2 f_{i,\,0}\,, \quad (i\text{=}1,2,3).$$

For $j = 1$ and $i = 1, 2, 3$, we obtain

$$\left[\begin{array}{ccc|ccc|ccc|ccc} -(4+2h) & 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -(4+2h) & 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -(4+2h) & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \mathbf{u} = \mathrm{F}$$

From the above and equation (13.6), we have

$$\begin{bmatrix} B-2hI & 2I & \mathbf{0} & \mathbf{0} \\ I & B & I & \mathbf{0} \\ \mathbf{0} & I & B & I \\ \mathbf{0} & \mathbf{0} & I & B \end{bmatrix} \mathbf{u} = \mathrm{F} ,$$

which can then be solved by a direct method or an iterative method for linear systems of equations.

13.5 Improvement of the Accuracy of Solution

One way to improve accuracy is by reducing the value of k and h, i.e., through mesh refinement. Here we introduce two other methods.

Deferred Approach to the Limit

Denote u = true solution,
u_h = finite difference solution,

then $u = u_h$ + error.

For example, let u_h be the solution of the Laplace equation by using the 5 point scheme, then

$$u = u_h + Ah^2 .$$

Based on the above formulae, to accelerate the convergence of numerical solutions, we could use the so-called deferred approach to the limit suggested by Richardson.

Let u_1= approximation obtained by using mesh size h, then

$$u = u_1 + Ah^p . \tag{13.12}$$

If the mesh size h is reduced from h to $h/2$, then

$$u = u_2 + A(h/2)^p. \tag{13.13}$$

where u_2 is the approximation obtained with $h/2$. from (13.12) and (13.13), by eliminating p, we obtain

$$u = \frac{2^p u_2 - u_1}{2^p - 1}. \tag{13.14}$$

For example, for $p = 2$

$$u = \frac{4u_2 - u_1}{3} .$$

If p is unknown, it can be estimated from three approximate solutions at the same point obtained by using different mesh sizes. Let u_1 , u_2 and u_3 are approximations at the same point obtained respectively using step size h_1 , h_2 and h_3 where $h_3 = \frac{1}{2}h_2 = \frac{1}{4}h_1$, then

$$u = u_1 + Ah_1^p,$$
$$u = u_2 + A\left(\tfrac{1}{2}\right)^p h_1^p,$$
$$u = u_3 + A\left(\tfrac{1}{4}\right)^p h_1^p,$$

from which, we have

$$\mathrm{u}_2 - u_1 + Ah_1^p\left[\left(\tfrac{1}{2}\right)^p - 1\right] = 0,$$
$$\mathrm{u}_3 - u_2 + Ah_1^p\left(\tfrac{1}{2}\right)^p\left[\left(\tfrac{1}{2}\right)^p - 1\right] = 0.$$

Hence,

$$2^p = \frac{\mathrm{u}_2 - u_1}{\mathrm{u}_3 - u_2}.$$

Deferred Correction Method

From Taylor's Theorem

$$u(x+h,\, y) = u_{i,j} + hu_x + \frac{h^2}{2}u_{xx} + \ldots\ldots + \ldots\ldots = e^{hD}u \tag{13.15}$$

$$u(x-h,\, y) = u_{i,j} - hu_x + \frac{h^2}{2}u_{xx} - \ldots\ldots\ldots\ldots\ldots = e^{-hD}u \tag{13.16}$$

where $D = \frac{\partial}{\partial x}$.

Using the central difference formula at (x, y)

$$\delta_x u_{i,j} = u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j} = \left(e^{\frac{h}{2}D} - e^{-\frac{h}{2}D}\right)u_{i,j} = 2\sinh\frac{hD}{2}u_{i,j} \tag{13.17}$$

$$\therefore\quad \delta_x = 2\sinh\frac{hD}{2} \tag{13.18}$$

$$D \;=\; \frac{2}{h}\sinh^{-1}\frac{\delta_x}{2} = \frac{2}{h}\left[\frac{\delta_x}{2} - \frac{1}{3!}\frac{\delta_x^3}{2^3} + \frac{3^2}{5!}\frac{\delta_x^5}{2^5} - \ldots\ldots\ldots\ldots\right]$$

$$D^2 = \frac{4}{h^2}\left[\sinh^{-1}\frac{\delta_x}{2}\right]^2 = \frac{1}{h^2}\left[\delta_x^2 - \frac{1}{12}\delta_x^4\ldots\ldots\ldots\ldots\right] \tag{13.19}$$

Thus for $\nabla^2 u = F(x)$, as

$$h^2 u_{xx} = \left(h^2\frac{\partial^2 u}{\partial y^2}\right) = \left(\delta_x^2 - \frac{1}{12}\delta_x^4 + \ldots\right)u,$$
$$h^2 u_{yy} = \left(h^2\frac{\partial^2 u}{\partial y^2}\right) = \left(\delta_y^2 - \frac{1}{12}\delta_y^4 + \ldots\right)u\;,$$

we get

$$\left(\delta_x^2+\delta_y^2\right)u_{i,j}-\frac{1}{12}\left(\delta_x^4+\delta_y^4\right)u_{i,j}\approx h^2F_{i,j}\,, \tag{13.20}$$

or
$$\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}u_{i,j}=h^2F_{i,j}+\frac{1}{12}\begin{bmatrix} & & 1 & & \\ & & -4 & & \\ 1 & -4 & 12 & -4 & 1 \\ & & -4 & & \\ & & 1 & & \end{bmatrix}u_{i,j}. \tag{13.21}$$

The initial approximation is given by solving

$$\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}u_{i,j}=h^2F_{i,j}\,, \tag{13.22}$$

which is, of course, the usual five-points formula.

The initial approximation is then used to calculate the second term in the right hand side of (13.21), and then the system (13.21) is solved for an improved solution.

13.6 Solution of PDE Boundary Value Problems using Maple/MATLAB

Solution of PDE Boundary Value Problems using Maple

The Maple function "**pdsolve**()" can be used to find solution of time related boundary value problems for partial differential equations. The syntax is

```
pdsolve(PDEs,conditions,numeric,vars,options)
```

where PDEs :	a single or a set of time-dependent partial differential equations in two independent variables
conditions :	a set of initial and boundary conditions
numeric :	keyword indicating a numerical solution is to be obtained
vars :	(optional) a dependent variable or a set of dependent variables for PDEs
options :	(optional) equations of the form keyword = value where keyword is one of 'indepvars', 'time', 'range', 'spacestep', 'timestep', 'bcopts', 'optimize', 'errorest', 'errortype', 'abstol', 'mintimestep' or 'maxtimestep'; specify options for the problem and its solution .

Example Solve
$$\begin{cases} \dfrac{\partial}{\partial t}u(x,t)=\dfrac{1}{10}\left(\dfrac{\partial^2}{\partial x^2}u(x,t)\right) \\ u(x,0)=1,\quad u(0,t)=0,\quad \dfrac{\partial}{\partial x}u(1,t)=0 \end{cases}$$

which is a heat conduction equation with the left end fixed at a constant temperature and the right end insulated.

Firstly, we define the partial differential equations by using the following Maple statement

```
> PDE := diff(u(x,t),t)=1/10*diff(u(x,t),x,x);
```

which produces

$$\boxed{PDE := \frac{\partial}{\partial t}u(x,t) = \frac{1}{10}\left(\frac{\partial^2}{\partial x^2}u(x,t)\right)}$$

Then we define the initial and boundary conditions as follows:

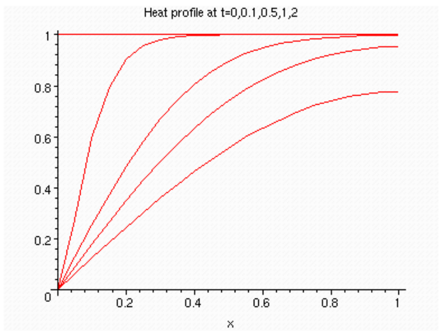
```
> IBC := {u(x,0)=1, u(0,t)=0, D[1](u)(1,t)=0};
```

which yields

$$\boxed{IBC := \{u(x,0)=1,\ u(0,t)=0,\ (D_1(u))(1,t)=0\}}$$

We now solve the above boundary value problem using function "**pdsolve**()" and plot the heat profiles at *t*=0, 0.1, 0.5, 1 and 2.

```
> pds := pdsolve(PDE, IBC, numeric) ;
> p1 := pds:-plot(t=0):
> p2 := pds:-plot(t=1/10):
> p3 := pds:-plot(t=1/2):
> p4 := pds:-plot(t=1):
> p5 := pds:-plot(t=2):
> plots[display]({p1,p2,p3,p4,p5},title=`Heat
profile at t=0,0.1,0.5,1,2`);
```



To see the output pds, we enter

```
> pds:-value(t=1,output=listprocedure);
```

to produces

$$\boxed{[x = \mathbf{proc}(x)\ \ldots\mathbf{end\ proc},\ t = 1.,\ u(x,t) = \mathbf{proc}(x)\ \ldots\mathbf{end\ proc}]}$$

Use "uval:=rhs(op(3,%))" to extract the solution u(x,t) from the third column of the above list and use function "fsolve( )" to determine the x value between 0 and 1 that give u(x,t)=1/2.

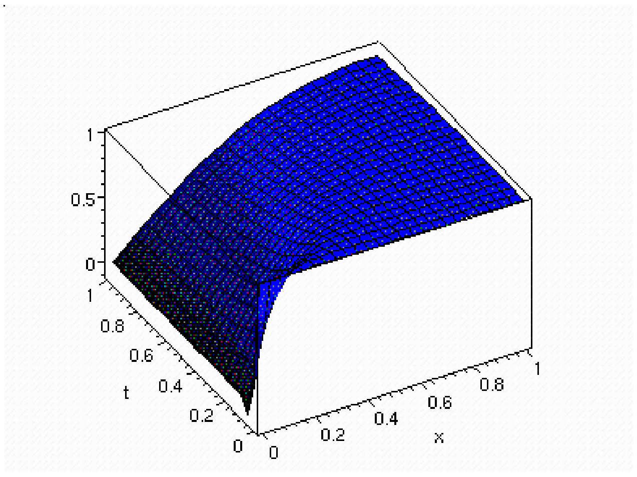
```
> uval := rhs(op(3,%));
> fsolve(uval(x)=1/2,x=0..1);
```

which gives

$$\boxed{0.2978753742}$$

and plot the result using the following statement

```
> pds:-plot3d(t=0..1,x=0..1,axes=boxed,
             orientation=[-120,40],
             color=[0,0,u]);
```



Solution of Boundary Value Problems for PDEs using MATLAB

The MATLAB PDE solver, **pdepe**(), solves initial-boundary value problems for systems of parabolic and elliptic PDEs in one space variable and time. There must be at least one parabolic equation in the system.

$$c(x,t,u,u'(x))\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m f(x,t,u,u'(x)\right) + s(x,t,u,u'(x)) \qquad (*)$$

with initial and boundary conditions in the following forms:

$$\begin{aligned} &u(x,t_0) = u_0(x) \\ &p(x,t,u) + q(x,t)f(x,t,u,u'(x)) = 0 \end{aligned} \qquad (**)$$

The PDEs (*) hold for $t_0 \le t \le t_f$ and $a \le x \le b$. The interval [a,b] must be finite. m can be 0, 1, or 2, corresponding to slab, cylindrical, or spherical symmetry, respectively. If m>0, then $a \ge 0$ must also hold. The basic syntax of the solver is

```
sol = pdepe(m,@pdefun,@icfun,@bcfun,xmesh,tspan)
```

where

`m` : specifies the symmetry of the problem; m can be 0 = slab, 1 = cylindrical, or 2 = spherical.

`pdefun` : Function that defines the components of the PDEs. It computes the terms c, f and s in Equation (*), and has the form

```
[c,f,s] = pdefun(x,t,u,dudx);
```

where `x` and `t` are scalars; and `u` and `dudx` are vectors that approximate the solution u and its partial derivative with respect to x; `c`, `f`, and `s` are column vectors; `c` stores the diagonal elements of the matrix c.

`icfun` : function that evaluates the initial conditions. It has the form

```
u = icfun(x);
```

When called with an argument `x`, **`icfun()`** evaluates and returns the initial values of the solution components at x in the column vector u.

`bcfun` : Function that evaluates the terms and the boundary conditions. It has the form

```
[pl,ql,pr,qr] = bcfun(xl,ul,xr,ur,t);
```

where `ul` is the approximate solution at the left boundary `xl=a` and `ur` is the approximate solution at the right boundary `xr=b`; `pl` and `ql` are column vectors corresponding to p and the diagonal of q evaluated at `xl`. Similarly, `pr` and `qr` correspond to `xr`.

Example Solve
$$\begin{cases} \dfrac{\partial}{\partial t}u(x,t) = \dfrac{1}{10}\left(\dfrac{\partial^2}{\partial x^2}u(x,t)\right) \\ u(x,0) = 1, \quad u(0,t) = 0, \quad \dfrac{\partial}{\partial x}u(1,t) = 0 \end{cases}$$

Firstly, rewrite the PDE in the form of (*) and the boundary conditions in the form of (**).

$$\frac{\partial u}{\partial t} = x^0 \frac{\partial}{\partial x}\left(x^0 \frac{1}{10}\frac{\partial u}{\partial x}\right) + 0$$

with parameter m=0 and the terms

$$c(x,t,u,u'(x)) = 1$$

$$f(x,t,u,u'(x)) = \frac{1}{10}\frac{\partial u}{\partial x}$$

$$s(x,t,u,u'(x)) = 0$$

and

$$u(0,t) + 0 \cdot \left(\frac{1}{10}\frac{\partial u}{\partial x}(0,t) \right) = 0 \quad \text{at} \quad x = 0$$

$$0 \quad + 10 \cdot \left(\frac{1}{10}\frac{\partial u}{\partial x}(1,t) \right) = 0 \quad \text{at} \quad x = 1$$

with the terms

$$pl = 1, \quad ql = 0, \quad pr = 0, \quad qr = 10.$$

Then, we create three M-files for the PDE function “`pdex1pde()`”, the initial condition function “`pdex1ic()`”and the boundary condition function “`pdex1bc()`” as follows.

```
function [c, f, s] = pdex1pde(x,t,u,DuDx)
c = 1;
f = (1/10)*DuDx;
s = 0;
```

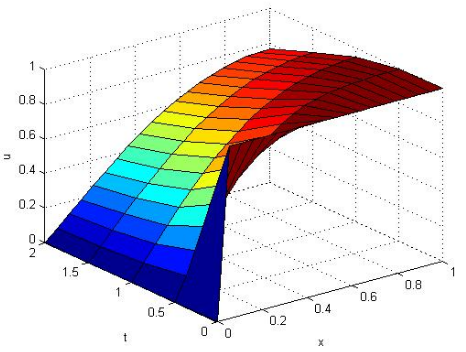
```
function  u0 = pdex1ic(x)
u0 = 1;
```

```
function [pl,ql,pr,qr]=pdex1bc(xl,ul,xr,ur,t)
pl =ul;
ql = 0;
pr = 0;
qr = 10;
```

Next we use the function “**`pdepe`**`()`” to find the solution on the mesh produced by 20 equally spaced points from the spatial interval [0,1] and five values t from the time interval [0, 2] and plot the solution.

```
>> x=linspace(0,1,20);
>> t=linspace(0,2,5);
>> m=0;
>> sol=pdepe(m,@pdex1pde,@pdex1ic,@pdex1bc,x,t);
>> u = sol( :, :, 1);
>> surf(x,t,u);
```

yields



Notes: The output argument `sol` is a three-dimensional array, such that:

`sol(:,:,k)`	approximates component `k` of the solution .
`sol(i,:,k)`	approximates component `k` of the solution at time `tspan(i)` and mesh points `xmesh(:)` .
`sol(i,j,k)`	approximates component k of the solution at time `tspan(i)` and the mesh point `xmesh(j)` .

5

EXERCISE 13

Q13.1 Write a subroutine for solving a linear system Ax = b using the SOR method. Then write a program that use the subroutine to solve the following system of equations correct to 5 decimal places

$$\begin{bmatrix} 10 & -8 & 0 \\ -8 & 10 & -1 \\ 0 & -1 & 10 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -6 \\ 9 \\ 28 \end{bmatrix}$$

Run the program for a sequence of values of ω between 1 and 2.

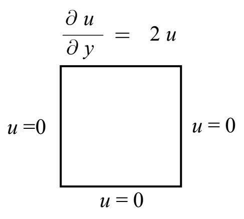
Q13.3 Consider

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = x - 2y$$

on a square with boundary conditions as shown.

a) Construct the finite difference scheme, use $N = 4$ (or $N = 3$)

b) Write a program to solve the BVP.



Q13.3 Write a program that use the subroutines LUFACT and SUBST to solve the system of equations in Q13.1.

Q13.4 Solve the following heat conduction problem using Maple/MATLAB built-in functions

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0 \quad \text{for } 0 < x < 1 \text{ and } t \le 0,$$

$$u(0,t) = u(1,t) = 0 \quad \text{for } t > 0\,,$$

$$u(x,00 = \sin(\pi x) \quad \text{for } 0 \le x \le 1\,.$$

References

1 Aikinson, L.V., Hartley, P.J. and Hudson, J. D. (1989). *Numerical Methods with Fortran 77 – A Practical Introduction*, Addison-Wesley.

2 AMES, W. F. (1977). *Numerical Methods for Partial Differential Equations*. 2nd ed. New York:Academic Press.

3 Bartle, Robert G. (1974). *The Element of Real Analysis*, N.Y.: Wiley.

4 Borse, G.J. (1997). *Numerical methods with MATLAB*. Boston:PWS.

5 Burden, R.L. and Faires, J.D. (1989). *Numerical Analysis*. 4th ed. Boston: PWS-KENT Publishing Company.

6 Butcher, J. (1987). *The Numerical Analysis of Ordinary Differential Equations*. New York: Wiley.

7 Chapra, S.C. and Canale, R.P. (1988). *Numerical Methods for Engineers*. 2nd ed. New York: McGraw-Hill.

8 Davis, P. J. and Rabinowitz, P. (1975). *Methods of Numerical Integration*. New York:Academic Press.

9 Dennis, J.E. Jr. and More, Jorge J. (1977). *Quasi-Newton Methods, Motivation and Theory*, SIAM review, Vol. 19, No. 1, pp46 – 89.

10 Ellis, T.M.R., Phillips, I.R. and Lahey, T.M. (1994). *Fortran 90 Programming*. Addison-Wesley.

11 Etter, D.M. (1992). *Fortran 77 with Numerical Methods for Engineers and Scientists*. The Benjamin/Cummings Publishing Company, Inc.

12 Garvan F. (2002). *The Maple Book*. Chapman & Hall/CRC.

13 Gerald, C.F. and Wheatley, P. O. (1994). *Applied Numerical Analysis*. Addison-Wesley.

14 Griffiths, D.V. and Smith, I.M. (1991). *Numerical Methods for Engineers*. Boca Raton, FL:CRC Press.

15 Abramowitz, M. and Stegun, I.A. (1972). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, Washington, D.C.: U.S. Dept. of Commerce:U.S. G.P.O.

16 Hamming, R.W. (1973). *Numerical Methods for Scientist and Engineers*. 2nd ed. New York: McGraw-Hill.

17 Hildebrand, F.B. (1974*). Introduction to numerical analysis*. 2nd ed. New York:McGraw-Hill.

18 HouseHolder, A. S. (1970). *The numerical treatment of a single nonlinear equation*. New York:McGraw-Hill.

19 Kahaner, D., Moler, C., Nash S. (1989). *Numerical methods and Software*. Englewood Cliffs, N.J.: Prince-Hall.

20 Lapidus, L. and Schiesser, W. E. (1976). *Numerical Methods for Differential Equations*. New York:John Wiley & Sons.

21 Lawson, C. L. and Hanson, R. J. (1974). *Solving Least Squares Problems*. Englewood Cliffs, N.J.: Prince-Hall.

22 Nakamura, S. (1993). *Applied Numerical Methods in C*. Englewood Cliffs, NJ:Prentice Hall.

23 Ortega, J. M. (1972). *Numerical Analysis-A second course*. New York: Academic Press.

24 Ortega, J. M. and W. G. Poole, Jr. (1981). *An Introduction to Numerical Methods for Differential Equations*. Mass.: Pitman Press, Marshfield.

25 Rice, John R. (1983). *Numerical Methods, Software, and Analysis*. New York:McGraw-Hill.

26 Rivlin, Theodore J. (1974). *The Chebyshev Polynomials*, New York: Wiley.

27 Ralston, A., and Rabinowitx, P. (1978). *A First Course in Numerical Analysis*. 2nd ed. New York:McGraw-Hill.

28 Schilling, R. J. and Harris S. L. (1999). *Applied Numerical Methods for Engineers Using MATLAB and C*. Brooks/Cole Publishing Company.

29 Stoer, J. and Bulirsch R. (1980). *Introduction to Numerical Analysis*. New York:Springer-Verlag.

30 Stroud, A. H. and Secrest, D. (1966). *Gaussian Quadrature Formulas*. Englewood Cliffs, N.J.: Prince-Hall.

31 Wiwatanapataphee B, Wu, Y.H. (2006). *Program Design Using C++/F95/MATLAB*. Bangkok: Misterkopy Publishing Company.

32 Young, D. M. (1971). *Iterative Solution of Large Linear Systems*. New York: Academic Press.

Appendix: Introduction to Maple

Maple is an interactive symbolic calculator and a programming language. One could use Maple solely interactively as a graphic calculator, or use it as a programming language, or use both features simultaneously. The aim of this Appendix is to provide a quick reference for using Maple (Version 11) on Standard Worksheet interface.

A1 Getting Start

To work with Maple, double-click the Maple icon and then open a new worksheet in worksheet mode from the File menu via **File**>*New*>*Worksheet Mode.* In worksheet, you can enter Maple commands, and then press **Enter** to execute the commands.

Maple commands are entered at the Maple prompt >, followed by either a semicolon (;) or a colon (:). If a commend ends with a semicolon, the executed result is displayed; while the result is computed but is *not* displayed if the commend ends with a colon. For example

```
 > factor(x^2 + 2*x + 1);
```

yields $(x+1)^2$

Notes:

- You can open, edit and save a worksheet in the same way as for a word document;
- You can execute a particular statement by moving the cursor to the end of the statement and pressing **Enter**; you could *also compute or recompute the entire worksheet* via **Edit**>Execute>Worksheet.
- You can enter commands using 1-D (like x^2) or 2-D Math by using palettes, such as the Matrix palette for inserting matrices and the Expression palette for entering expressions like $\int_0^1 e^x x dx$ etc. However you must use 1-D Math input when programming in Maple. To convert 2-D Math input to 1-D Math input, simply use **Format** > Convert To > 1-D Math Input. You can press F5 to change between 1D and 2D modes.

A2 Arithmetic and Assignment Operations

The basic arithmetic calculations can be performed by using arithmetic operators: +, -, *, /(division) and ^ exponentiation. For example

```
 >2+3*2+5/2;
```

yields 10.5

If the evaluated value of the expression is to be used later, we can assign (store) it into a variable by using the assignment operator := . For example

```
> x:= 2^3:
> y:=x+5;
```

yields y:=13

Notes: The assignment operator can also be used to store an expression in a variable. For example

```
> eq1: = x^2-1=0:
> xsol:=solve(eq1,x);
```

stores the equation $x^2 - 1 = 0$ in the variable eq1 and then solve the equation to yield the roots `xsol:=1, -1.`

A3 Basic Maple Commands and Built-in Functions

There are many commands and built-in functions available in Maple (or its associated packages) for performing certain tasks or solving certain mathematical problems either symbolically or numerically.

Maple built-in functions for solving most of the mathematical problems covered in this book are given in each chapter with some illustrative examples. However, for quick referencing, we list below some common commands and built-in functions often used in mathematical calculation

Command/ Built-in function	Syntax	Operation	Example
evalf	evalf(expr)	Evaluate the value of the expression	evalf(cos(Pi/6))
diff	diff(f(x,y),x,y)	Differentiate *f*(*x*,*y*)with respect to *x* and *y*	diff (x^3*y, x, y)

lhs
rhs
	lhs(expr)
	rhs(expr)
		Return the left (right) hand side of the expr	e:=y=a*x^2+b
			lhs(e); (*returns y*)
			rhs(e); (*returns a*x^2+b*)

?	?	bring up the help menu	

A4 Data Types and Structures

Maple can handle various data structures including: Expression sequences, Sets, Lists, Tables, Arrays, Matrices and Vectors, Functional Operators and Strings.

Expression Sequence

Expression sequence is a group of expressions separated by commas such as

```
 > S:= 1, x, cos(x), exp(x)
```

To access one of the expressions, enter the sequence name followed by the position of the expression enclosed in brackets (). For example

```
 > S(3);
```

yields `cos(x)`

Maple Set

A set is an expression sequence enclosed in curly braces { } such as

```
> L:={1,6, x, y, 2*a}
```

A Maple set has the basic properties of a mathematical set, for example

```
> {1,3,5} ∪ {2,4,6}
```

yields `{1,2,3,4,5,6}`

Note: The union operator is available in 1-D Math input as union.

Maple List

A list is an expression sequence enclosed in brackets [ ] such as

```
> L:= [2,6, 5, 8, 0];
```

To refer to an element in a list, Use square brackets. For example:

```
> L [2..4];
```

yields `[6,5,8]`

Some commands accept a list (or set) of expressions. For example, you can solve a list (or set) of equations using the command

```
> solve ([x-y^2+2=0,x+y=0]);
```

to yield $\{y=-2, x=2\}, \{x=-1, y=1\}$

Matrices and Vectors

Matrices and Vectors are data structures used in linear algebra and vector calculus computations such as

$$> M := \begin{bmatrix} 1 & 0 \\ 2 & 3 \end{bmatrix} : v := <1,2>;$$

$$> M \cdot v;$$

yields $\begin{bmatrix} 1 \\ 5 \end{bmatrix}$

Functional Operator

A functional operator is a mapping $f: x \to y(x)$. The value of $f(x)$ is the result of evaluating $y(x)$.

Using functional operators, you can define mathematical functions. For example, define a function that adds 1 to its input.

```
add1:= x ->x+1
```

You can evaluate the function **add1** with symbolic or numeric arguments. For example

```
> add1 (12); add1 (x+y);
```

yields: 13 x+y+1

A5 Maple Selection Control

You can specify that Maple perform an action only if a condition holds. You can also specify Maple to choose between alternative courses of action, depending upon conditions that are not known until the program is executed. The selection control can be implemented by the following if structure

```
if   (condition 1)  then
       statement sequence 1
elif(condition 2) then
       statement sequence 2
…
else
       statement sequence N
end if
```

Notes: (i) The **elif** (**els**e**if**) clauses are optional. You can specify any number of **elif** clauses; the **else** clause is also optional.

(ii) Once the if structure is executed, Maple will first test condition 1. If condition 1 holds, Maple executes statement sequence 1 and then exits the if structure; otherwise Maple tests condition 2 and so on. If all conditions are not true, statement sequence *N* will be executed.

Example.

```
>  x := 18:
>  if (x<=10) then
      y:=1+x
   elif (x<=20) then
      y:=2+x
   else
      y:=3+x
   end if:
>  y;
```

yields
```
20
```

Note: To enter Maple statements without execution, press **Shift+Enter**

A6 Maple Repetition Control

Maple provides two repetition controls, by a ***counter*** or by a ***condition***. If the number of iterations (repetitions) is known or can be predetermined, we use

```
for counter from n0 by increment to n  do
        statements to be done
end do
```

Notes: If the *increment* is 1, then 'by *increment*' can be omitted.

Example 1: Given $x = (1,2,3,4,5,6)$, calculate the sum of all entries of x .

```
> x:=[1,2,3,4,5,6]:
> SumAll:=0
> for i from 1 to 6 do
      SumAll:=SumAll+x[i]
  end do:
> SumAll:=SumAll;
```

yields `SumAll:=21`

Example 2: Given $x = (1,2,3,4,5,6)$, calculate the sum of all odd entries of x, starting from 5.

```
> x:=[1,2,3,4,5,6]:
> SumOdd:=0
> for i from 5 by -2 to 1 do
      SumOdd:=SumOdd+x[i]
  end do:
> SumOdd:=SumOdd;
```

yields `SumOdd:=9`

If we do not know when to terminate the loop, but want to stop it when a condition becomes true, then we can choose a large enough integer number *n* and use the construction

```
for i from 1 to n   while condition do
        statement(s)
end do
```

Example 3: Given $x = (1,2,3,4,5,6)$, add the entries, starting from the first entry, while the sum is still less than 12.

```
SumCond:=0:
for i from 1 to 100  while SumCond<12 do:
    SumCond:=SumCond+x[i]:
end do:
SumCond:=SumCond;
```

yields `SumCond:=15`

A7 Maple Procedures

A Maple procedure is a program consisting of Maple statements. Using procedures, you can quickly execute the contained sequence of statements.

A Maple procedure starts with a procedure header and ends with **end proc** as shown below:

```
> procedure_name:= proc(var1,var2,…,varn)
     Maple statements
  end proc
```

To run a procedure, enter its name followed by argument values within braces:

```
> procedure_name(var1_value,var2_value,…)
```

Example. Write a procedure to calculate $f(x)=\begin{cases}|x| & \text{if } -1<x<1\\ 1 & \text{otherwise}\end{cases}$

```
> f:=proc(x)
     If(-1<x and x<1) then
         fV:=abs(x)
     else
         fV:=1
     end if
  end proc:
> f:=f(-0.8);
```

yields `f:=0.8`

Note: When a procedure is called, Maple returns **only** the last computation encountered in the procedure. It is irrelevant whether you use semicolons or colons as statement separators. One could use the return statement to invoke an explicit return

A8 Maple Graphics

Two-dimensional plotting

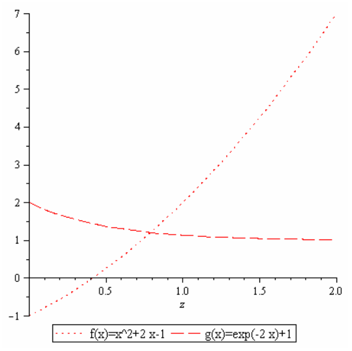
The syntax to plot the function f(x) for x from a to b is `plot(f(x),x=a..b)`

For example, `> plot (cos(x),x=0..2*Pi);`

To do multiple plotting, we can use the display function in the plot package and use option "`legend=s`" and "`linestyle=n`" to distinguish one from the other:

```
> with(plots):
> p1:=plot(x^2+2x-1,x=0..2,linestyle=DOT,legend= "f(x)=x^2+2x-1"):
> p2:=plot(exp(-2x)+1,x=0..2,linestyle=DASH,legend="g(x)=exp(-2x)+1"):
> display(p1,p2);
```

yields



To save a figure, click on the plot in worksheet that you want to save, and then right-mouse click to popup a menu and choose "`export`" and then choose the format that you want the figure to be saved.

To plot data points $\{(x_i y_i)\}_{i=1}^{n}$, we can use command

```
> L:[[x1,y1],[x2,y2],…,[xn,yn]]:
> plot(L);
```

Notes: To plot the points with no connected line, we can use the command

```
> plot(L, style=point, symbol=circle);
```

Three-dimensional plot

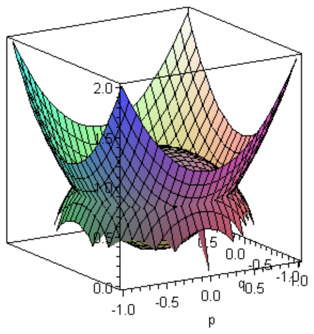
The syntax for plotting *f*(*x*,*y*) for *x* from *a* to *b* and *y* from *c* to *d* is

plot3d(f(x,y),x=a..b,y=c..d)

To do multiple plotting, we can use the display function in the plot package and use option "legend=s" and "linestyle=n" to distinguish one from the other:

```
> with(plots):
> p1:=plot(x^2+y^2, x=-1..1,y=-1..1):
> p2:=plot(sqrt(1-x^2-y^2),x=-1..1,y=-1..1):
> display(p1,p2);
```

yields



INDEX