數值分析
Numerical Analysis
3 學分

姓名：_________________ 學號：_________________

授課教師：吳俊概

目錄

第 0 章 基礎介紹(Preliminaries) .............................................................. 2
第 1 章 非線性方程式之求解(Solving Nonlinear Equations) ......................... 10
第 2 章 聯立方程式之求解(Solving Sets of Equations) ............................... 28
第 3 章 內插與曲線配適(Interpolation and Curve Fitting) ........................... 53
第 4 章 函數之近似(Approximation of Functions) ...................................... 80
第 5 章 數值微分與積分(Numerical Differentiation and Integration) ............. 96
第 6 章 常微分方程式之求解(Numerical Solution of Ordinary Differential Equation) ...... 121
0 Preliminaries

• This book teaches (本課程介紹)
  1) how a computer can be used to solve problems that may not be solvable by the techniques that are taught in most calculus courses. (如何利用電腦來求解以前不方便求解的問題)
  2) how those problems that you may have solved before can be solved in a different way. (如何用不同的方式來求解以前解過的問題)

0.1 Analysis Versus Numerical Analysis (分析與數值分析之差異)

• The word analysis (分析) in mathematics usually means to solve a problem through equations (使用方程式來求解問題的答案). The solving procedures may include algebra, calculus, differential equations, or the like. (求解過程可能用到代數、微積分、微分方程式等運算工具)

**Numerical analysis** (數值分析) is similar in that problems solved, but the only procedures that are used are arithmetic: add, subtract, multiply, divide and compare. (求解過程中只用到算數運算：加、減、乘、除及比較)

Since the arithmetic operations are exactly those that computers can do, numerical analysis and computers are intimately related. (數值分析與電腦息息相關)

• Differences between analytical solutions (解析解) and numerical solutions (數值解):

  1) An analytical solution is usually given in terms of mathematical functions. The behavior and properties of the function are often apparent. However, a numerical solution is always an approximation. It can be plotted to show some of the behavior of the solution. (解析解指能以數學函數表示的一種沒有誤差的解；數值解則是一種有誤差的近似解)

  2) An analytical solution is not always meaningful by itself. (解析解不一定都是意義的)
    Example 1: \( \sqrt{3} \) as one of the roots of \( x^3 - x^2 - 3x + 3 = 0 \).
    Example 2: \( \int_0^\pi \sqrt{1 + \cos^2 x} \, dx \) as the length of one arch of the curve \( y = \sin x \).
This book will describe numerical methods that can solve these little problems efficiently and also those for much more difficult ones.

3) While the numerical solution is an approximation, it can usually be evaluated as accurate as we need. Actually, evaluating an analytic solution numerically is subject to the same errors. (雖然數值解終究是一種近似解，但其準確程度通常可以達到我們的要求。事實上，在我們計算一個解析解的數值大小時，也會產生相同程度的誤差。)

• The analysis of computer errors and the other sources of errors in numerical methods is critically important and will be introduced throughout this book. (本課程將介紹電腦的誤差與數值方法的誤差之不同)

0.2 Computers and Numerical Analysis (電腦與數值分析之關係)

• 關係圖：( 數值分析工作 = 數值方法 + 程式軟體 + 電腦硬體 )

| Numerical Methods + Programs | Computers | Numerical Analysis |

• As you will learn enough about many numerical methods, you will be able to write programs to implement them. (在學好各種數值方法的同時，就是在學會如何寫程式來套用它們。)

• Programs can be written in any computer language. The specific computer language used is not very important. Most of the methods will be described fully throughout pseudocode (虛擬碼) in such a form that translating this code into a program is relatively straightforward. (程式可以用任何一種語言來撰寫，用哪一種語言來撰寫並不是很重 要。大多數的方法都以虛擬碼的形式被完整地描述出來，可以相當直接地將之改寫 成一般的程式碼。)

• Actually, writing programs is not always necessary. Numerical analysis is so important that extensive commercial software packages are available. (有許多商用的套裝軟體可以用於數值分析)

• A set of books, called Numerical Recipes (數值菜單), lists and discusses numerical analysis programs in a variety of languages, i.e. C and FORTRAN.
0.4 Kinds of Errors in Numerical Procedures

- The total error comprises of (總誤差由下列誤差組成)
  1) **Model Error** (模型誤差): due to the mismatch between the physical situation and the mathematical model
  2) **Data Error** (資料誤差): due to the measurements of doubtful accuracy
  3) **Human Error** (人為誤差): due to human blunders (粗心)
  4) **Truncation Error** (截去誤差): referred to those errors caused by the method itself (數值方法本身所產生的誤差)
     i.e. a) due to truncating an infinite series into a finite one
        \[ e^x = \sum_{n=0}^{\infty} \frac{1}{n!} x^n = 1 + x + \frac{1}{2} x^2 + \frac{1}{6} x^3 + \sum_{n=4}^{\infty} \frac{1}{n!} x^n \approx 1 + x + \frac{1}{2} x^2 + \frac{1}{6} x^3 \]
     b) due to the truncation of an iteration process (由於疊代過程的停止)
  5) **Round-Off Error** (四捨五入誤差): due to the precision of computer (肇因於電腦精確度的不足)
     - 計算的誤差等於截去誤差加上四捨五入誤差，即
       \[
       \text{computational error} = \text{truncation error} + \text{round-off error}
       \]
     - In Chapter 5, we will show how the derivative of a function can be found numerically. Analytically, \( df / dx \) is given by
       \[
       f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}.
       \]
       If we make \( h \) smaller and smaller, the truncation error is reduced and the result is closer and closer to the true value for the derivative.

       But at some point, depending on the precision of the computer, round-off errors will dominate and the results become less accurate. There is a point where the computational error is least.

  6) **Propagated Error** (傳播誤差): the error in the succeeding steps of a process due to an occurrence of an earlier error
     - A method is called **unstable** (非穩定的) if errors are magnified continuously as the method continues and will eventually overshadow the true value or destroy its validity.
A method is called **stable** (穩定的) if errors made at early points die out as the method continues.

- There are two common ways to express the size of the error in a computed result: **absolute error** (絕對誤差) and **relative error** (相對誤差).

  
  \[
  \text{relative error} = \frac{\text{absolute error}}{|\text{true value}|}, \quad \text{which is a desirable one.}
  \]

  
  \[
  \text{absolute error} = |\text{true value} - \text{approximate value}|, \quad \text{which is usually used when the magnitude of the true value is small.}
  \]

- Another term commonly used to express accuracy is the **number of significant digits** (有效位數). The number of significant digits is defined as follows:

  
  Let the true value have digits \( d_1, d_2, \ldots, d_n, d_{n+1}^{+}, \ldots \)

  and the approximate value have digits \( d_1, d_2, \ldots, d_n, e_{n+1}, e_{n+2}, \ldots \)

  where \( d_i \neq 0 \) and \( d_{n+1}^{+} \neq e_{n+1}^{+} \).

  Then we say they agree to \( n \) significant digits if

  \[
  |d_{n+1} - e_{n+1}| < 5^n \]

  and \( n-1 \) significant digits if

  \[
  |d_{n+1} - e_{n+1}| \geq 5^n.
  \]

**Example 0.7**  Let the true value = \( \frac{10}{3} \), the approximate value = 3.333.

Then the absolute error = \( \left| \frac{10}{3} - 3.333 \right| = \frac{1}{3000} \).

the relative error = \( \frac{1/3000}{10/3} = \frac{1}{10000} \),

the number of significant digit is 4.

- When we compute a value of a function, \( y = f(x) \), we may not get exactly the correct value of \( y \) due to computational error. (由於計算誤差的存在，在電腦裡計算得到的函數值並非完全準確)

- The accuracy of a numerical solution depends not only on the computer’s accuracy. The problem itself is a factor. (一個數值解的準確度不只和電腦的準確度有關，還和問題的本身有關。)

A problem is called **well-posed** if a solution 1) exists, 2) is unique and 3) varies continuously when the values of its parameters vary continuously. (若一問題之解為存
在、唯一且大小呈連續變化，則稱之為設定合宜的問題）

An *ill-conditioned* problem (條件有誤的問題) is particularly sensitive to changes in the values of the parameters. In this case, a small change in the input causes a large change in the output. A *well-conditioned* problem (條件合宜的問題) is not so sensitive. A well-conditioned problem gives useful results in spite of small inaccuracies in the parameters. (反之，稱之為條件有誤的問題)

- A computer number has three parts to store a floating-point number (浮點數): the *sign* (正負符號), *fraction* (分數) (also called *mantissa* (假數)) and *exponent* (指數) parts. The *IEEE standard* is by far the most common to store the number as a binary quantity:

<table>
<thead>
<tr>
<th>Precision</th>
<th>Length</th>
<th>Sign</th>
<th>Mantissa</th>
<th>Exponent</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>32</td>
<td>1</td>
<td>23(+1)</td>
<td>8</td>
<td>$10_{10}^{\pm38}$</td>
</tr>
<tr>
<td>Double</td>
<td>64</td>
<td>1</td>
<td>52(+1)</td>
<td>11</td>
<td>$10_{10}^{\pm308}$</td>
</tr>
<tr>
<td>Extended</td>
<td>80</td>
<td>1</td>
<td>64</td>
<td>15</td>
<td>$10_{10}^{\pm4931}$</td>
</tr>
</tbody>
</table>

It is obvious that only a finite number of different values can be stored in a computer that uses the IEEE standard. And since there is an infinite number of real numbers, it is clear that there must be gaps between the stored values. This is the source of round-off error.

There is a most important consequence to this. When writing a computer program, never use a test such as

If $A=B$, then… (在電腦程式裡，切莫做兩個浮點數是否相等之比較)

Instead, one should write

If $|A-B|\leq TOL$, then… (tolerance 為容許誤差值)
There is a largest number and smallest number (in magnitude) in the system. Quantities that exceed the maximum cause overflow; too small numbers cause underflow.

When overflow occurs, many computers replace the value with a special bit pattern that represents infinity; when underflow occurs, they replace the value with zero.

In single precision, the smallest and largest numbers are:
- Smallest: \(2.93873 \times 10^{-39}\),
- Largest: \(3.40282 \times 10^{38}\).

The storage of zero is a special case. In the IEEE standard, zero is stored as all zeros, that is, the sign, fraction and exponent parts are all zeros.

- The term “\(\varepsilon\)” (the Greek latter epsilon) is used to represent the smallest machine value that can be added to 1.0 that gives a result distinguishable from 1.0.
  - In IEEE single precision: \(\varepsilon = 1.1920E-7 \approx 2^{-23}\),
  - In IEEE double precision: \(\varepsilon = 2.2204E-16 \approx 2^{-52}\).

When adding a set of numbers, adding them in the order from smallest (in magnitude) to the largest gives a more accurate result than if they were added from largest to smallest. For example, adding 0.0001 one thousand times should equal 1.0 exactly, but this is not true with single precision.

- There is another unexpected result. Compute in single precision the following expression:
  \[
  \frac{(X + Y)^2 - 2XY - Y^2}{X^2} = Z.
  \]

  We have

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1000</td>
<td>1.000000</td>
</tr>
<tr>
<td>0.001</td>
<td>1000</td>
<td>0.9999998</td>
</tr>
<tr>
<td>0.0001</td>
<td>1000</td>
<td>0.999213</td>
</tr>
<tr>
<td>0.00001</td>
<td>1000</td>
<td>1.000444</td>
</tr>
<tr>
<td>0.000001</td>
<td>1000</td>
<td>0.682120</td>
</tr>
<tr>
<td>0.0000001</td>
<td>1000</td>
<td>-79.58079</td>
</tr>
</tbody>
</table>
Homework #1

1. Let \( f(x) = e^x \). Use the definition \( f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \) to find \( f'(1) \) for various small values of \( h \).

| \( h \)       | \( (a) = \lim_{h \to 0} \frac{e^{1+h} - e^1}{h} \) | relative error, \( \frac{|(a) - e|}{e} \times 100\% \) |
|-------------|---------------------------------|---------------------------------|
| 0.1         |                                  |                                 |
| 0.01        |                                  |                                 |
| 0.001       |                                  |                                 |
| 0.0001      |                                  |                                 |
| 0.00001     |                                  |                                 |
| ……          |                                  |                                 |

2. For single and double precisions, find
   1) the maximum positive floating-point number, (最大的正的浮點數)
   2) the minimum positive floating-point number, (最小的正的浮點數)
   3) “\( \varepsilon \)” (epsilon)
in a 32-bit IBM compatible PC. Show in detail the procedures that you have used.

0.5 Parallel and Distributed Computing (平行式與分散式計算)

- One of the first techniques to increase the operating speed of a computer was “pipelining” (管線) – performing a second instruction within the CPU before the previous instruction is completed. This technique takes advantage of the fact that doing a single “instruction” actually involves several micro-coded steps. Pipelining permits a speedup by a factor of two or more.

Another technique is to build vector processing (向量式處理) operations into the CPU. Because the individual steps required to solve sets of equations involve many multiplications of a vector by another vector, these machines offer speed improvements by a factor of 5 or 10, but not the factor of 10,000 that is really desired.

The current trend is to use parallel computing (平行式計算) – putting several machines or CPUs to work on a single problem. For parallel computers, all machines run under a single clock (synchronous/同步).
Recently, much work and interest are found in distributed computing — putting many different computers to work separately on their own tasks as well as in conjunction with each other. For distributed computers, each machine runs under its own clock (asynchronous).

The questions about programming a parallel system are not yet settled. If it were possible to have the compiler recognize parallelism within a conventional program, the task of programming would be much easier.

- The term “speedup” (加速) is used to describe the increased performance of a parallel system compared to a single processor.

The term “efficiency” (效率) is used to describe how the speedup compares to the number of processors used.

Example

When a polynomial is evaluated, it is inefficient to do it in the sequential way:

\[ a_3 x^3 + a_2 x^2 + a_1 x + a_0. \]

(6 multiplications and 3 additions)

Instead, it is efficient to put it into “nested form” (巢式) as

\[ ((a_3 x + a_2) x + a_1) x + a_0 \]

(3 multiplications and 3 additions)

Nested multiplication is not only faster, but there is less error due to round off. (巢式計算不僅算得較快，也因為四捨五入誤差的減少而算得較準確)

“The purpose of computing is insight, not numbers,”

計算的目的在發掘真相，而不在數字本身。

said Hamming.
1 Solving Nonlinear Equations (求解非線性方程式)

- The values of \( x \) that make \( f(x) = 0 \) are called the roots (根) of the equation. They are also called the zeros (零解) of \( f(x) \).

This chapter describes some of the many methods for solving the roots of \( f(x) = 0 \) by numerical procedures.

A total of ten root-finding procedures are shown in this chapter. Five of these are described in detail, the others are only mentioned. Why so many? It is because there are often many numerical methods for solving a problem, and one has to find out a favor within them. (對應於一種問題的求解，通常會發展出許多種數值方法，吾人需從中選擇偏愛的方法)

1.1 Interval Halving or Bisection Method (二分法)

- If \( f(x) \) is continuous and changes signs at two \( x \)-values, then there must be at least one root between the two values. (一個連續函數 \( f(x) \)，若其值在兩個不同的 \( x \) 位置上變號，則在該兩 \( x \) 位置間至少有一根)

The bisection method successively divides the initial interval in half, finds in which half the root must lie, and repeats with the endpoints of the smaller interval.
- Algorithm of the Bisection Method:

Given \( x_1 \) and \( x_2 \) such that \( f(x_1) \cdot f(x_2) < 0 \),

Repeat

Set \( x_3 = \frac{1}{2}(x_1 + x_2) \)

If \( f(x_1) \cdot f(x_3) < 0 \) then

Set \( x_2 = x_3 \)

Else

Set \( x_1 = x_3 \)

End if

Until \( |x_1 - x_2| < 2 \cdot TOL \)

---

Example: Find a zero root of \( f(x) = 3x + \sin x - e^x \) in the interval \([0, 1]\)

```c
#include <stdio.h>
#include <math.h>
int main()
{
    float x1 = 0.0 , x2 = 1.0 , x3;
    float fx1 , fx2 , fx3;
    float TOL = 1.0E-4;
    int Iter = 1;
    FILE *fPtr;

    fPtr = fopen( "output.dat" , "w" );
    fx1 = 3.0 * x1 + sin( x1 ) - exp( x1 );
    fx2 = 3.0 * x2 + sin( x2 ) - exp( x2 );
    fprintf( fPtr , "%d %10s %10s %10s %10s\n" ,
             "Iter" , "x1" , "x2" , "x3" , "f(x3)" , "ERRmax" );

    while ( 1 ) {
        x3 = 0.5 * ( x1 + x2 );
        fx3 = 3.0 * x3 + sin( x3 ) - exp( x3 );
        fprintf( fPtr , "%4d %10.5f %10.5f %10.5f %10.5f\n" ,
                 Iter , x1 , x2 , x3 , fx3 , 0.5*fabs(x1-x2) );
        if ( fx1 * fx3 < 0.0 ) {
            x2 = x3;
            fx2 = fx3;
        } else {
            x1 = x3;
            fx1 = fx3;
        }
        if ( fabs( x1 - x2 ) < 2.0 * TOL ) break;
        Iter++;
    }
    fprintf( fPtr , "The root we have found is :%10.5f\n" , 0.5 * ( x1 + x2 ) );
    fclose( fPtr );
    return 0;
}
```


<table>
<thead>
<tr>
<th>Iteration</th>
<th>(X_1)</th>
<th>(X_2)</th>
<th>(X_3)</th>
<th>(F(X_3))</th>
<th>Maximum error</th>
<th>Actual error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00000</td>
<td>1.00000</td>
<td>0.50000</td>
<td>0.33070</td>
<td>0.50000</td>
<td>0.13958</td>
</tr>
<tr>
<td>2</td>
<td>0.00000</td>
<td>0.50000</td>
<td>0.25000</td>
<td>0.28662</td>
<td>0.25000</td>
<td>-0.11042</td>
</tr>
<tr>
<td>3</td>
<td>0.25000</td>
<td>0.50000</td>
<td>0.37500</td>
<td>0.03628</td>
<td>0.12500</td>
<td>0.01458</td>
</tr>
<tr>
<td>4</td>
<td>0.25000</td>
<td>0.37500</td>
<td>0.31250</td>
<td>-0.12190</td>
<td>0.06250</td>
<td>-0.04792</td>
</tr>
<tr>
<td>5</td>
<td>0.31250</td>
<td>0.37500</td>
<td>0.34375</td>
<td>-0.04196</td>
<td>0.03125</td>
<td>-0.01667</td>
</tr>
<tr>
<td>6</td>
<td>0.34375</td>
<td>0.37500</td>
<td>0.35958</td>
<td>-0.00262</td>
<td>0.01563</td>
<td>-0.00105</td>
</tr>
<tr>
<td>7</td>
<td>0.35938</td>
<td>0.37500</td>
<td>0.36719</td>
<td>0.01689</td>
<td>0.00781</td>
<td>0.00677</td>
</tr>
<tr>
<td>8</td>
<td>0.35938</td>
<td>0.36719</td>
<td>0.36328</td>
<td>0.00715</td>
<td>0.00391</td>
<td>0.00286</td>
</tr>
<tr>
<td>9</td>
<td>0.35938</td>
<td>0.36328</td>
<td>0.36133</td>
<td>0.00227</td>
<td>0.00195</td>
<td>0.00091</td>
</tr>
<tr>
<td>10</td>
<td>0.36133</td>
<td>0.36133</td>
<td>0.36035</td>
<td>-0.00018</td>
<td>0.00098</td>
<td>-0.00007</td>
</tr>
<tr>
<td>11</td>
<td>0.36035</td>
<td>0.36133</td>
<td>0.36084</td>
<td>0.00105</td>
<td>0.00049</td>
<td>0.00042</td>
</tr>
<tr>
<td>12</td>
<td>0.36035</td>
<td>0.36084</td>
<td>0.36060</td>
<td>0.00044</td>
<td>0.00024</td>
<td>0.00017</td>
</tr>
<tr>
<td>13</td>
<td>0.36035</td>
<td>0.36060</td>
<td>0.36047</td>
<td>0.00013</td>
<td>0.00012</td>
<td>0.00005</td>
</tr>
</tbody>
</table>

Advantages: (優點)

1) It is guaranteed to work if \( f(x) \) is continuous in \([a, b]\) and if \( x = a \) and \( x = b \) actually bracket a root.

2) The number of iterations to achieve a specified accuracy is known in advance.

\[
\text{Error after } n \text{ iterations} < \left| \frac{b-a}{2^n} \right|
\]

Disadvantages: (缺點)

1) It is slow to converge.

The bisection method is generally recommended for finding an approximate value for the root, and then this value is refined by more efficient methods. (二分法常被用來鎖定根的位置，然後再用其他更有效率的方法將準確度提高) The reason is that most other root-finding methods require a starting value near a root – lacking this, they may fail completely.

Graphing the function \( f(x) \) is always helpful in showing where roots occur. (繪製函數的圖形對於標示根的所在位置是很有幫助的) When there are multiple roots, the bisection method may not be applicable, because the function may not change sign at points on either side of the roots. Here a graph will be most important to reveal the situation.
1.2 Linear Interpolation/Extrapolation Methods (線性內插/外插法)

• Most function can be approximated by a straight line over a small interval. The two methods of this section are based on doing just that.

The Secant Method (割線法)
• The secant method begins by finding two points on the curve of \( f(x) \), hopefully near to the root we seek. A graph or a few applications of bisection might be used to determine the approximate location of the root. The two points may be on one side or on opposite sides of the root.

\[
\frac{x_1 - x_2}{f(x_1) - 0} = \frac{x_0 - x_1}{f(x_0) - f(x_1)}
\]

\[ \Rightarrow x_2 = x_1 - f(x_1) \cdot \frac{x_0 - x_1}{f(x_0) - f(x_1)} \]

• Algorithm of the Secant Method:

~~~~~~~~~~~
Given \( x_0 \) and \( x_1 \) that are near the root,
Repeat
Set \( x_2 = x_1 - f(x_1) \cdot \frac{x_0 - x_1}{f(x_0) - f(x_1)} \)
Set \( x_0 = x_1 \)
Set \( x_1 = x_2 \)
Until \( |x_0 - x_1| < 2 \cdot TOL \)

~~~~~~~~~~~

Example: Find a zero root of \( f(x) = 3x + \sin x - e^x \) in the interval \([0, 1]\)

```c
#include <stdio.h>
#include <math.h>
int main()
{
```
```c
float x0 = 1.0, x1 = 0.0, x2;
float fx0, fx1, fx2;
float TOL = 1.0E-4;
int Iter = 1;
FILE *fPtr;

fPtr = fopen(“output.dat”, “w”);
fx0 = 3.0 * x0 + sin( x0 ) - exp( x0 );
fx1 = 3.0 * x1 + sin( x1 ) - exp( x1 );
fprintf( fPtr, “Iter. %10s %10s %10s %10s
”, “x0”, “x1”, “x2”, “f(x2)”);
while ( 1 ) {
x2 = x1 - fx1 * ( x0 - x1 ) / ( fx0 - fx1 );
fx2 = 3.0 * x2 + sin( x2 ) - exp( x2 );
fprintf( fPtr, ”%4d %10.5f %10.5f %10.5f %10.5f
”, Iter, x0, x1, x2, fx2, 0.5*fabs(x0-x1) );
    x0 = x1;
x1 = x2;
fx0 = fx1;
fx1 = fx2;
if ( fabs( x0 - x1 ) < 2.0 * TOL ) break;
    Iter++;
}
fprintf( fPtr, “The root we have found is:%10.5f
”, 0.5 * ( x0 + x1 ) );
fclose( fPtr );
return 0;
}
```

The root we have found is : 0.36042

<table>
<thead>
<tr>
<th>Iter.</th>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>f(x2)</th>
<th>ERRmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.47099</td>
<td>0.26516</td>
<td>0.50000</td>
</tr>
<tr>
<td>2</td>
<td>0.00000</td>
<td>0.47099</td>
<td>0.37228</td>
<td>0.02953</td>
<td>0.23549</td>
</tr>
<tr>
<td>3</td>
<td>0.47099</td>
<td>0.37228</td>
<td>0.35990</td>
<td>-0.00129</td>
<td>0.04936</td>
</tr>
<tr>
<td>4</td>
<td>0.37228</td>
<td>0.35990</td>
<td>0.36042</td>
<td>0.00001</td>
<td>0.00619</td>
</tr>
<tr>
<td>5</td>
<td>0.35990</td>
<td>0.36042</td>
<td>0.36042</td>
<td>0.00000</td>
<td>0.00026</td>
</tr>
</tbody>
</table>

The root we have found is : 0.36042

Table 1.2 Secant method on f(x) = 3x + sin(x) - e^x

At x = 0.3604217, tolerance of 0.000001 met!

The Linear Interpolation Method (線性內插法)

- A method, called linear interpolation or false position, is similar to the bisection method except the next iterate is taken at the intersection of a line between the pair of x-values and
the $x$-axis rather than at the midpoint.

- The false position method gives faster convergence than does the bisection method, but at the expense of a more complicated algorithm.

- Algorithm of the Linear Interpolation Method:

\[
\text{Given } x_0 \text{ and } x_1 \text{ such that } f(x_0) \cdot f(x_1) < 0,
\]

\[
\text{Repeat}
\]

\[
\text{Set } x_2 = x_1 - f(x_1) \cdot \frac{x_0 - x_1}{f(x_0) - f(x_1)}
\]

\[
\text{If } f(x_2) \cdot f(x_0) < 0 \text{ then}
\]

\[
\text{Set } x_1 = x_2
\]

\[
\text{Else}
\]

\[
\text{Set } x_0 = x_2
\]

\[
\text{End if}
\]

\[
\text{Until ( } |x_0 - x_1| < 2 \cdot TOL \text{ or } |f(x_2)| < TOL \) }
\]

Example: Find a zero root of $f(x) = 3x + \sin x - e^x$ in the interval $[0, 1]$

```c
#include <stdio.h>
#include <math.h>

int main()
{
    float x0 = 1.0, x1 = 0.0, x2;
    float fx0, fx1, fx2;
    float TOL = 1.0E-4;
    int Iter = 1;
    FILE *fPtr;

    fPtr = fopen( "output.dat" , "w" );
    fx0 = 3.0 * x0 + sin( x0 ) - exp( x0 );
    fx1 = 3.0 * x1 + sin( x1 ) - exp( x1 );
    fprintf( fPtr , "Iter. %10.5f %10.5f %10.5f %10.5f\n", 
    "x0" , "x1" , "x2" , "f(x2)" );

    while ( 1 ) {
        x2 = x1 - fx1 * ( x0 - x1 ) / ( fx0 - fx1 );
        fx2 = 3.0 * x2 + sin( x2 ) - exp( x2 );
        fprintf( fPtr , "%4d \%10.5f \%10.5f \%10.5f \%10.5f\n", 
            Iter , x0 , x1 , x2 , fx2 );
        if ( fx2 * fx0 < 0.0 ) {
            x1 = x2;
            fx1 = fx2;
        } else {
            x0 = x2;
            fx0 = fx2;
        }
        if ( fabs( x0 - x1 ) < 2.0 * TOL || fabs( fx2 ) < TOL ) break;
        Iter++;
    }

    return 0;
}
```
```c
} fprintf ( fPtr, "The root we have found is :\%10.5f\n", x2 );
fclose ( fPtr );
return 0;
}

執行結果─
Iter. x0 x1 x2 f(x2)
1 1.00000 0.00000 0.47099 0.26516
2 0.47099 0.00000 0.37228 0.02953
3 0.37228 0.00000 0.36160 0.00294
4 0.36160 0.00000 0.36054 0.00029
5 0.36054 0.00000 0.36043 0.00003
The root we have found is : 0.36043
```

The root we have found is : 0.36043

<table>
<thead>
<tr>
<th>Iteration</th>
<th>x</th>
<th>f(x)</th>
<th>x</th>
<th>f(x)</th>
<th>x</th>
<th>f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.330704</td>
<td>0.470990</td>
<td>0.265160</td>
<td>0.470990</td>
<td>0.265160</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>-0.286621</td>
<td>0.372277</td>
<td>0.029533</td>
<td>0.372277</td>
<td>0.029533</td>
</tr>
<tr>
<td>3</td>
<td>0.375</td>
<td>0.036281</td>
<td>0.361598</td>
<td>2.94 * 10^-3</td>
<td>0.359904</td>
<td>-1.29 * 10^-3</td>
</tr>
<tr>
<td>4</td>
<td>0.3125</td>
<td>-0.121899</td>
<td>0.360538</td>
<td>2.90 * 10^-4</td>
<td>0.360424</td>
<td>5.55 * 10^-6</td>
</tr>
<tr>
<td>5</td>
<td>0.34375</td>
<td>-0.041956</td>
<td>0.360433</td>
<td>2.93 * 10^-5</td>
<td>0.360422</td>
<td>3.55 * 10^-7</td>
</tr>
</tbody>
</table>

Error after 5 iterations: 0.01667
-1.17 * 10^-5 < -1 * 10^-7

(Exact value of root is 0.360421703.)

結論：5次疊代後之誤差：二分法 > 線性內插法 > 割線法，所以「割線法」較優！

- 16 -
1.3 Newton’s Method (牛頓法)

- **Newton’s method** is one of the most widely used methods of solving equations. (牛頓法是最廣泛使用的求解根的方法) Like the previous ones, this method is also based on a linear approximation of the function, but does so using a tangent to the curve.

![Figure 1.3 A graphical description of Newton’s Method](image)

- It follows immediately from the right triangle shown in Figure 1.3 that

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

Newton’s algorithm is widely used because it is more rapidly convergent than any of the methods discussed so far. (牛頓法之所以被廣泛地使用，是因為它是我們已經討論過的方法中收斂速度最快的)

We will show in a later section that the method is quadratically convergent. It means that the error of each step approaches a constant \( K \) times the square of the error of the previous step. (牛頓法的誤差收斂速度為二次式，即每次疊代後的誤差值約略等於某個常數 \( K \) 乘以疊代前誤差值的平方) The net result of this is that the number of decimal places of accuracy nearly doubles at each iteration.

However, there is the need for two function evaluations \( f(x_n) \) and \( f'(x_n) \) at each step, and we must obtain the derivative function at the start. (每次疊代需計算兩次函數值，且函數的導數必須事先知道)

- The method may converge to a root different from the expected one or diverge if the starting value is not close enough to the root. (如果給定的起始點不夠靠近要找的那個解，牛頓法可能會收斂於另一個解或發散掉)
Figure 1.4 A fail situation (一種找解失敗的情形)

- Algorithm for Newton’s Method:

\[
\begin{align*}
\text{Given } x_0 \text{ reasonably close to the root,} \\
\text{Repeat} \\
\text{Set } x_1 = x_0 \\
\text{Compute } f(x_0), \ f'(x_0) . \\
\text{If } f'(x_0) = 0.0 \text{ break} \\
\text{Set } x_0 = x_0 - \frac{f(x_0)}{f'(x_0)} \\
\text{Until } (|x_0 - x_1| < TOL \text{ or } |f(x_0)| < TOL )
\end{align*}
\]

Example: Find a zero root of \( f(x) = 3x + \sin x - e^x \) beginning with \( x_0 = 0.0 \).

\[
\begin{align*}
x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} = 0.0 - \frac{-1.0}{3.0} = 0.33333 ; \\
x_2 &= x_1 - \frac{f(x_1)}{f'(x_1)} = 0.33333 - \frac{-0.068418}{2.54934} = 0.36017 ; \\
x_3 &= x_2 - \frac{f(x_2)}{f'(x_2)} = 0.36017 - \frac{-6.279 \times 10^{-4}}{2.50226} = 0.3604217 ; \\
\ldots
\end{align*}
\]

```c
#include <stdio.h>
#include <math.h>
int main()
{
   float x0 = 0.0 , x1;
   float fx0 , fx1 , fpx0;
   float TOL = 1.0E-4;
   int Iter = 1;
   FILE *fPtr;
   ```
fPtr = fopen( "output.dat" , "w" );
fx0 = 3.0 * x0 + sin( x0 ) - exp( x0 );
fp0 = 3.0 + cos( x0 ) - exp( x0 );
printf( fPtr , "Iter. %10s %10s %10s\n", "x0" , "x1" , "f(x2)" );
while ( 1 ) {
x1 = x0 - fx0 / fp0;
fx1 = 3.0 * x1 + sin( x1 ) - exp( x1 );
printf( fPtr , "%4d %10.5f %10.5f %10.5f\n", Iter , x0 , x1 , fx1 );
if ( fabs( x0 - x1 ) < TOL || fabs( fx1 ) < TOL ) break;
x0 = x1;
fx0 = fx1;
fp0 = 3.0 + cos( x0 ) - exp( x0 );
Iter++;
}
printf( fPtr , "The root we have found is :%10.5f\n" , x1 );
fclose( fPtr );
return 0;
}

關係幾何與牛頓法（牛頓法與割線法之間的關係）

- 有另一個替代方法來獲得牛頓法的導數。

我們可以計算 \( f(x) \) 在兩個紧密接近的值 \( x \) 與 \( f \) 的差值，再除以 \( x \)-值的差值，即

\[
f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.
\]

然後對 \( \text{牛頓法} \)

\[
x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \approx x_n - \frac{f(x_n)}{f(x_n) - f(x_{n-1})} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}
\]

它與 \( \text{割線法} \) 完全相同。
Newton’s method works with complex roots if we give it a complex value for the starting value. (牛頓法也適用於複數根的求取)

Example: Find zero roots of $f(x) = x^3 + 2x^2 - x + 5$.

Figure 1.5 The graph of $f(x) = x^3 + 2x^2 - x + 5$.

Figure 1.5 shows the graph of $f(x)$. It has a real root at about $x = -3$.

1) If we begin with a real starting value, say $x_0 = -3$, we get convergence to the root $x = -2.92585$.

2) If we begin with $x_0 = 1 + i$, we get these successive iterates:

\[
\begin{align*}
  x_0 &= 1 + i; \\
  x_1 &= 0.486238 + 1.04587i; \\
  x_2 &= 0.448139 + 1.23665i; \\
  x_3 &= 0.462720 + 1.22242i; \\
  x_4 &= 0.462925 + 1.22253i; \\
  \cdots
\end{align*}
\]

3) If we begin with $x_0 = 1 - i$, the method converges to the conjugate.

Solve by MATLAB:

```matlab
>> p = [1 2 -1 5]
p =
  1   2  -1   5
>> r = roots(p)
r =
 -2.9259
  0.4629 + 1.2225i
  0.4629 - 1.2225i
```

- 20 -
Homework #2
The function \( f(x) = 2 \sin x - \frac{1}{4} e^x - 1 \) is zero for two values near \( x = -5 \). (函数在 \( x = -5 \)附
近有兩個根) Let the tolerance error be \( TOL = 1 \times 10^{-6} \).
Use
1) the bisection method,
2) Newton’s method to find roots by starting with \( x = -3, -5 \) and \(-7\).

1.4 Muller’s Method

• Muller’s method is based on approximating the function in the neighborhood of the root
by a quadratic polynomial (二次多項式).

![Figure 1.7](image)

Figure 1.7 A graphical description for Muller’s method

Let \( \gamma = h_2/h_1 \). We have

\[
a = \frac{\gamma f_1 - (1 + \gamma) f_0 + f_2}{\gamma(1 + \gamma) h_1^2}, \quad b = \frac{f_1 - f_0 - ah_1^2}{h_1}, \quad c = f_0.
\]

Then the root of \( av^2 + bv + c = 0 \) is

\[
v = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (\text{if} \quad b > 0, \text{ choose minus}; \quad \text{if} \quad -b < 0, \text{ choose plus})
\]

or

\[
\frac{-2c}{b \pm \sqrt{b^2 - 4ac}} \quad (\text{if} \quad b > 0, \text{ choose plus}; \quad \text{if} \quad b < 0, \text{ choose minus})
\]

\[
\Rightarrow \quad x = x_0 + \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad \text{or} \quad x = x_0 + \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}
\]
Experience shows that Muller’s method converges at a rate that is similar to that for Newton’s method. It does not require the evaluation of derivatives and needs only one function evaluation per iteration. (經驗顯示 Muller 法的收斂速度類似於牛頓法。Muller 法的使用不需要計算導數且每次疊代只需計算一次函數值。)

Example 1.2
Find a root between 0 and 1 of the same transcendental function as before: \( f(x) = 3x + \sin(x) - e^x \). Let

\[
\begin{align*}
\quad x_0 & = 0.5, \quad f(x_0) = 0.330704 & h_1 & = 0.5, \\
\quad x_1 & = 1.0, \quad f(x_1) = 1.123489 & h_2 & = 0.5, \\
\quad x_2 & = 0.0, \quad f(x_2) = -1 & \gamma & = 1.0.
\end{align*}
\]

Then

\[
\begin{align*}
a &= \frac{(1.0)(1.123189) - 0.330704(2.0) + (-1)}{1.0(0.5)^2(2.0)} = -1.07644, \\
b &= \frac{1.123189 - 0.330704 - (-1.07644)(0.5)^2}{0.5} = 2.12319, \\
c &= 0.330704,
\end{align*}
\]

and

\[
\text{root} = 0.5 - \frac{2(0.330704)}{2.12319 + \sqrt{(2.12319)^2 - 4(-1.07644)(0.330704)}} = 0.354914.
\]

For the next iteration, we have

\[
\begin{align*}
\quad x_0 &= 0.354914, \quad f(x_0) = -0.0138066 & h_1 &= 0.145086, \\
\quad x_1 &= 0.5, \quad f(x_1) = 0.330704 & h_2 &= 0.354914, \\
\quad x_2 &= 0, \quad f(x_2) = -1 & \gamma &= 2.44623.
\end{align*}
\]

Then

\[
\begin{align*}
\quad a &= \frac{(2.44623)(0.330704) - (-0.0138066)(3.44623) + (-1)}{2.44623(0.145086)^2(3.44623)} = -0.808314, \\
b &= \frac{0.330704 - (-0.0138066) - (-0.808314)(0.145086)^2}{0.145086} = 2.49180, \\
c &= -0.0138066,
\end{align*}
\]

\[
\text{root} = 0.354914 - \frac{2(-0.0138066)}{2.49180 + \sqrt{(2.49180)^2 - 4(-0.808314)(-0.0138066)}} = 0.360465.
\]

After a third iteration, we get 0.3604217 as the value for the root, which is identical to that from Newton’s method after three iterations.
1.5 **Fixed-Point Iteration / \( x = g(x) \) Method** (固定點疊代法)

- *Fixed-point iteration* can be a useful way to get a root of \( f(x) = 0 \). To use the method, we rearrange \( f(x) = 0 \) into an equivalent form \( x = g(x) \), which usually can be done in several ways. Whenever we have \( r = g(r) \), \( r \) is said to be a fixed point for the function \( g \). The iterative form

\[
x_{n+1} = g(x_n), \quad n=0, 1, 2, \ldots,
\]

converges to the fixed point \( r \), a root of \( f(x) \).

Example: Find the roots of \( f(x) = x^2 - 2x - 3 = 0 \).

It is easy to show the two roots at \( x = -1 \) and \( x = 3 \).

1) Suppose we rearrange to give this equivalent form:

\[
x = g_1(x) = \sqrt{2x + 3}.
\]

If we start with \( x = 4 \) and iterate with the fixed-point algorithm, we have

\[
x_0 = 4, \\
x_1 = \sqrt{11} = 3.31662, \\
x_2 = \sqrt{9.63325} = 3.10375, \\
x_3 = \sqrt{9.20750} = 3.03439, \\
x_4 = \sqrt{9.06877} = 3.01144, \\
x_5 = \sqrt{9.02288} = 3.00381,
\]

and it appears that the values are converging on the root at \( x = 3 \).

2) Another arrangement of \( f(x) \) is

\[
x = g_2(x) = \frac{3}{x - 2}.
\]

If we start with \( x = 4 \), we have

\[
x_0 = 4, \\
x_1 = 1.5, \\
x_2 = -6, \\
x_3 = -0.375, \\
x_4 = -1.263158,
\]
\( x_5 = -0.919355, \)
\( x_6 = -1.02762, \)
\( x_7 = -0.990876, \)
\( x_8 = -1.00305, \)

and it seems that the values are converging to the other root at \( x = -1 \). We also see that the convergence is oscillatory rather than monotonic as we saw in the first case.

3) Consider a third rearrangement:
\[
x = g_3(x) = \frac{(x^2 - 3)}{2}.
\]
Starting with \( x = 4 \), we have
\[
x_0 = 4, \]
\( x_1 = 6.5, \)
\( x_2 = 19.625, \)
\( x_3 = 191.070, \)

and the iterates are obviously diverging.

It appears that the different behaviors (convergence or divergence) depend on whether the magnitude of the slope of the curve \( y = g(x) \) is less or greater than 1. (當函數 \( g(x) \)在交叉點的切線斜率小於 1 時，疊代會收斂；而大於 1 時，會發散)

- **Algorithm for the Fixed-Point Method:**
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Given \( x_i \) reasonably close to the root, 
  Rearrange the equation to an equivalent form \( x = g(x) \). 
  Repeat
  - Set \( x_2 = x_1 \)
  - Set \( x_1 = g(x_1) \)
  Until ( \( |x_i - x_{i-1}| < TOL \) )
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

- The fixed-point method converges at a **linear** rate. It is said to linearly convergent, meaning that the error at each successive iteration is a constant fraction of the previous error. (Actually, this is true only as the errors approach zero.)
If \( g(x) = \sqrt{2x + 3} \)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Error</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.31662</td>
<td>0.31662</td>
</tr>
<tr>
<td>2</td>
<td>0.10375</td>
<td>0.32767</td>
</tr>
<tr>
<td>3</td>
<td>0.03439</td>
<td>0.33143</td>
</tr>
<tr>
<td>4</td>
<td>0.01144</td>
<td>0.33270</td>
</tr>
<tr>
<td>5</td>
<td>0.00381</td>
<td>0.33312</td>
</tr>
<tr>
<td>6</td>
<td>-0.00126</td>
<td>0.34254</td>
</tr>
<tr>
<td>7</td>
<td>0.00012</td>
<td>0.33029</td>
</tr>
<tr>
<td>8</td>
<td>-0.00005</td>
<td>0.33435</td>
</tr>
</tbody>
</table>

If \( g(x) = \frac{3}{x-2} \)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Error</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.50000</td>
<td>0.50000</td>
</tr>
<tr>
<td>2</td>
<td>-5.00000</td>
<td>-2.00000</td>
</tr>
<tr>
<td>3</td>
<td>0.62500</td>
<td>-0.12500</td>
</tr>
<tr>
<td>4</td>
<td>-0.26316</td>
<td>-0.42105</td>
</tr>
<tr>
<td>5</td>
<td>0.08065</td>
<td>-0.30645</td>
</tr>
<tr>
<td>6</td>
<td>-0.00126</td>
<td>-0.34254</td>
</tr>
<tr>
<td>7</td>
<td>0.00012</td>
<td>0.33029</td>
</tr>
<tr>
<td>8</td>
<td>-0.00005</td>
<td>0.33435</td>
</tr>
</tbody>
</table>

Notice that the magnitudes of the ratios seem to be leveling out at 0.3333.

**Method of Aitken Acceleration** (Aitken 加速法)

- For any iterative process where the errors decrease proportionally, we can speed the convergence by a technique known as **Aitken acceleration**.

Let \( \frac{e_{n+2}}{e_{n+1}} = \frac{e_{n+1}}{e_n} \),

\[
R - x_{n+2} = \frac{R - x_{n+1}}{R - x_n}, \quad \text{where } R \text{ is the true value for the root.}
\]

\[
R^2 - R(x_n + x_{n+2}) + x_n \cdot x_{n+2} = R^2 - R \cdot 2x_{n+1} + x_{n+1}^2
\]

\[
R = \frac{x_n \cdot (x_n - 2x_{n+1} + x_{n+2}) - (x_{n+1} - x_n)^2}{x_n - 2x_{n+1} + x_{n+2}} = \frac{x_n - (x_{n+1} - x_n)^2}{x_n - 2x_{n+1} + x_{n+2}}
\]

\[
R = \frac{x_n - \Delta x_n^2}{\Delta x_{n+1} - \Delta x_n} = x_n - \frac{\Delta x_n^2}{\Delta^2 x_n}
\]

where \( \Delta x_n = x_{n+1} - x_n \), \( \Delta^2 x_n = \Delta x_{n+1} - \Delta x_n \).

Applying to the example \( x = g_1(x) = \sqrt{2x + 3} \), we have

<table>
<thead>
<tr>
<th>( x )</th>
<th>( \Delta x )</th>
<th>( \Delta^2 x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.31662</td>
<td>0.68338</td>
<td>0.47051</td>
</tr>
<tr>
<td>3.31662</td>
<td>0.21287</td>
<td>0.47051</td>
</tr>
<tr>
<td>3.10375</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
R = 4 - \frac{0.68338^2}{0.47051} = 3.00744 , \text{ which is close to the root } x = 3.
\]
1.6 Multiple Roots

- The methods we describe do not work well for multiple roots. And a slow convergence (收敛慢) is resulted usually.

- Another disadvantage to using these methods to find multiple roots is the imprecision (不精确). Because the curve is “flat” (平坦的) in the neighborhood of the root, there is a “neighborhood of uncertainty” around the root where values of \( f(x) \) are very small. Using double precision (使用倍精度浮點數來計算) will decrease the neighborhood of uncertainty.

- Whenever we want to find roots that are near \( f'(x) = 0 \), we are in trouble. It is strongly recommended that you graph the function (繪製函數圖形), before jumping into a root-finding routine, to see in advance whether such problems will arise.

- MATLAB’s ‘vpa’ command can give as much precision as desired, even to 100 significant figures, so this “neighborhood” can be very small.

Example: Graph the function \( f(x) = 2 \sin x - \frac{1}{4} e^x - 1 \).

```c
#include <stdio.h>
#include <math.h>
int main()
{
    float x, y, x1 = -10., x2 = 2.;
    int Num = 100, i;
    FILE *fPtr;

    fPtr = fopen( "output.dat" , "w" );
    fprintf( fPtr , " x       y\n" );
```
```c
for ( i = 0 ; i <= Num ; i++ ) {
    x = x1 + i * ( x2 - x1 ) / Num;
    y = 2.0 * sin( x ) - 0.25 * exp( x ) - 1.0;
    fprintf( fPtr, "%f \n", x, y );
}
fclose( fPtr );
return 0;
}
```

執行結果—

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10.000000</td>
<td>0.088031</td>
</tr>
<tr>
<td>-9.880000</td>
<td>-0.120689</td>
</tr>
<tr>
<td>-9.760000</td>
<td>-0.342056</td>
</tr>
<tr>
<td>-9.640000</td>
<td>-0.572887</td>
</tr>
<tr>
<td>-9.520000</td>
<td>-0.809861</td>
</tr>
<tr>
<td>-9.400000</td>
<td>-1.049572</td>
</tr>
</tbody>
</table>

• x-y 繪圖軟體 Grapher v.1.27 之使用：\[ y = 2 \sin x - \frac{1}{4} e^x - 1 \]
2 Solving Sets of Equations (求解聯立方程組)

- Linear equations are the basis for mathematical models of a variety of applications. Solving sets of linear equations is the most frequently used numerical procedure when real-world situations are modeled. The methods for solving ordinary and partial differential equations depend on them.

2.1 Matrices and Vectors (矩陣與向量)

- When a system of equations has more than two or three equations, it is difficult to discuss them without using matrices (矩陣) and vectors (向量).

A matrix of $n$ rows and $m$ columns is said to be $n \times m$. The first subscript $n$ denotes the row, and the second subscript $m$ denotes the column. Capital letters are used to refer to matrices. For example,

$$
A = \begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1m} \\
a_{21} & a_{22} & \cdots & a_{2m} \\
\vdots \\
a_{n1} & a_{n2} & \cdots & a_{nm}
\end{bmatrix} = [a_{ij}], \quad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, m.
$$

Two matrices of the same size may be added or subtracted.

Multiplication of two matrices:

$$
A_{n \times m} \times B_{m \times r} = C_{n \times r} = [c_{ij}] = \sum_{k=1}^{m} a_{ik}b_{kj}, \quad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, r.
$$

In general $AB \neq BA$

- The definition of matrix multiplication permits us to write the set of linear equations

$$
\begin{align*}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m &= b_2 \\
\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m &= b_n
\end{align*}
$$

to a compact matrix form as

$$
Ax = b
$$

where
\[
A = \begin{bmatrix}
  a_{11} & a_{12} & \cdots & a_{1m} \\
  a_{21} & a_{22} & \cdots & a_{2m} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{n1} & a_{n2} & \cdots & a_{nm}
\end{bmatrix}, \quad x = \begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_n
\end{bmatrix}, \quad b = \begin{bmatrix}
  b_1 \\
  b_2 \\
  \vdots \\
  b_n
\end{bmatrix}.
\]

- For a square matrix, the elements on the main diagonal are those from the upper-left to the lower-right corner. If all elements except those on the diagonal are zero, the matrix is called a **diagonal matrix** (對角矩陣). If the nonzero elements of a diagonal matrix all are equal to one, the matrix is called the **identity matrix** (單位矩陣) of order \(n\).

The **transpose** (轉置) of a matrix is the matrix obtained by writing the rows as columns or by writing the columns as rows.

When a matrix is square, a quantity called its trace is defined. The **trace** (跡) of a square matrix is the sum of the elements on its main diagonal. For any matrix, the sum of its eigenvalues equals to its trace.

If all the elements above the diagonal are zero, a matrix is called **lower-triangular** (下三角); if all the elements below the diagonal are zero, a matrix is called **upper-triangular** (上三角). They are of special importance in solving sets of linear equations. If a matrix is **triangular**, its **eigenvalues** (特徵值) are equal to the diagonal elements.

**Tridiagonal matrices** (三對角矩陣) are those that have nonzero elements only on the diagonal and in the positions adjacent to the diagonal. They are special important in certain partial differential equations.

For a tridiagonal matrix, only the nonzero values need to be recorded, and that means that the \(n \times n\) matrix can be compressed into a matrix of 3 columns and \(n\) rows.

**Division** of matrices is not defined. (矩陣的除法沒有定義)

- The **determinant** (行列式) of a square matrix is a number. Calculating determinant may require a lot of work if the matrix is of large size. Much better ways to get the determinant will be described in Section 2.2.
• **Meanings of Eigenvalue and Eigenvector** (特徵值與特徵向量之意義)

**Stretching of an elastic membrane**

An elastic membrane in the $x_1x_2$-plane with boundary circle $x_1^2 + x_2^2 = 1$ (Fig. 145) is stretched so that a point $P: (x_1, x_2)$ goes over into the point $Q: (y_1, y_2)$ given by

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = Ax = \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}; \quad \text{in components,} \quad y_1 = 5x_1 + 3x_2, \quad y_2 = 3x_1 + 5x_2.$$

Find the principal directions, that is, the directions of the position vector $\mathbf{x}$ of $P$ for which the direction of the position vector $\mathbf{y}$ of $Q$ is the same or exactly opposite. What shape does the boundary circle take under this deformation?

![Fig. 145. Undeformed and deformed membrane in Example 1](image)

**Solution.** We are looking for vectors $\mathbf{x}$ such that $\mathbf{y} = \lambda \mathbf{x}$. Since $\mathbf{y} = A\mathbf{x}$, this gives $A\mathbf{x} = \lambda \mathbf{x}$, an equation of the form (1), an eigenvalue problem. In components, $A\mathbf{x} = \lambda \mathbf{x}$ is

$$\begin{align*}
5x_1 + 3x_2 &= \lambda x_1, \\
3x_1 + 5x_2 &= \lambda x_2.
\end{align*}$$

(2)

The characteristic equation is

$$\begin{vmatrix}
5 - \lambda & 3 \\
3 & 5 - \lambda
\end{vmatrix} = (5 - \lambda)^2 - 9 = 0.$$

(3)

Its solutions are $\lambda_1 = 8$ and $\lambda_2 = 2$. These are the eigenvalues of our problem. For $\lambda = \lambda_1 = 8$, our system (2) becomes

$$\begin{align*}
-3x_1 + 3x_2 &= 0, \quad \text{Solution } x_2 = x_1. \quad x_1 \text{ arbitrary,}
\\
3x_1 - 3x_2 &= 0. \quad \text{for instance, } x_1 = x_2 = 1.
\end{align*}$$

For $\lambda_2 = 2$, our system (2) becomes

$$\begin{align*}
3x_1 + 3x_2 &= 0, \quad \text{Solution } x_2 = -x_1. \quad x_1 \text{ arbitrary,}
\\
3x_1 + 3x_2 &= 0. \quad \text{for instance, } x_1 = 1, x_2 = -1.
\end{align*}$$

We thus obtain as eigenvectors of $A$, for instance,

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ corresponding to } \lambda_1; \quad \begin{bmatrix} 1 \\ -1 \end{bmatrix} \text{ corresponding to } \lambda_2;$$

(or a nonzero scalar multiple of these). These vectors make $45^\circ$ and $135^\circ$ angles with the positive $x_1$-direction. They give the principal directions, the answer to our problem. The eigenvalues show that in the principal directions the membrane is stretched by factors 8 and 2, respectively; see Fig. 145.
2.2 Elimination Methods (消去法)

• Cramer’s rule
  Let \(|A| \neq 0\). Then the unique solution of \(AX = B\) is
  \[x_k = \frac{|A(k;B)|}{|A|}, \quad k = 1, 2, 3, \ldots, n,\]
  where \(A(k;B)\) is the matrix obtained from \(A\) by replacing column \(k\) of \(A\) by \(B\). It used to be that students were taught to use Cramer’s rule, in which a system can be solved through the use of determinants.

  However, Cramer’s rule is inefficient (無效率的) and is almost impossible to use if the system of equations is large.

• The elementary row operation (基本列運算) can be used in solving a linear system.
  There are three of these operations:
  1. We may multiply any row of the coefficient matrix by a constant.
  2. We can add a multiple of one row to a multiple of any other row.
  3. We can interchange the order of any two rows.

• Transfer a coefficient matrix to a triangular one is helpful for solving a system of equations. The first objective of the elimination method is to change the matrix of coefficients so that it is triangular. As an example,

  \[
  \begin{align*}
  4x_1 - 2x_2 + x_3 &= 15 \\
  -3x_1 - x_2 + 4x_3 &= 8 \\
  x_1 - x_2 + 3x_3 &= 13
  \end{align*}
  \]

  \[
  \begin{align*}
  3 \times R_1 + 4 \times R_2 & \Rightarrow \quad 4x_1 - 2x_2 + x_3 = 15 \\
  \quad (-1) \times R_1 + 4 \times R_3 & \Rightarrow \quad -10x_2 + 19x_3 = 77 \\
  \quad 2 \times R_2 + (-10) \times R_3 & \Rightarrow \quad -2x_2 + 11x_3 = 37
  \end{align*}
  \]

  Then back-substitution (代回) will give the values of \(x_1, x_2\) and \(x_3\).
The procedures can be presented in matrix notation:

\[
\begin{bmatrix}
4 & -2 & 1 & 15 \\
-3 & -1 & 4 & 8 \\
1 & -1 & 3 & 13 \\
\end{bmatrix}
\]

\[
3 \times R_1 + 4 \times R_2 \Rightarrow \begin{bmatrix} 4 & -2 & 1 & 15 \\ 0 & -10 & 19 & 77 \end{bmatrix}
\]

\[
(-1) \times R_1 + 4 \times R_3 \Rightarrow \begin{bmatrix} 4 & -2 & 1 & 15 \\ 0 & -2 & 11 & 37 \end{bmatrix}
\]

\[
2 \times R_2 + (-10) \times R_3 \Rightarrow \begin{bmatrix} 4 & -2 & 1 & 15 \\ 0 & 0 & -72 & -216 \end{bmatrix}
\]

The essence (本質) of any elimination method is to reduce the coefficient matrix to a triangular one and then use back-substitution to get the solution.

**Gaussian Elimination** (高斯消去法)

- The procedure just described has a major problem. Observe that the transformed coefficients can become very large as we convert to a triangular system. The method called **Gaussian elimination** avoids this by subtracting \( a_{ij}/a_{ii} \) times the first equation from the \( i \)th equation to make the transformed numbers in the first column equal to zero, and doing similarly for the rest of the columns. Here is an example:

\[
\begin{bmatrix}
4 & -2 & 1 & 15 \\
-3 & -1 & 4 & 8 \\
1 & -1 & 3 & 13 \\
\end{bmatrix}
\]

\[
R_2 - (-\frac{3}{4})R_1 \Rightarrow \begin{bmatrix} 4 & -2 & 1 & 15 \\ 0 & -2.5 & 4.75 & 19.25 \end{bmatrix}
\]

\[
R_3 - (\frac{1}{3})R_1 \Rightarrow \begin{bmatrix} 4 & -2 & 1 & 15 \\ 0 & -0.5 & 2.75 & 9.25 \end{bmatrix}
\]

\[
R_3 - (\frac{1}{5})R_2 \Rightarrow \begin{bmatrix} 4 & -2 & 1 & 15 \\ 0 & 0 & 1.80 & 5.40 \end{bmatrix}
\]

- We must always guard against dividing by zero. A useful strategy to avoid (if possible) such zero divisors, called **pivoting** (轉動), is to rearrange the equations so as to put the coefficient of largest magnitude on the diagonal at each step.
• When there are large differences in magnitude of coefficients in one equation compared to the other equations, we may need to scale (放大缩小) the values.

• In certain cases, the coefficients are such that the results are particularly sensitive to round-off errors, such systems are called ill-conditioned (不良条件).

**LU Decomposition** (LU 分解法)

- In process of forming an upper-triangular coefficient matrix, if at each stage we store the ratio of coefficients in place of zero, our final form would be

\[
\begin{bmatrix}
4 & -2 & 1 & 15 \\
-\frac{3}{4} & -2.5 & 4.75 & 19.25 \\
\frac{1}{4} & \frac{1}{4} & 1.80 & 5.40
\end{bmatrix}
\]

Let \( A = \begin{bmatrix} 4 & -2 & 1 \\ -3 & -1 & 4 \\ 1 & -1 & 3 \end{bmatrix} \), \( L = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{3}{4} & 1 & 0 \\ \frac{1}{4} & \frac{1}{4} & 1 \end{bmatrix} \) and \( U = \begin{bmatrix} 4 & -2 & 1 \\ 0 & -2.5 & 4.75 \\ 0 & 0 & 1.80 \end{bmatrix} \).

It is found that the matrix \( A \) can be decomposed into a lower-triangular matrix \( L \) and an upper-triangular matrix \( U \). That is

\[ A = L \cdot U \]

or

\[
\begin{bmatrix}
4 & -2 & 1 \\
-3 & -1 & 4 \\
1 & -1 & 3
\end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{3}{4} & 1 & 0 \\ \frac{1}{4} & \frac{1}{4} & 1 \end{bmatrix} \begin{bmatrix} 4 & -2 & 1 \\ 0 & -2.5 & 4.75 \\ 0 & 0 & 1.80 \end{bmatrix}
\]

This procedure is called a **LU decomposition** of \( A \).

Recall that \( \det(A) = \det(L \cdot U) = \det(L) \cdot \det(U) = 1 \cdot \det(U) = \det(U) = 4 \cdot (-2.5) \cdot (1.8) = -18 \).

From this example, we see that Gaussian elimination does the following:

1. It finds the solution of the system of equations.
2. It computes the determinant of a matrix very efficiently.
3. It can provide us with the **LU decomposition** of the matrix of coefficients, in the sense that the product of the two matrices, \( L \cdot U \), may give us a permutation of the rows of the original matrix.
• **Algorithm for Gaussian Elimination**

To solve a system of $n$ linear equations: $Ax = b$.  

For $j = 1$ To $(n - 1)$

\[ \text{pvt} = |a[j,j]| \]

\[ \text{pivot} \{ j \} = j \]

\[ \text{ipvt\_temp} = j \]

For $i = j + 1$ To $n$ \hspace{1em} (Find pivot row)

IF $|a[i,j]| > \text{pvt}$ \hspace{1em} Then

\[ \text{pvt} = |a[i,j]| \]

\[ \text{ipvt\_temp} = i \]

End IF

End For $i$

(Switch rows if necessary)

IF pivot $\{ j \} \not< > \text{ipvt\_temp}$

[switch rows(rows $j$ and ipvt\_temp)]

End For $i$

(Store multipliers)

\[ a[i,j] = a[i,j] / a[j,j] \]

End For $i$

(Create zeros below the main diagonal)

For $i = j + 1$ To $n$

For $k = j + 1$ To $n$

\[ a[i,k] = a[i,k] - a[i,j] \ast a[j,k] \]

End For $k$

\[ b[i] = b[i] - a[i,j] \ast b[j] \]

End For $i$

End For $j$;

(Back Substitution Part)

\[ x[n] = b[n] / a[n,n] \]

For $j = n - 1$ Down To 1

\[ x[j] = b[j] \]

For $k = n$ Down To $j + 1$

\[ x[j] = x[j] - x[k] \ast a[j,k] \]

End For $k$

\[ x[j] = x[j] / a[j,j] \]

End For $j$.

• Interchanging rows in a large matrix can be expensive. There is a better way. That is, we keep track of the order of the rows in an *order vector* and, when a row change is indicated, we only interchange the corresponding elements in the order vector. Using an order vector saves computer time because only two numbers of this vector are interchanged; we do not have to switch all the elements of the two rows.

• The algorithm for Gaussian elimination will be clarified by an additional numerical example. Solve the following system of equations using Gaussian elimination.
Back-substitution gives

\[
\begin{align*}
R_1 \leftrightarrow R_4 & \Rightarrow \\
R_2 \leftrightarrow R_3 & \Rightarrow \\
\end{align*}
\]

\[
\begin{bmatrix}
0 & 2 & 0 & 1 & 0 \\
6 & 1 & -6 & -5 & 6 \\
2 & 2 & 3 & 2 & -2 \\
4 & -3 & 0 & 1 & -7 \\
6 & 1 & -6 & -5 & 6
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 2 & 0 & 1 & 0 \\
6 & 1 & -6 & -5 & 6 \\
2 & 2 & 3 & 2 & -2 \\
4 & -3 & 0 & 1 & -7 \\
0 & 2 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
6 & 1 & -6 & -5 & 6 \\
0 & 1.6667 & 5 & 3.6667 & -4 \\
0 & -3.6667 & 4 & 4.3333 & -11 \\
0 & 2 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
6 & 1 & -6 & -5 & 6 \\
0 & 1.6667 & 5 & 3.6667 & -4 \\
0 & -3.6667 & 4 & 4.3333 & -11 \\
0 & 2 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
6 & 1 & -6 & -5 & 6 \\
0 & 1.6667 & 5 & 3.6667 & -4 \\
0 & -3.6667 & 4 & 4.3333 & -11 \\
0 & 2 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
6 & 1 & -6 & -5 & 6 \\
0 & 1.6667 & 5 & 3.6667 & -4 \\
0 & -3.6667 & 4 & 4.3333 & -11 \\
0 & 2 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
6 & 1 & -6 & -5 & 6 \\
0 & 1.6667 & 5 & 3.6667 & -4 \\
0 & -3.6667 & 4 & 4.3333 & -11 \\
0 & 2 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
6 & 1 & -6 & -5 & 6 \\
0 & 1.6667 & 5 & 3.6667 & -4 \\
0 & -3.6667 & 4 & 4.3333 & -11 \\
0 & 2 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
6 & 1 & -6 & -5 & 6 \\
0 & 1.6667 & 5 & 3.6667 & -4 \\
0 & -3.6667 & 4 & 4.3333 & -11 \\
0 & 2 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
6 & 1 & -6 & -5 & 6 \\
0 & 1.6667 & 5 & 3.6667 & -4 \\
0 & -3.6667 & 4 & 4.3333 & -11 \\
0 & 2 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
6 & 1 & -6 & -5 & 6 \\
0 & 1.6667 & 5 & 3.6667 & -4 \\
0 & -3.6667 & 4 & 4.3333 & -11 \\
0 & 2 & 0 & 1 & 0
\end{bmatrix}
\]

Back-substitution gives

\[
\begin{align*}
x_4 &= -1.9999 \\
x_3 &= 0.33325 \\
x_2 &= 1.0000 \\
x_1 &= -0.50000
\end{align*}
\]

The correct answers are \( x_4 = -2 \), \( x_3 = \frac{1}{3} \), \( x_2 = 1 \) and \( x_4 = -\frac{1}{2} \). In this calculation we have carried five significant digits and rounded each calculation. Even so, we do not have five-digit accuracy in the answers. The discrepancy (不一致處) is due to round off.

- In the following discussion we will discuss how to minimize the effects of round off and avoid conditions that can cause round-off errors to be magnified.
Replace the zeros below the main diagonal with the ratio of coefficients at each step, the resulting augmented matrix would be

\[
\begin{bmatrix}
6 & 1 & -6 & -5 & 6 \\
(0.66667) & -3.6667 & 4 & 4.3333 & -11 \\
(0.33333) & (-0.45454) & 6.8182 & 5.6364 & -9.0001 \\
(0.0) & (-0.54545) & (0.32) & 1.5600 & -3.1199
\end{bmatrix}
\]

This gives an LU decomposition as

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0.66667 & 1 & 0 & 0 & 0 \\
0.33333 & -0.45454 & 1 & 0 & 0 \\
0.0 & -0.54545 & 0.32 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
6 & 1 & -6 & -5 \\
-3.6667 & 4 & 4.3333 \\
6.8182 & 5.6364 \\
1.5600 & 3.1199
\end{bmatrix}
\]

MATLAB gets a more accurate solution:

```matlab
>> A = [ 0 2 0 1 ; 2 2 3 2 ; 4 -3 0 1 ; 6 1 -6 -5 ];
>> b = [ 0 -2 -7 6 ]';
>> A\b
ans =
   -0.5000
    1.0000
    0.3333
   -2.0000
```

**Gauss-Jordan Elimination**

- There are many variants to the Gaussian elimination scheme. One variant sometimes used is the **Gauss-Jordan** scheme. In it, the elements above the diagonal are made zero at the same time that zeros are created below the diagonal. Usually, the diagonal elements are made ones at the same time that the reduction is performed. That is, the coefficient matrix is transformed into the identity matrix. When this has been accomplished, the column of the right-hand sides has been transformed into the solution vector.

Pivoting is normally employed to preserve arithmetic accuracy.

For the previous example:
The fifth column is now the solution. It differs slightly from that obtained with Gaussian elimination because round-off errors have been entered in a different way.

**Operational Count (運算量計算)**

- The efficiency of a numerical procedure is ordinarily measured by counting the number of arithmetic operations (算數運算) that are required. In the past, only multiplication and divisions were counted because they used to take much longer to perform than additions
and subtractions. In today’s computers using math processors, all four of these take about the same time, so we should count them all.

If \( n \) denotes the number of equations to be solved, then a total of \( \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{2}{5}n \) operations (including addition, subtraction, multiplication and division) for Gaussian elimination and \( n^3 + n^2 - 2n \) operations for Gauss-Jordan elimination are required.

The method of Gauss-Jordan elimination really requires almost 50% more operations than that of the Gaussian elimination.

**Scaling**

- When some rows have coefficients that are very large in comparison to those in other rows, round-off errors may result in solution inaccuracy. A technique, called *scaling*, can be used to reduce the errors. The term “scaling” means to adjust the coefficients to make the largest in each row of the same magnitude.

Coefficients may differ in magnitude for several reasons. It might be caused by relations where the quantities are in widely different units: microvolts versus kilovolts, seconds versus years, for example. It could be due to inherently large numbers in just one equation. Here is an example: given

\[
A = \begin{bmatrix} 3 & 2 & 100 \\ -1 & 3 & 100 \\ 1 & 2 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 105 \\ 102 \\ 2 \end{bmatrix}
\]

whose correct answer obviously is \( x = [1.00, 1.00, 1.00]^T \).

If we solve this by partial pivoting but with only three digits of precision to emphasize round-off error, we get the augmented matrix after triangularization:

\[
\begin{bmatrix} 3 & 2 & 100 & 105 \\ 0 & 3.67 & 133 & 135 \\ 0 & 0 & -82.4 & -82.6 \end{bmatrix}
\]

from which the solution vector is readily found to be the erroneous value of \( x = [0.939, 1.09, 1.00]^T \). The trouble here is that the coefficients in the third equation are much smaller than those in the other two. Doing scaling in the original equations gives (still using just three digits):

- 38 -
\[
0.01 \times R_1 \quad \Rightarrow \quad \begin{bmatrix} 0.03 & 0.02 & 1 \end{bmatrix} \quad 1.05
\]
\[
0.01 \times R_2 \quad \Rightarrow \quad \begin{bmatrix} -0.01 & 0.03 & 1 \end{bmatrix} \quad 1.02
\]
\[
0.5 \times R_3 \quad \Rightarrow \quad \begin{bmatrix} 0.5 \quad 1 \quad -0.5 \quad 1 \end{bmatrix}
\]

\[
R_1 \iff R_3 \quad \Rightarrow \quad \begin{bmatrix} 0.5 \quad 1 \quad -0.5 \quad 1 \end{bmatrix}
\]
\[
0.02 \times R_1 + R_2 \quad \Rightarrow \quad \begin{bmatrix} 0 \quad 0.05 \quad 0.99 \quad 1.04 \end{bmatrix}
\]
\[
-0.06 \times R_1 + R_3 \quad \Rightarrow \quad \begin{bmatrix} 0 \quad -0.04 \quad 1.03 \quad 0.99 \end{bmatrix}
\]
\[
4 \times R_2 + 5 \times R_3 \quad \Rightarrow \quad \begin{bmatrix} 0 \quad 0 \quad 9.01 \quad 9.01 \end{bmatrix}
\]

and back substitution gives the correct answer: \( x = [1.00, 1.00, 1.00] \).

**Using the LU matrix for Multiple Right-Hand Sides**

- Many physical situations are modeled with a large set of linear equations: an example is determining the internal temperatures in a nuclear reactor, and knowing the maximum temperature is critical. The equations will depend on the geometry and certain external factors that will determine the right-hand sides.

- If we want the solution for many different values of these right-hand sides, it is inefficient to solve the system from the start with each one of the right-hand-side values. In this case using the LU equivalent of the coefficient matrix is preferred.

\[
Ax = b
\]

\[
A = LU \quad \Rightarrow \quad LUx = b
\]

Let \( Ux = y \quad \Rightarrow \quad Ly = b \)

It is easy to solve \( y \) by forward-substitution, and then solve \( x \) by back-substitution.

The operational count for either forward- or back-substitution is exactly \( n^2 \) operations, so solving \( Ax = b \) will take only \( 2n^2 \) operations if the \( LU \) equivalent of \( A \) is already known. Note that \( 2n^2 \) is significantly less than \( \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n \) that is required by the method of Gaussian elimination.
**Homework #3**

Use the subprograms provided in Numerical Recipes to solve the system

\[
\begin{align*}
4.63x_1 - 1.21x_2 + 3.22x_3 &= 2.22, \\
-3.07x_1 + 5.48x_2 + 2.11x_3 &= -3.17, \\
1.26x_1 + 3.11x_2 + 4.57x_3 &= 5.11.
\end{align*}
\]

a) by using the Gaussian-Jordan elimination method;
b) by using the *LU* decomposition method.

### 2.3 The Inverse of a Matrix (反矩陣)

- One way to find the inverse of matrix *A* is to employ the minors of its determinant, but this is not efficient. The *Cramer’s formula* is

\[
A^{-1} = \frac{1}{\det A} \begin{bmatrix}
A_{11} & A_{21} & \cdots & A_{n1} \\
A_{12} & A_{22} & \cdots & A_{n2} \\
\vdots & \vdots & \ddots & \vdots \\
A_{1n} & A_{2n} & \cdots & A_{nn}
\end{bmatrix},
\]

where \(A_{ij}\) is the minors of \(A\).

- **Example 2.2** Given matrix *A*, the Gauss-Jordan method can be used to find its inverse.

\[
A = \begin{bmatrix}
1 & -1 & 2 \\
3 & 0 & 1 \\
1 & 0 & 2
\end{bmatrix}
\]

Augment *A* with the identity matrix and then reduce:

\[
\begin{bmatrix}
1 & -1 & 2 & | & 1 & 0 & 0 \\
3 & 0 & 1 & | & 0 & 1 & 0 \\
1 & 0 & 2 & | & 0 & 0 & 1
\end{bmatrix} \Rightarrow
\begin{bmatrix}
1 & -1 & 2 & | & 1 & 0 & 0 \\
0 & 3 & -5 & | & -3 & 1 & 0 \\
0 & 1 & 0 & | & -1 & 0 & 1
\end{bmatrix}
\]

\[
R_2 \leftrightarrow R_3 \Rightarrow
\begin{bmatrix}
1 & -1 & 2 & | & 1 & 0 & 0 \\
0 & 1 & 0 & | & -1 & 0 & 1 \\
0 & 3 & -5 & | & -3 & 1 & 0
\end{bmatrix}
\]

\[
-3 \times R_2 + R_3 \Rightarrow
\begin{bmatrix}
1 & -1 & 2 & | & 1 & 0 & 0 \\
0 & 1 & 0 & | & -1 & 0 & 1 \\
0 & 0 & -5 & | & 0 & 1 & -3
\end{bmatrix}
\]
\[ R_1 + \frac{2}{5} R_3 \Rightarrow \begin{bmatrix} 1 & -1 & 0 & 1 & \frac{2}{5} & -\frac{6}{5} \\ 0 & 1 & 0 & -1 & 0 & 1 \end{bmatrix} \]
\[ -\frac{1}{5} R_3 \Rightarrow \begin{bmatrix} 0 & 0 & 1 & 0 & -\frac{1}{5} & \frac{3}{5} \end{bmatrix} \]
\[ R_1 + R_2 \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & \frac{2}{5} & -\frac{1}{5} \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & -\frac{1}{5} & \frac{3}{5} \end{bmatrix} \]
\[ \Rightarrow A^{-1} = \begin{bmatrix} 0 & \frac{2}{5} & \frac{1}{5} \\ -1 & 0 & 1 \\ 0 & -\frac{1}{5} & \frac{3}{5} \end{bmatrix} \]

- If we have the inverse of a matrix, we can use it to solve a set of equations, \( Ax = b \):

\[ A^{-1}Ax = A^{-1}b \]
\[ \Rightarrow x = A^{-1}b \]

This would seem like a good way to solve equations, but is not the best way to solve a system. Getting the \( LU \) equivalent of \( A \) first and using the \( L \) and \( U \) to solve \( Ax = b \) requires only two back-substitutions.

**Singular Matrix** (奇異矩陣)

- The definition of a *singular matrix* is a matrix that *does not have an inverse*.

- Example  Consider the matrix

\[
A = \begin{bmatrix} 1 & -2 & 3 \\ 2 & 4 & -1 \\ -1 & -14 & 11 \end{bmatrix}
\]

\[ \gg A = \begin{bmatrix} 1 & -2 & 3 \\ 2 & 4 & -1 \\ -1 & -14 & 11 \end{bmatrix}; \]
\[ \gg \text{lu}(A) \]
\[ \text{ans} = \]
\[ \begin{array}{ccc} 2.0000 & 4.0000 & -1.0000 \\ -0.5000 & -12.0000 & 10.5000 \\ 0.5000 & 0.3333 & 0.0000 \end{array} \]

\[ \gg \text{inv}(A) \]
\[ \text{Warning: Matrix is singular to working precision.} \]
\[ \text{Ans} = \]
\[ \begin{array}{ccc} \text{Inf} & \text{Inf} & \text{Inf} \\ \text{Inf} & \text{Inf} & \text{Inf} \\ \text{Inf} & \text{Inf} & \text{Inf} \end{array} \]
Here are five other ways to see if a matrix is singular.

1. A singular matrix has a **determinant** of zero.
2. The **rank** (秩) of the matrix is less the number of rows.
3. A singular matrix has **rows** that are **linearly dependent** vectors.
4. A singular matrix has **columns** that are **linearly dependent** vectors.
5. A set of equations with this coefficient matrix has **no unique solution**.

- A system is called **redundant** if the corresponding coefficient matrix has rows that are linearly dependent vectors. There are **infinite solutions** that satisfy the equations.

Example:

```plaintext
>> Ab = [ 1 -2 3 5 ; 2 4 -1 7 ; -1 -14 11 1 ]
```

```
Ab =
  1  -2  3  5
  2   4 -1  7
  1 -14 11  1
```

```plaintext
>> lu(Ab)
ans =
     2.0000   4.0000  -1.0000   7.0000
  (-0.5000) -12.0000  10.5000  4.5000
  (0.5000)  (0.3333)   0        0
```

A system is called **inconsistent** if there is **no solution** that satisfies the equations.

```plaintext
>> Ab = [ 1 -2 3 5 ; 2 4 -1 7 ; -1 -14 11 1 ]
```

```
Ab =
  1  -2  3  5
  2   4 -1  7
  1 -14 11  1
```

```plaintext
>> lu(Ab)
ans =
     2.0000   4.0000  -1.0000   7.0000
  (-0.5000) -12.0000  10.5000  4.5000
  (0.5000)  (0.3333)   0  -0.3333
```

In either case, there is no unique solution to a system with a singular coefficient matrix.
A comparison of singular and nonsingular matrices:

<table>
<thead>
<tr>
<th>For Singular Matrix $A$:</th>
<th>For Nonsingular Matrix $A$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>It has no inverse, $A^{-1}$</td>
<td>It has an inverse, $A^{-1}$ exists</td>
</tr>
<tr>
<td>Its determinant is zero</td>
<td>The determinant is nonzero</td>
</tr>
<tr>
<td>There is no unique solution to the system $Ax = b$</td>
<td>There is a unique solution to the system $Ax = b$</td>
</tr>
<tr>
<td>Gaussian elimination cannot avoid a zero on the diagonal</td>
<td>Gaussian elimination does not encounter a zero on the diagonal</td>
</tr>
<tr>
<td>The rank is less than $n$</td>
<td>The rank equals $n$</td>
</tr>
<tr>
<td>Rows are linearly dependent</td>
<td>Rows are linearly independent</td>
</tr>
<tr>
<td>Columns are linearly dependent</td>
<td>Columns are linearly independent</td>
</tr>
</tbody>
</table>

2.4 Ill-Conditioned Systems (條件不良的系統)

A system whose coefficient matrix is nearly singular is called **ill-conditioned**. When a system is ill-conditioned, the solution is very sensitive to changes in the right-hand vector. It is also sensitive to small changes in the coefficients.

Example 1: Consider

$$A = \begin{bmatrix} 3.02 & -1.05 & 2.53 \\ 4.33 & 0.56 & -1.78 \\ -0.83 & -0.54 & 1.47 \end{bmatrix}$$

The LU equivalent has a very small element in $A_{3,3}$:

$$LU = \begin{bmatrix} 4.33 & -0.56 & -1.78 \\ 0.6975 & -1.4406 & 3.7715 \\ -0.1917 & 0.3003 & -0.0039 \end{bmatrix}$$

The inverse of $A$:

$$A^{-1} = \begin{bmatrix} 5.6611 & -7.2732 & -18.5503 \\ 200.5046 & -268.2570 & -669.9143 \\ 76.8511 & -102.6500 & -255.8846 \end{bmatrix}$$

We see that $A^{-1}$ has elements very large in comparison to $A$.

Both of these results suggest that matrix $A$ is nonsingular but is “almost singular”.

Consider the system $Ax = b$.

If $b = [-1.61, 7.23, -3.38]^T$, then the solution is $x = [1.0000, 2.0000, -1.0000]^T$.

If $b = [-1.60, 7.23, -3.38]^T$, then the solution is $x = [1.0566, 4.0051, -0.2315]^T$.

If $b = [-1.60, 7.22, -3.38]^T$, then the solution is $x = [1.0727, 4.6826, 0.0265]^T$. 
The last two answers differ much from the first.

- Even a system of only two equations shows the effect of near singularity:

Example 2: Consider

\[
\begin{bmatrix} 1.01 & 0.99 \\ 0.99 & 1.01 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2.00 \\ 2.00 \end{bmatrix}
\]

The solution is clearly to be \( x = 1.00, \ y = 1.00 \).

However, if we make a small change to the \( b \)-vector, to \( [2.02, 1.98]^T \), the solution now is \( x = 2.00, \ y = 0.00 \); if we had another small change to the \( b \)-vector, to \( [1.98, 2.02]^T \), we would have \( x = 0.00, \ y = 2.00 \).

- It is helpful to think of the system, \( Ax = b \), as a linear system solver machine. For an ill-conditioned system, small changes in the input make large changes in the output.

- In some situations, one can transform the ill-conditioned problem into an equivalent set of equations that are not ill-conditioned. The efficiency of this scheme is related to the relative amount of computation required for the transformation, compared to the cost of doing the calculations in higher precision.

- *Another interesting phenomenon of an ill-conditioned system* is that we cannot test for the accuracy of the computed solution merely by substituting it into the equations to see whether the right-hand sides are reproduced.

Example 3: Consider

\[
A = \begin{bmatrix} 3.02 & -1.05 & 2.53 \\ 4.33 & 0.56 & -1.78 \\ -0.83 & -0.54 & 1.47 \end{bmatrix}, \ b = \begin{bmatrix} -1.61 \\ 7.23 \\ -3.38 \end{bmatrix}.
\]

The exact solution is \( x = [1, 2, -1]^T \). Of course, \( Ax = b \) is satisfied.

However, if we substitute a clearly erroneous vector \( \bar{x} = [0.880, -2.34, -2.66]^T \), we get \( A\bar{x} = [-1.6152, 7.2348, -3.3770]^T \), which is very close to \( b \).

**Effect of Precision**

- We have mentioned that it is difficult to get an accurate solution when a system is
ill-conditioned and have demonstrated that small changes in either the coefficients or the right-hand side make large changes in the solution. The solution is also depending on the accuracy of the arithmetic computations.

Example 4: Solve

\[
\begin{bmatrix}
3.02 & -1.05 & 2.53 & -1.61 \\
4.33 & 0.56 & -1.78 & 7.23 \\
-0.83 & -0.54 & 1.47 & -3.38
\end{bmatrix}
\]

If a precision of 10 digits is used, we have

\[
\begin{bmatrix}
1 & 0 & 0 & 1.000000037 \\
0 & 1 & 0 & 2.000001339 \\
0 & 0 & 1 & -0.999994882
\end{bmatrix}
\]

which is pretty close to the exact solution, \( x = [1, 2, -1]^T \).

If a precision of 20 digits is used, we get a more accurate solution but it is still not exact. If a precision of 4 digits is used, we get a poor approximation to the exact solution.

\[
\begin{bmatrix}
1 & 0 & 0 & .9824 \\
0 & 1 & 0 & 1.346 \\
0 & 0 & 1 & -1.250
\end{bmatrix}
\]

If a precision of 3 digits is used, we have

\[
\begin{bmatrix}
1 & 0 & -0.73 & 0 \\
0 & 1 & -2.62 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

The coefficient matrix is singular at that precision.

**Norms** (範數)

- The degree of ill-conditioning of a matrix is measured by its *condition number* (條件數).

It is defined in terms of its *norms*, a measure of the magnitude of the matrix.

It is usually use \( \| A \| \) to represent the norm of matrix \( A \). For any definition of norms, it is required

1) \( \| A \| \geq 0 \) and \( \| A \| = 0 \) if and only if \( A = 0 \).

2) \( \| kA \| = k \| A \| \).

3) \( \| A + B \| \leq \| A \| + \| B \| \). (the triangle inequality)

4) \( \| A \cdot B \| \leq \| A \| \cdot \| B \| \) and \( \| A \| = 0 \) if and only if \( A = 0 \).
For the special kind of matrices, called vectors, we can compute the norm of a vector $x = (x_1, x_2, x_3, \ldots, x_n)$ as

a) the Euclidean norm: $\|x\|_2 \equiv \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$;

b) the $p$-norm: $\|x\|_p \equiv (|x_1|^p + |x_2|^p + \ldots + |x_n|^p)^{\frac{1}{p}}$,

for $p = 1 \Rightarrow \|x\|_1 \equiv |x_1| + |x_2| + \ldots + |x_n| = $ sum of magnitude;

for $p = 2 \Rightarrow \|x\|_2 \equiv (x_1^2 + x_2^2 + \ldots + x_n^2)^{\frac{1}{2}} = $ sum of magnitude;

for $p = \infty \Rightarrow \|x\|_{\infty} \equiv \max_{1 \leq i \leq n} (|x_i|) = $ maximum-magnitude norm.

Which is best may depend on the problem itself. In most cases, satisfactory results are obtained with any of these measures.

When we solve a system of equations, we hope that the error of solution is small. Norms can be used to see how great the error is.

Let $\bar{x}$ be an approximation solution to the true solution $x$ for $Ax = b$. And define the residual $\Delta b = b - A\bar{x}$ and the solution error $\Delta x = x - \bar{x}$. Then

$$\Delta b = b - A\bar{x} = Ax - A\bar{x} = A(x - \bar{x}) = A\Delta x.$$ 

Hence,

$$\Delta x = A^{-1}\Delta b.$$ 

Take norms and it is required

$$\|\Delta x\| \leq \|A^{-1}\| \cdot \|\Delta b\|.$$ 

Also for $\Delta b = A\Delta x$, we have

$$\|\Delta b\| \leq \|A\| \cdot \|\Delta x\|.$$ 

The above two equations are combined to get

$$\frac{\|\Delta b\|}{\|A\|} \leq \|\Delta x\| \leq \|A^{-1}\| \cdot \|\Delta b\|. \quad (1)$$

Similarly, for $x = A^{-1}b$, we have

$$\|x\| \leq \|A^{-1}\| \cdot \|b\|;$$

and for $Ax = b$, we have

$$\|b\| \leq \|A\| \cdot \|x\|.$$ 

The above two equations are combined to get

$$\frac{\|b\|}{\|A\|} \leq \|x\| \leq \|A^{-1}\| \cdot \|b\|. \quad (2)$$
From Eqs. (1) and (2), we reach a most important relationship:

$$\frac{1}{\| A^{-1} \|} \cdot \frac{\| \Delta b \|}{\| b \|} \leq \frac{\| \Delta x \|}{\| x \|} \leq \| A \| \cdot \| A^{-1} \| \cdot \frac{\| \Delta b \|}{\| b \|}.$$  

**Condition Numbers**

- The *condition number* $\chi$, defined as $\| A \| \cdot \| A^{-1} \|$ (the product of the norm of $A$ and the norm of its inverse), is the best measure of ill-conditioning. A small number means good-conditioning, a large number means ill-conditioning. So, the previous equation can be written as

$$\frac{1}{\chi} \cdot \frac{\| \Delta b \|}{\| b \|} \leq \frac{\| \Delta x \|}{\| x \|} \leq \chi \cdot \frac{\| \Delta b \|}{\| b \|},$$

where $\| \Delta x \|/\| x \|$ is the relative error and $\| \Delta b \|/\| b \|$ is the relative residual. **When $\chi$ is near unity, the relative residual is a good measure of the relative error.**

We have already seen that an ill-conditioned system is extremely sensitive to small changes in the coefficients. The condition number let us relate the change in the solution vector to such errors in the coefficients of the set of equations $Ax = b$.

Let $\Delta A$ be the errors caused in measuring the coefficients of $A$ and $\bar{x}$ be the solution of the perturbed system. Then the actual set of equations being solved is $(A + \Delta A)\bar{x} = b$. We desire to know how large the solution error $\Delta x = x - \bar{x}$ is.

Using $Ax = b$ and $(A + \Delta A)\bar{x} = b$, we have

$$x = A^{-1}b = A^{-1}[(A + \Delta A)\bar{x}] = A^{-1}[((A + (A + \Delta A) - A)\bar{x}]$$

$$= (A^{-1}A + A^{-1}\Delta A)\bar{x} = (I + A^{-1}\Delta A)\bar{x} = \bar{x} + A^{-1}\Delta A \bar{x}$$

$$\Rightarrow \Delta x = x - \bar{x} = A^{-1}\Delta A \bar{x}.$$  

Taking norms, we get

$$\| \Delta x \| \leq \| A^{-1} \| \cdot \| \Delta A \| \cdot \| \bar{x} \| \leq \| A^{-1} \| \cdot \| A \| \cdot \| \Delta A \| / \| A \|. $$

$$\Rightarrow \frac{\| \Delta x \|}{\| \bar{x} \|} \leq \chi \cdot \frac{\| \Delta A \|}{\| A \|}.$$  

It says that the norm of the solution error relative to the norm of the computed solution can be as large as the relative error in the coefficients of $A$ multiplied by the condition number.
In fact, if the value of $\| \Delta x \| / \| x \|$ is $10^{-p}$, we know that $x$ is probably correct to $p$ digits.

The net effect is that, if the coefficients of $A$ are known to only four-digit precision and the condition number is 1000, the computed solution $\bar{x}$ may have only one digit of accuracy.

**Iterative Improvement**

- It is possible to apply iterative improvement to correct the computed solution $\bar{x}$ so that it more closely agrees with the true solution $x$. Let’s define $\Delta x = x - \bar{x}$ and $\Delta b = b - A\bar{x}$. Then

  \[ A(\Delta x) = A(x - \bar{x}) = Ax - A\bar{x} = b - A\bar{x} = \Delta b. \]

  It means that, if we could solve this equation for $\Delta x$, we could apply this as a correction to $\bar{x}$.

The process of iterative improvement is based on solving $A(\Delta x) = \Delta b$. The computation must be as precise as possible. Unless the system is so ill-conditioned that we cannot get a reasonable approximation to $\Delta x$. We will usually get an improved estimate $x + \Delta x$ for $x$.

**Example 5:** Consider

\[
A = \begin{bmatrix} 4.23 & -1.06 & 2.11 \\ -2.53 & 6.77 & 0.98 \\ 1.85 & -2.11 & -2.32 \end{bmatrix}, \quad b = \begin{bmatrix} 5.28 \\ -5.22 \\ -2.58 \end{bmatrix}
\]

whose true solution is $x = [1, 1, 1]^T$. If inadequate precision is used, we might get an approximate solution as $\bar{x} = [0.991, 0.997, 1.000]^T$. By using double precision, we compute $\Delta b = b - A\bar{x} = [0.0349, -0.00246, 0.0103]^T$. Then we solve $A(\Delta x) = \Delta b$ to get

\[
\Delta x = \begin{bmatrix} 0.00822 \\ 0.00300 \\ -0.00000757 \end{bmatrix}.
\]

Finally, we have

\[
x + \Delta x = \begin{bmatrix} 0.999 \\ 1.000 \\ 1.000 \end{bmatrix}.
\] which is almost exactly the correct solution.
2.5 Iterative Method (疊代法)

• Gaussian elimination and its variants are called direct methods (直接法). An entirely different way to solve systems of equations is through iteration. In this, we start with an initial estimate of the solution vector and proceed to refine this estimate.

There are times when an iterative method is preferred over a direct method, especially when the coefficient matrix is sparse.

Diagonally dominant (對角主導)

• A system is called diagonally dominant if the system of equations can be ordered so that each diagonal entry of the coefficient matrix is larger in magnitude than the sum of the magnitudes of the order coefficients in that row. For such a system, the iteration will converge for any starting values.

Although this may seem like a very restrictive condition, it turns out that there are many applied problems that have this property.

Jacobi Method

• The Jacobi method is also called “the method of simultaneous displacement (同時位移法)”. Consider

\[
\begin{align*}
6x_1 - 2x_2 + x_3 &= 11 \\
x_1 + 2x_2 - 5x_3 &= -1 \\
-2x_1 + 7x_2 + 2x_3 &= 5
\end{align*}
\]

\[
R_2 \leftrightarrow R_3 \quad \Rightarrow \quad \begin{align*}
6x_1 - 2x_2 + x_3 &= 11 \\
-2x_1 + 7x_2 + 2x_3 &= 5, \text{ which is diagonally dominant.} \\
x_1 + 2x_2 - 5x_3 &= -1
\end{align*}
\]

The iterative methods depend on the rearrangement of the equations in this manner:

\[
x_i = \frac{b_i}{a_{ii}} - \sum_{j \neq i}^{n} \frac{a_{ij}}{a_{ii}} x_j, \quad i = 1, 2, \ldots, n.
\]

Each equation is now solved for the variables in succession:
\( x_1 = 1.8333 + 0.3333x_2 - 0.1667x_3, \)
\( x_2 = 0.7143 + 0.2857x_1 - 0.2857x_3, \)
\( x_3 = 0.2000 + 0.2000x_1 + 0.4000x_2. \)

The **Jacobi method** indicates the iterative process by putting superscripts on variables to indicate successive iterates:

\[
\begin{align*}
    x_1^{(k+1)} &= 1.8333 + 0.3333x_2^{(k)} - 0.1667x_3^{(k)}, \\
    x_2^{(k+1)} &= 0.7143 + 0.2857x_1^{(k)} - 0.2857x_3^{(k)}, \\
    x_3^{(k+1)} &= 0.2000 + 0.2000x_1^{(k)} + 0.4000x_2^{(k)},
\end{align*}
\]

where the superscript \((k+1)\) indicates the \((k+1)\)st iterate. Starting with an initial vector \(x^{(0)} = (0, 0, 0)^T\), we get

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.833</td>
<td>2.038</td>
</tr>
<tr>
<td>0</td>
<td>0.714</td>
<td>1.181</td>
</tr>
<tr>
<td>0</td>
<td>0.200</td>
<td>0.852</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.085</td>
<td>1.053</td>
<td>1.080</td>
</tr>
<tr>
<td>2.004</td>
<td>1.001</td>
<td>1.038</td>
</tr>
<tr>
<td>1.994</td>
<td>0.990</td>
<td>1.001</td>
</tr>
<tr>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
</tr>
<tr>
<td>2.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Note that this method is exactly the same as the method of fixed-point iteration for a single equation.

**Gauss-Seidel Iteration**

- If we proceed to improve each \(x\)-value using always the most recent approximations of the other variables. Such a method is called the Gauss-Seidel method. For the previous example, the iterative scheme is

\[
\begin{align*}
    x_1^{(k+1)} &= 1.8333 + 0.3333x_2^{(k)} - 0.1667x_3^{(k)}, \\
    x_2^{(k+1)} &= 0.7143 + 0.2857x_1^{(k+1)} - 0.2857x_3^{(k)}, \\
    x_3^{(k+1)} &= 0.2000 + 0.2000x_1^{(k+1)} + 0.4000x_2^{(k+1)},
\end{align*}
\]

Beginning with \(x^{(0)} = (0, 0, 0)^T\), successive estimates of solution are

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.833</td>
<td>2.069</td>
</tr>
<tr>
<td>0</td>
<td>1.238</td>
<td>1.002</td>
</tr>
<tr>
<td>0</td>
<td>1.062</td>
<td>1.015</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.998</td>
<td>0.995</td>
<td>0.998</td>
</tr>
<tr>
<td>1.999</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>
Note that the rate of convergence for Gauss-Seidel is more rapid than that for Jacobi.

- It should be emphasized that without diagonal dominance, neither Jacobi nor Gauss-Seidel is sure to converge.

In fact, there are examples where Jacobi converges and Gauss-Seidel diverges from the same starting vector.

Given diagonal dominance in the coefficient matrix, the Gauss-Seidel method is often the better choice. However, we may still prefer the Jacobi method if we are running the program on parallel processors because all \( n \) equations can be solved simultaneously at each iteration.

**Accelerating Convergence**

- Convergence in the Gauss-Seidel method can be speeded if we do what is called *over-relaxing*.

The standard relationship for Gauss-Seidel iteration for the set of equations \( Ax = b \), for variable \( x_i \), can be written as

\[
x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j>i+1}^{n} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n,
\]

where the superscript \((k+1)\) indicates the \((k+1)\) iterate. An algebraically equivalent form is

\[
x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j>i}^{n} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n,
\]

because \( x_i^{(k)} \) is both added to and subtracted from the right side. Relaxation can be applied to get

\[
x_i^{(k+1)} = x_i^{(k)} + w \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j>i}^{n} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n.
\]

which is called *over-relaxation* if \( 1 < w < 2 \) and called *under-relaxation* (不足鬆弛法) if \( 0 < w < 1 \).
Table 2.1 shows how the convergence rate is influenced by the value of \( w \) for the system

\[
\begin{bmatrix}
-4 & 1 & 1 & 1 \\
1 & -4 & 1 & 1 \\
1 & 1 & -4 & 1 \\
1 & 1 & 1 & -4
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix}
=
\begin{bmatrix}
1 \\
1 \\
1 \\
1
\end{bmatrix}
\]

by starting with an initial estimate of \( x = [0, 0, 0, 0]^T \). The exact solution is \( x = [-1, -1, -1, -1]^T \).

<table>
<thead>
<tr>
<th>( w, ) the overrelaxation factor</th>
<th>Number of iterations to reach error ( &lt;1 \times 10^{-5} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>24</td>
</tr>
<tr>
<td>1.1</td>
<td>18</td>
</tr>
<tr>
<td>1.2</td>
<td>13</td>
</tr>
<tr>
<td>1.3</td>
<td>11</td>
</tr>
<tr>
<td>1.4</td>
<td>14</td>
</tr>
<tr>
<td>1.5</td>
<td>18</td>
</tr>
<tr>
<td>1.6</td>
<td>24</td>
</tr>
<tr>
<td>1.7</td>
<td>35</td>
</tr>
<tr>
<td>1.8</td>
<td>55</td>
</tr>
<tr>
<td>1.9</td>
<td>100+</td>
</tr>
</tbody>
</table>

**Homework #4**

Starting with the initial vector \( x = [0, 0, 0]^T \) to solve the system of equations

\[
\begin{align*}
4.63x_1 - 1.21x_2 + 3.22x_3 &= 2.22, \\
-3.07x_1 + 5.48x_2 + 2.11x_3 &= -3.17, \\
1.26x_1 + 3.11x_2 + 4.57x_3 &= 5.11.
\end{align*}
\]

a) Use the Jacobi method.
b) Use the Gauss-Seidel method.
c) Show the influences of the over-relaxation factor \( w \) that speed the solutions?
3 Interpolation and Curve Fitting (內插與曲線近似)

• It once was the case that students found values for sines, logarithms, and other non-algebraic functions from tables rather than getting the values using a computer or calculator.

• There are four reasons why we devote such a lengthy chapter:
  1) Interpolation methods are the basis for many other procedures that you will study in this course, such as numerical integration and differentiation;
  2) they are behind the ways that we use to solve ordinary and partial-differential equations;
  3) they demonstrate important theory about polynomials and the accuracy of numerical methods; and
  4) they are one of the more important ways that curves are drawn on your computer screen.

• In this chapter you will learn how scattered data can be interpolated to estimate values at uniformly positioned grid points.

3.1 Interpolating Polynomials

• The reasons for using polynomials are
  1) they are “nice” (smooth and continuously differentiable) functions;
  2) their evaluation requires only those arithmetic operations that computers can do.

Fitting a Polynomial to Data

• Suppose that we have the following data pairs:

<table>
<thead>
<tr>
<th>x</th>
<th>f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>22.0</td>
</tr>
<tr>
<td>2.7</td>
<td>17.8</td>
</tr>
<tr>
<td>1.0</td>
<td>14.2</td>
</tr>
<tr>
<td>4.8</td>
<td>38.3</td>
</tr>
</tbody>
</table>

They can be used to determine the four unknown coefficients of the cubic polynomial
\[ f(x) = ax^3 + bx^2 + cx + d. \] That is, solve
\[
\begin{align*}
3.2^3 + b(3.2)^2 + c(3.2) + d &= 22.0, \\
2.7^3 + b(2.7)^2 + c(2.7) + d &= 17.8, \\
1.0^3 + b(1.0)^2 + c(1.0) + d &= 14.2, \\
4.8^3 + b(4.8)^2 + c(4.8) + d &= 38.3,
\end{align*}
\]
to get \( a = -0.5275 \), \( b = 6.4952 \), \( c = -16.1177 \), and \( d = 24.3499 \). At \( x = 3.0 \), the estimated value is \( 20.212 \).

The technique is awkward (難用的):

1) especially if we want a new polynomial that is also made to fit at a new point,
2) or if we want to see what difference it would make to use a quadratic instead of a cubic.
3) Furthermore, it leads to an ill-conditioned system of equations. For this example, the condition number is about 2700.

### Lagrangian Polynomials

- The Lagrangian polynomial is perhaps the simplest way to exhibit the existence of a polynomial for interpolation with unevenly spaced data.

As an example, suppose we have a set of four-point data as:

\[
\begin{align*}
x_0 & \quad f_0 \\
x_1 & \quad f_1 \\
x_2 & \quad f_2 \\
x_3 & \quad f_3
\end{align*}
\]

The Lagrangian form for this is
\[
P_3(x) = \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)} f_0 + \frac{(x - x_0)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)} f_1
\]
\[
+ \frac{(x - x_0)(x - x_1)(x - x_3)}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_3)} f_2 + \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)} f_3. \tag{3.1}
\]

This equation is made up of four terms, each of which is a cubic in \( x \). It will have \((n+1)\) terms when the degree is \( n \). The arithmetic in this method is tedious (冗長的).
Example 3.1 For the previous case,

\[
P_3(3.0) = \frac{(3.0 - 2.7)(3.0 - 1.0)(3.0 - 4.8)}{(3.2 - 2.7)(3.2 - 1.0)(3.2 - 4.8)} \quad (22.0)
\]

\[
+ \frac{(3.0 - 3.2)(3.0 - 1.0)(3.0 - 4.8)}{(2.7 - 3.2)(2.7 - 1.0)(2.7 - 4.8)} \quad (17.8)
\]

\[
+ \frac{(3.0 - 3.2)(3.0 - 2.7)(3.0 - 4.8)}{(1.0 - 3.2)(1.0 - 2.7)(1.0 - 4.8)} \quad (14.2)
\]

\[
+ \frac{(3.0 - 3.2)(3.0 - 2.7)(3.0 - 1.0)}{(4.8 - 3.2)(4.8 - 2.7)(4.8 - 1.0)} \quad (38.3).
\]

Carrying out the arithmetic, \( P_3(3.0) = 20.21 \). We get the same result as before. Note that every \( n \)th-degree polynomial that passes through the same \((n+1)\) points is identical.

- It is most important that you **never fit a polynomial of a degree higher than 4 or 5 to a set of points**. If the degree is too low, the interpolating polynomial does not give good estimates of \( f(x) \); if the degree is too high, undesirable oscillations in polynomial values can occur. If you want to fit a set of more than six points, be sure to **break up the set into subsets**.

Figure 3.18 illustrates why this is so necessary. A still better way to fit a large number of data points is to use **spline curves** as described in Section 3.3.
3.2 Divided Differences

• There are three disadvantages to use the Lagrangian polynomial for interpolation:
  1) it involves more arithmetic operations than does the divided-difference method;
  2) if we desire to add or subtract a point from the set used to construct the polynomial, we essentially have to start over in the computation;
  3) if we desire to interpolate at a new \( x \)-value, we have to repeat all of the computation.

The divided-difference method avoids all of these disadvantages.

Divided-Difference Tables

• Assume that the function, \( f(x) \), is known at several values for \( x \):

\[
\begin{array}{c|c}
 x_0 & f_0 \\
 x_1 & f_1 \\
 x_2 & f_2 \\
 x_3 & f_3 \\
\end{array}
\]

It is not required that the \( x \)’s are evenly spaced or arranged in any particular order.

Let the \( n \)-degree polynomial

\[
P_n(x) = a_0 + (x - x_0)a_1 + (x - x_0)(x - x_1)a_2 + \cdots + (x - x_0)(x - x_1) \cdots (x - x_{n-1})a_n.
\]

We will show that the \( a \)’s are readily determined by using what are called the divided differences of the tabulated values.

Define the first-order divided difference between \( x_0 \) and \( x_1 \) as

\[
f[x_0, x_1] = \frac{f_1 - f_0}{x_1 - x_0}.
\]

Note that \( f[x_0, x_1] = f[x_1, x_0] \). Second- and higher-order divided differences are defined in terms of lower-order differences. For example,

\[
f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}
\]

and

\[
f[x_0, x_1, \ldots, x_n] = \frac{f[x_1, x_2, \ldots, x_n] - f[x_0, x_1, \ldots, x_{n-1}]}{x_n - x_0}.
\]

The concept is even extended to a zero-order difference:

\[
f[x_i] = f_i.
\]

A divided-difference table is shown in Table 3.1. Table 3.2 shows the results applied for the previous example.
We are now ready to establish the values of \( a \)'s by these divided differences.

At \( x = x_0 \), \( P_n(x_0) = f_0 = a_0 \).

At \( x = x_1 \), \( P_n(x_1) = f_1 = a_0 + (x_1 - x_0)a_1 \).

At \( x = x_2 \), \( P_n(x_2) = f_2 = a_0 + (x_2 - x_0)a_1 + (x_2 - x_0)(x_2 - x_1)a_2 \).

\[ \vdots \]

At \( x = x_n \), \( P_n(x_n) = f_n = a_0 + (x_2 - x_0)a_1 + (x_2 - x_0)(x_2 - x_1)a_2 + \cdots + (x_n - x_0)(x_n - x_1) \cdots (x_n - x_{n-1})a_n \).

\[ \Rightarrow \]

\[ a_0 = f_0, \]

\[ a_1 = \frac{f_1 - f_0}{x_1 - x_0} = f[x_0, x_1], \]

\[ a_2 = \frac{(f_2 - f_1)(x_2 - x_1) - (f_1 - f_0)(x_2 - x_0)}{x_2 - x_0} = \frac{f[x_0, x_1, x_2]}{x_2 - x_0} = f[x_0, x_1, x_2], \]

\[ \vdots \]

\[ a_n = f[x_0, x_1, \ldots, x_n]. \]

\[ \Rightarrow \]

\[ P_n(x) = f[x_0] + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + \cdots + (x - x_0)(x - x_1) \cdots (x - x_{n-1})f[x_0, x_1, \ldots, x_n] + \cdots \]
Example 3.3 Write the interpolating polynomial of degree-3 that fits the data of Table 3.2 at all points from $x_0 = 3.2$ to $x_3 = 4.8$.

Table 3.2

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$f_i$</th>
<th>$f[x_0, x_1]$</th>
<th>$f[x_0, x_1, x_2]$</th>
<th>$f[x_0, x_1, x_2, x_3]$</th>
<th>$f[x_0, x_1, x_2, x_3, x_4]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>22.0</td>
<td>8.400</td>
<td>2.856</td>
<td>-0.528</td>
<td>0.256</td>
</tr>
<tr>
<td>2.7</td>
<td>17.8</td>
<td>2.118</td>
<td>2.012</td>
<td>0.0865</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>14.2</td>
<td>6.342</td>
<td>2.263</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.8</td>
<td>38.3</td>
<td>16.750</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td>51.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$P_3(x) = 22.0 + 8.400(x - 3.2) + 2.856(x - 3.2)(x - 2.7) - 0.528(x - 3.2)(x - 2.7)(x - 1.0)$.

What is the fourth-degree polynomial that fits at all five points? We only have to add one more term to $P_3(x)$.

$P_4(x) = P_3(x) + 0.256(x - 3.2)(x - 2.7)(x - 1.0)(x - 4.8)$

When this method is used for interpolation, we observe that nested multiplication can be used to cut down on the number of arithmetic operations, for example:

$P_3(x) = 22.0 + (x - 3.2)[8.400 + (x - 2.7)[2.856 - 0.528(x - 1.0)]]$.

**Divided-Differences for a Polynomial**

- Assume that the function, $f(x)$, is known at several values for $x$:

  It is of interest to look at the divided differences for $f(x) = P_n(x)$. Suppose that $f(x)$ is the cubic

  $f(x) = 2x^3 - x^2 + x - 1$.

  Here is its divided-difference table:

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$f[x_i]$</th>
<th>$f[x_1, x_2]$</th>
<th>$f[x_1, x_2, x_3]$</th>
<th>$f[x_1, x_2, x_3, x_4]$</th>
<th>$f[x_1, x_2, x_3, x_4, x_5]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.30</td>
<td>-0.7360</td>
<td>2.4800</td>
<td>3.0000</td>
<td>2.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>1.00</td>
<td>1.0000</td>
<td>3.6800</td>
<td>3.6000</td>
<td>2.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.70</td>
<td>-0.1040</td>
<td>2.2400</td>
<td>5.4000</td>
<td>2.0000</td>
<td></td>
</tr>
<tr>
<td>0.60</td>
<td>-0.3280</td>
<td>8.7200</td>
<td>8.2000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.90</td>
<td>11.0080</td>
<td>21.0200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.10</td>
<td>15.2120</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Observe that the third divided differences are all the same and it follows that all higher divided differences will be zero. This indicates that the function behaves nearly like a polynomial of that degree.

- It is important to recognize that *every polynomial of degree $n$ that has the same value at*
\( n+1 \) distinct points is exactly the same. The interpolating polynomials obtained by the Lagrangian method and through divided differences look different but they are really identical.

**Error of Interpolation from Differences**

- Use of a higher-degree polynomial for interpolation is usually *not preferred*.

Example 3.4 Find the error of the interpolates for \( f(1.75) \) using polynomials of degree-1, -2, and -3 for \( f(x) = x^2 e^{-x/2} \).

<table>
<thead>
<tr>
<th>( x_i )</th>
<th>( f[x_i] )</th>
<th>( f[x_{i-1}, x_{i+1}] )</th>
<th>( f[x_i, \ldots, x_{i+2}] )</th>
<th>( f[x_i, \ldots, x_{i+3}] )</th>
<th>( f[x_i, \ldots, x_{i+4}] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.10</td>
<td>0.6981</td>
<td>0.8593</td>
<td>-0.1755</td>
<td>0.0032</td>
<td>0.0027</td>
</tr>
<tr>
<td>2.00</td>
<td>1.4715</td>
<td>0.4381</td>
<td>-0.1631</td>
<td>0.0191</td>
<td></td>
</tr>
<tr>
<td>3.50</td>
<td>2.1287</td>
<td>-0.0511</td>
<td>-0.0657</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.00</td>
<td>2.0521</td>
<td>-0.2877</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.10</td>
<td>1.4480</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3 Errors of interpolation for \( f(1.75) \)

<table>
<thead>
<tr>
<th>Degree</th>
<th>Interpolated value</th>
<th>Actual error</th>
<th>( f^{(n+1)} ) maximum</th>
<th>( f^{(n+1)} ) minimum</th>
<th>Upper bound</th>
<th>Lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.25668</td>
<td>0.01996</td>
<td>-0.3679</td>
<td>0.0594</td>
<td>0.0299</td>
<td>-0.00483</td>
</tr>
<tr>
<td>2</td>
<td>1.28520</td>
<td>-0.00856</td>
<td>-0.8661</td>
<td>0.1249</td>
<td>0.0059</td>
<td>-0.0408</td>
</tr>
<tr>
<td>3</td>
<td>1.28611</td>
<td>-0.00947</td>
<td>1.1398</td>
<td>-0.0359</td>
<td>0.0014</td>
<td>-0.0439</td>
</tr>
</tbody>
</table>

In this case, observe that the use of a cubic polynomial does not improve the accuracy.

**Evenly Spaced Data** (等間距資料)

- If the \( x \)-values are evenly spaced (等間距), getting an interpolating polynomial is considerably simplified.

Instead of using divided differences (i.e. \( f[x_0, x_i] = \frac{f_i - f_0}{x_i - x_0} \)), “ordinary differences” (i.e. \( \Delta f_0 = f_i - f_0 \)) are used. Suppose that a table have entries indexed from 0 to \( N \).

First-order differences:

\[ \Delta f_i = f_{i+1} - f_i, \quad i = 0, 1, 2, \ldots, N-1. \]

Second-order differences:

\[ \Delta^2 f_i = \Delta(\Delta f_i) = \Delta(f_{i+1} - f_i) = \Delta f_{i+1} - \Delta f_i = f_{i+2} - 2f_{i+1} + f_i, \quad i = 0, 1, 2, \ldots, N-2. \]

...
Nth-order differences:
\[ \Delta^n f_i = f_{i+n} - nf_{i+n-1} + \frac{n(n-1)}{2!} f_{i+n-2} - \cdots \pm f_i, \quad i = 0, 1, 2, \ldots, N-n. \]

An interpolating polynomial of degree \( n \) can be written as
\[ P_n(x_s) = f_0 + s\Delta f_0 + \frac{s(s-1)}{2!} \Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!} \Delta^3 f_0 + \cdots + \frac{s(s-1)(s-2)(s-3)\ldots(s-n+1)}{n!} \Delta^n f_0 \]
where \( s = (x - x_0)/h \), with \( h = \Delta x \) the uniform spacing in \( x \)-values. The interpolating polynomial is called the **Newton-Gregory forward polynomial**.

Example: Given the table of \( x \), \( f(x) \) values, and the columns of differences, find \( f(0.73) \) from a cubic interpolating polynomial.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( f(x) )</th>
<th>( \Delta f )</th>
<th>( \Delta^2 f )</th>
<th>( \Delta^3 f )</th>
<th>( \Delta^4 f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.000</td>
<td>0.203</td>
<td>0.017</td>
<td>0.024</td>
<td>0.020</td>
</tr>
<tr>
<td>0.2</td>
<td>0.203</td>
<td>0.220</td>
<td>0.041</td>
<td>0.044</td>
<td>0.052</td>
</tr>
<tr>
<td>0.4</td>
<td>0.423</td>
<td>0.261</td>
<td>0.085</td>
<td>0.096</td>
<td>0.211</td>
</tr>
<tr>
<td>0.6</td>
<td>0.684</td>
<td>0.346</td>
<td>0.181</td>
<td>0.307</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>1.030</td>
<td>0.527</td>
<td>0.488</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>1.557</td>
<td>1.015</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>2.572</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In order to center the \( x \)-value around \( x = 0.73 \), we must use the four entries beginning with \( x = 0.4 \). Inserting the proper values into the expression for the Newton-Gregory polynomial, we get
\[ P_3(0.73) = 0.423 + (1.65)(0.261) + \frac{(1.65)(0.65)}{2!} (0.085) + \frac{(1.65)(0.65)(-0.35)}{3!} (0.096) \]
\[ = 0.893 \]
The function is actually for \( f(x) = \tan(x) \), so we know that the true value of \( f(0.73) \) is 0.895. The error is 0.002.

One nice feature of a table of ordinary differences is that an error in an entry for \( f(x) \) can be readily detected. Such an error causes a disruption to the regular progression of values in the columns of differences. (使用一般差分表的一個好處，是 \( f(x) \) 值的輸入錯誤可以被偵測知道。因為錯誤的 \( f(x) \) 值會破壞差分表各欄位中數值變化的規則性。)
3.3 Spline Curves (Spline 曲線)

• There are times when fitting an interpolating polynomial to data points is very difficult. Here is an example where we try to fit to data pairs from a known function \( f(x) = \cos^{10}(x) \) on the interval [-2, 2].

![Figure 3.1](image)

None of the polynomials is a good presentation of the function. In particular, observe how the 8th-degree polynomial deviates widely near \( |x| = 2 \). Polynomials of degree higher than 8 will exhibit even greater deviations because the polynomials must have many zeros, while we try to match \( f(x) \) where it is flat.

![Figure 3.2](image)

We might think that a solution to the problem would be to break up the interval [-2, 2] into subintervals and fit separate polynomials to the function in these smaller intervals. Figure 3.2 shows a much better fit if we use a quadratic between \( x = -0.65 \) and \( x = 0.65 \), with
$P(x) = 0$ outside that interval. That is better, but there are discontinuities in the slope where the separate polynomials join.

An answer to the dilemma is to use **spline curves**. It borrows from the idea of a device used in drafting. A draftsman fits curves such as our example by bending a flexible rod to conform to the curve. The points at which the splines join are called **knots**.

Spline curves may be of varying degrees. If the polynomials are all of degree-1, we have a **linear spline**. In this case, the slopes at the knots are discontinuous. Splines of degree higher than 1 do not have this problem. **Most often cubic splines are used.**

### The Equation for a Cubic Spline

- According to the laws of beam flexure, both the slope and curvature are everywhere continuous. It requires that the mathematical spline curve should be of at least degree-3.

Write the equation for a cubic polynomial, $g_i(x)$, in the $i$th interval between points $(x_i, y_i)$ and $(x_{i+1}, y_{i+1})$ as

$$g_i(x) = a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + d_i.$$  \hspace{1cm} (3.9)

And the first- and second –order derivatives of $g_i(x)$ are

$$g_i'(x) = 3a_i(x-x_i)^2 + 2b_i(x-x_i) + c_i.$$  \hspace{1cm}

$$g_i''(x) = 6a_i(x-x_i) + 2b_i.$$  

If there are $n+1$ points, the number of intervals and the number of $g_i(x)$’s are $n$. Thus, there are $4n$ unknowns, which are the \{ $a_i$, $b_i$, $c_i$, $d_i$ \} for \( i = 0, 1, 2, \ldots, n-1 \).

To solve the $4n$ unknowns, $4n$ conditions/equations/constraints are required:

$$g_i(x_i) = y_i, \hspace{1cm} i = 0, 1, 2, \ldots, n-1 \hspace{1cm} \Rightarrow \hspace{1cm} n \hspace{0.5mm} \text{constraints.} \hspace{1cm} (3.10a)$$  

$$g_i(x_{i+1}) = y_{i+1}, \hspace{1cm} i = 0, 1, 2, \ldots, n-1 \hspace{1cm} \Rightarrow \hspace{1cm} n \hspace{0.5mm} \text{equations.} \hspace{1cm} (3.10b)$$  

$$g_i'(x_{i+1}) = g_i'(x_{i+1}), \hspace{1cm} i = 0, 1, 2, \ldots, n-2 \hspace{1cm} \Rightarrow \hspace{1cm} n-1 \hspace{0.5mm} \text{equations.} \hspace{1cm} (3.10c)$$  

$$g_i''(x_{i+1}) = g_i''(x_{i+1}), \hspace{1cm} i = 0, 1, 2, \ldots, n-2 \hspace{1cm} \Rightarrow \hspace{1cm} n-1 \hspace{0.5mm} \text{equations.} \hspace{1cm} (3.10d)$$

The development is simplified if we write the equations in terms of the second derivatives,
that is, \( S_0, S_1, S_2, \ldots, S_n \).

Let \( S_i = g''(x_i) \) for \( i = 0, 1, 2, \ldots, n \), and let \( h_j = x_{i+1} - x_i \) for \( i = 0, 1, 2, \ldots, n-1 \).

1) Eq.(3.10a) \Rightarrow d_i = y_i

2) \( S_i = g''(x_i) = 6a_i(x_i - x_i) + 2b_i = 2b_i \Rightarrow b_i = \frac{1}{2} S_i \)

3) \( S_{i+1} = g''(x_{i+1}) = 6a_i(x_{i+1} - x_i) + 2b_i \Rightarrow a_i = \frac{1}{6h_i}(S_{i+1} - S_i) \)

Substitute the expressions for \( a_i, b_i, d_i \) into equation (3.9) to get

\[
y_{i+1} = g_i(x_{i+1}) = \frac{(S_{i+1} - S_i)}{6h_i}(x_{i+1} - x_i)^3 + \frac{S_i}{2}(x_{i+1} - x_i)^2 + c_i(x_{i+1} - x_i) + y_i.
\]

\[
y_{i+1} = \frac{S_i}{h_i} h_i^3 + \frac{S_i}{2} h_i^2 + c_i h_i + y_i.
\]

\[
\Rightarrow c_i = \frac{y_{i+1} - y_i}{h_i} - \frac{(S_{i+1} + S_i)}{6h_i} h_i
\]

Rewrite Eq.(3.10c) as \( g_{i-1}'(x_i) = g_i'(x_i) \) for \( i = 1, 2, \ldots, n-1 \)

\[
3a_{i-1}(x_i - x_{i-1})^2 + 2b_{i-1}(x_i - x_{i-1}) + c_{i-1} = 3a_i(x_i - x_i)^2 + 2b_i(x_i - x_i) + c_i.
\]

\[
\Rightarrow 3a_{i-1} + 2b_{i-1} + \frac{h_i}{h_{i-1}} + \frac{h_i}{h_{i-1}} - \frac{(S_{i+1} + S_i)}{6} h_i = \frac{h_i}{h_{i-1}} - \frac{(S_{i+1} + S_i)}{6} h_i.
\]

Simplifying this equation, we get

\[
h_{i-1}S_{i-1} + (2h_{i-1} + 2h_i)S_i + h_iS_{i+1} = 6(\frac{h_i - h_{i-1}}{h_i} - \frac{h_{i-1} - h_i}{h_{i-1}}).
\]

which can be written in matrix form:

\[
\begin{bmatrix}
h_0 & 2(h_0 + h_1) & h_1 \\
h_1 & 2(h_1 + h_2) & h_2 \\
h_2 & 2(h_2 + h_3) & h_3 \\
\vdots & & \vdots \\
h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\
h_{n-1} & 2h_{n-1} & \end{bmatrix}
\begin{bmatrix}
h_0 \\
h_1 \\
h_2 \\
\vdots \\
h_{n-2} \\
h_{n-1}
\end{bmatrix}
= \begin{bmatrix}
S_0 \\
S_1 \\
S_2 \\
\vdots \\
S_{n-1} \\
S_n
\end{bmatrix}
= \begin{bmatrix}
f[x_0, x_1] - f[x_0, x_1] \\
f[x_1, x_2] - f[x_1, x_2] \\
f[x_2, x_3] - f[x_2, x_3] \\
\vdots \\
f[x_{n-2}, x_{n-1}] - f[x_{n-2}, x_{n-1}]
\end{bmatrix}
\]

There are overall \( n-1 \) equations for \( n+1 \) unknowns \( (S_0, S_1, S_2, \ldots, S_n) \). We need two additional equations involving \( S_0 \) and \( S_n \) by specifying conditions pertaining to the end intervals of the whole curve. The following four alternative choices are often used:

1) Take \( S_0 = S_n = 0 \). This condition, called a nature spline, matches precisely to the drafting device.

2) Take \( f'(x_0) = A \) and \( f'(x_n) = B \) to fix the slopes at each end.

At left end: \( \frac{f(x_0) - A}{h_0/2} = \frac{1}{2}(2S_0 + S_1) \Rightarrow 2h_0S_0 + h_0S_1 = 6(f[x_0, x_1] - A) \)

At right end: \( \frac{B - f(x_{n-1}, x_n)}{h_{n-1}/2} = \frac{1}{2}(S_{n-1} + 2S_n) \Rightarrow h_{n-1}S_{n-1} + 2h_{n-1}S_n = 6(B - f[x_{n-1}, x_n]) \)
3) Take \( S_0 = S_1 \) and \( S_n = S_{n-1} \).

4) Take \( S_0 \) as a linear extrapolation from \( S_1 \) and \( S_2 \), and \( S_n \) as a linear extrapolation from \( S_{n-1} \) and \( S_{n-2} \).

For each end condition, the coefficient matrices become

**Condition 1:** \( S_0 = S_n = 0 \)

\[
\begin{bmatrix}
2(h_0 + h_1) & h_1 & & & & & \\
 h_1 & 2(h_0 + h_2) & h_2 & & & & \\
 & h_2 & 2(h_0 + h_3) & h_3 & & & \\
 & & & \ddots & & & \\
 & & & & h_{n-2} & 2(h_0 + h_{n-1}) & h_{n-1}
\end{bmatrix}
\begin{bmatrix}
S_1 \\
S_2 \\
\vdots \\
S_{n-1}
\end{bmatrix}
= 6
\begin{bmatrix}
f[x_1, x_2] - f[x_0, x_1] \\
f[x_2, x_3] - f[x_1, x_2] \\
\vdots \\
f[x_{n-1}, x_n] - f[x_{n-2}, x_{n-1}]
\end{bmatrix}

\Rightarrow n-1 \text{ equations to solve for } n-1 \text{ unknowns}

**Condition 2:** \( 2h_0 S_0 + h_0 S_1 = 6(f[x_0, x_1] - A) \) and \( h_{n-1} S_{n-1} + 2h_{n-2} S_n = 6(B - f[x_{n-1}, x_n]) \)

\[
\begin{bmatrix}
2h_0 & h_0 \\
 h_0 & 2(h_0 + h_1) & h_1 & & & & \\
 & h_1 & 2(h_0 + h_2) & h_2 & & & \\
 & & h_2 & 2(h_0 + h_3) & h_3 & & \\
 & & & \ddots & & & \\
 & & & & h_{n-2} & 2(h_0 + h_{n-1}) & h_{n-1}
\end{bmatrix}
\begin{bmatrix}
S_0 \\
S_1 \\
\vdots \\
S_{n-1}
\end{bmatrix}
= 6
\begin{bmatrix}
f[x_0, x_1] - A \\
f[x_1, x_2] - f[x_0, x_1] \\
f[x_2, x_3] - f[x_1, x_2] \\
\vdots \\
f[x_{n-1}, x_n] - f[x_{n-2}, x_{n-1}]
\end{bmatrix}

\Rightarrow n+1 \text{ equations to solve for } n+1 \text{ unknowns}

**Condition 3:** \( S_0 = S_1 \) and \( S_n = S_{n-1} \)

\[
\begin{bmatrix}
(3h_0 + 2h_1) & h_1 & & & & & \\
 h_1 & 2(h_0 + h_2) & h_2 & & & & \\
 & h_2 & 2(h_0 + h_3) & h_3 & & & \\
 & & \ddots & & & & \\
 & & & h_{n-2} & 2(h_0 + h_{n-1}) & h_{n-1}
\end{bmatrix}
\begin{bmatrix}
S_1 \\
S_2 \\
\vdots \\
S_{n-1}
\end{bmatrix}
= 6
\begin{bmatrix}
f[x_1, x_2] - f[x_0, x_1] \\
f[x_2, x_3] - f[x_1, x_2] \\
\vdots \\
f[x_{n-1}, x_n] - f[x_{n-2}, x_{n-1}]
\end{bmatrix}

\Rightarrow n-1 \text{ equations to solve for } n-1 \text{ unknowns}

**Condition 4:** \( S_0 \) and \( S_n \) are linear extrapolations:

\[
\begin{bmatrix}
(3h_0 + 2h_1) & h_1 & & & & & \\
 h_1 & 2(h_0 + h_2) & h_2 & & & & \\
 & h_2 & 2(h_0 + h_3) & h_3 & & & \\
 & & \ddots & & & & \\
 & & & h_{n-2} & 2(h_0 + h_{n-1}) & h_{n-1}
\end{bmatrix}
\begin{bmatrix}
S_1 \\
S_2 \\
\vdots \\
S_{n-1}
\end{bmatrix}
= 6
\begin{bmatrix}
f[x_1, x_2] - f[x_0, x_1] \\
f[x_2, x_3] - f[x_1, x_2] \\
\vdots \\
f[x_{n-1}, x_n] - f[x_{n-2}, x_{n-1}]
\end{bmatrix}

\Rightarrow n-1 \text{ equations to solve for } n-1 \text{ unknowns}
If the data are evenly spaced, the matrices reduce to a simple form.

After the $S_i$ values are obtained, we get the coefficients $a_i$, $b_i$, $c_i$ and $d_i$ for the cubic splines in each interval:

\[
a_i = \frac{1}{6h} (S_{i+1} - S_i),
\]
\[
b_i = \frac{1}{2} S_i,
\]
\[
c_i = \frac{x_{i+1} - x_i}{h} - \frac{(S_{i+1} + 2S_i)}{6} h, 
\]
\[
d_i = y_i.
\]

**EXAMPLE 3.5**

Fit the data of Table 3.6 with a natural cubic spline curve, and evaluate the spline values $g(0.66)$ and $g(1.75)$. [The true relation is $f(x) = 2e^x - x^2$.] We see that $h_0 = 1.0$, $h_1 = 0.5$, and $h_2 = 0.75$. The divided differences that we can use to get the right-hand sides of our equations are $f[0, 1] = 2.4366$, $f[1, 1.5] = 4.5536$, and $f[1.5, 2.25] = 9.5995$.

**Table 3.6**

<table>
<thead>
<tr>
<th>$x$</th>
<th>$f(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>2.0000</td>
</tr>
<tr>
<td>1.0</td>
<td>4.4366</td>
</tr>
<tr>
<td>1.5</td>
<td>6.7134</td>
</tr>
<tr>
<td>2.25</td>
<td>13.9130</td>
</tr>
</tbody>
</table>

![Figure 3.3](image)

For a natural cubic spline, we use end condition I and solve

\[
\begin{bmatrix} 3.0 & 0.5 \\ 0.5 & 2.5 \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} = \begin{bmatrix} 12.7020 \\ 30.2754 \end{bmatrix},
\]

giving $S_1 = 2.2920$ and $S_2 = 11.6518$. ($S_0 = S_3 = 0$, of course.) Using these $S$'s, we compute the coefficients of the individual cubic splines to arrive at

<table>
<thead>
<tr>
<th>$i$</th>
<th>Interval</th>
<th>$g_i(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0.0, 1.0]</td>
<td>$0.3820(x - 0)^3 + 0(x - 0)^2 + 2.0546(x - 0) + 2.0000$</td>
</tr>
<tr>
<td>1</td>
<td>[1.0, 1.5]</td>
<td>$3.1199(x - 1)^3 + 1.146(x - 1)^2 + 3.2005(x - 1) + 4.4366$</td>
</tr>
<tr>
<td>2</td>
<td>[1.5, 2.25]</td>
<td>$-2.5893(x - 1.5)^3 + 5.8259(x - 1.5)^2 + 6.6866(x - 1.5) + 6.7134$</td>
</tr>
</tbody>
</table>

Figure 3.3 shows the cubic spline curve. (You should verify that these equations satisfy all the conditions that were given for cubic spline curves.)

We use $g_0$ to find $g(0.66)$: It is 3.4659. (True = 3.4340)

We use $g_2$ to find $g(1.75)$: It is 8.7087. (True = 8.4467)
Some observations on this example: (a) We were given four points that define three intervals, (b) on each of the three intervals a $g(x)$ is defined, and (c) because each $g$ has four coefficients, we must evaluate 12 unknown coefficients. However, by introducing the $S$'s, we only had to solve two equations!

**EXAMPLE 3.7**

Fit cubic splines to $f(x) = \cos^{10}(x)$ with knots at $-2, -1, -0.5, 0, 0.5, 1,$ and 2. Figure 3.5 shows the points superimposed on the spline function, and Table 3.9 compares the values from the splines with the true values for the function at several points. The agreement is excellent. (The figure and table are on p. 179.)

![Figure 3.5](image)

**Table 3.9** A cubic spline fitted to the function $f(x) = \cos^{10}(x)$, end condition 1

<table>
<thead>
<tr>
<th>$x$-value</th>
<th>Spline value</th>
<th>$f(x)$</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2.00</td>
<td>0.0002</td>
<td>0.0002</td>
<td>0.0000</td>
</tr>
<tr>
<td>-1.75</td>
<td>-0.0046</td>
<td>0.0000</td>
<td>0.0046</td>
</tr>
<tr>
<td>-1.50</td>
<td>-0.0073</td>
<td>0.0000</td>
<td>0.0073</td>
</tr>
<tr>
<td>-1.25</td>
<td>-0.0058</td>
<td>0.0000</td>
<td>0.0058</td>
</tr>
<tr>
<td>-1.00</td>
<td>0.0021</td>
<td>0.0021</td>
<td>-0.0000</td>
</tr>
<tr>
<td>-0.75</td>
<td>0.0467</td>
<td>0.0440</td>
<td>-0.0027</td>
</tr>
<tr>
<td>-0.50</td>
<td>0.2709</td>
<td>0.2709</td>
<td>-0.0000</td>
</tr>
<tr>
<td>-0.25</td>
<td>0.7283</td>
<td>0.7292</td>
<td>0.0009</td>
</tr>
<tr>
<td>0.00</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.25</td>
<td>0.7283</td>
<td>0.7292</td>
<td>0.0009</td>
</tr>
<tr>
<td>0.50</td>
<td>0.2709</td>
<td>0.2709</td>
<td>-0.0000</td>
</tr>
<tr>
<td>0.75</td>
<td>0.0467</td>
<td>0.0440</td>
<td>-0.0027</td>
</tr>
<tr>
<td>1.00</td>
<td>0.0021</td>
<td>0.0021</td>
<td>-0.0000</td>
</tr>
<tr>
<td>1.25</td>
<td>-0.0058</td>
<td>0.0000</td>
<td>0.0058</td>
</tr>
<tr>
<td>1.50</td>
<td>-0.0073</td>
<td>0.0000</td>
<td>0.0073</td>
</tr>
<tr>
<td>1.75</td>
<td>-0.0046</td>
<td>0.0000</td>
<td>0.0046</td>
</tr>
<tr>
<td>2.00</td>
<td>0.0002</td>
<td>0.0002</td>
<td>-0.0000</td>
</tr>
</tbody>
</table>

**Homework #5**

A fictitious chemical experiment produces seven data points:

<table>
<thead>
<tr>
<th>$t$</th>
<th>-1.0</th>
<th>-0.96</th>
<th>-0.86</th>
<th>-0.79</th>
<th>0.22</th>
<th>0.50</th>
<th>0.93</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>-1.00</td>
<td>-0.151</td>
<td>0.894</td>
<td>0.986</td>
<td>0.895</td>
<td>0.500</td>
<td>-0.306</td>
</tr>
</tbody>
</table>

d) Plot the points and interpolate a smooth curve by intuition.

e) Plot the unique sixth-degree polynomial that interpolates these points.

f) Use a spline program to evaluate enough points to plot this curve.

Compare your results with the graph in Figure 3.18.
3.4 Bezier Curves and B-Spline Curves (Bezier 曲線與 B-spline 曲線)

- Bezier curves and B-spline are widely used in computer graphics and computer-aided design. B-splines are often used to numerically integrate and differentiate functions that are defined only through a set of data points.

These two types of curves are not really interpolating splines, because the curves do not normally pass through all of the points.

These two new spline curves have a nice geometric property, say a “local” effect. It means that when changing one of the points we change only one portion of the curve. For the cubic spline curve of the previous section, changing just one point has a “global” effect in that the entire curve from the first to the last point is affected.

Bezier Curves

- Bezier curves are named after the French engineer P. Bezier of the Renault Automobile Company. He developed them in the early 1960s to fill a need for curves whose shape can be readily controlled by changing a few parameters. Bezier’s application was to construct pleasing surfaces for car bodies.

Suppose we are given a set of control points, \( p_i = (x_i, y_i), \ i = 0, 1, 2, \ldots, n \). Figure 3.6 is an example.

The points do not necessarily progress from left to right. We treat the coordinates of each point as a two-component vector,

\[
p_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}.
\]

The set of points, in parametric form, is
\[ p(u) = \begin{bmatrix} x(u) \\ y(u) \end{bmatrix}, \ 0 \leq u \leq 1. \]

The \textit{nth-degree Bezier polynomial} determined by \( n+1 \) points is given by

\[ p(u) = \sum_{i=0}^{n} \frac{n!}{i!(n-i)!} (1-u)^{n-i} u^i \cdot p_i. \]

For \( n = 2 \), \( p(u) = 1 \cdot (1-u)^2 \cdot p_0 + 2 \cdot (1-u) \cdot u \cdot p_1 + 1 \cdot u^2 \cdot p_2 \). The preceding equation represents the pair of equations:

\[
\begin{align*}
x(u) &= (1-u)^2 x_0 + 2(1-u)u x_1 + u^2 x_2, \\
y(u) &= (1-u)^2 y_0 + 2(1-u)u y_1 + u^2 y_2.
\end{align*}
\]

Observe that \( x(0) = x_0, \ y(0) = y_0 \) and \( x(1) = x_2, \ y(1) = y_2 \). As \( u \) takes on values between 0 and 1, a curve is traced that goes from the first point to the third of the set. \textit{Ordinary the curve will not pass through the central point of the three.}

For \( n = 3 \), the \textit{Bezier cubic} is

\[
\begin{align*}
x(u) &= (1-u)^3 x_0 + 3(1-u)^2 u x_1 + 3(1-u)u^2 x_2 + u^3 x_3, \\
y(u) &= (1-u)^3 y_0 + 3(1-u)^2 u y_1 + 3(1-u)u^2 y_2 + u^3 y_3.
\end{align*}
\]

Observe again that \( p(0) = (x(0), y(0)) = p_0 \) and \( p(1) = (x(1), y(1)) = p_3 \), and that the curve will not ordinarily go through the intermediate points. As illustrated in Figure 3.7, changing the intermediate points (control points) changes the shape of the curve.

![Bezier curves defined by four and seven points](image-url)
It is of interest to list the properties of Bezier cubics:

1) \( p(0) = p_0, \quad p(1) = p_3. \)

2) Because \( \frac{dx}{du} = 3(x_i - x_0) \) and \( \frac{dy}{du} = 3(y_i - y_0) \) at \( u = 0 \), the slope of the curve at \( u = 0 \) is \( \frac{dy}{dx} = \frac{y_i - y_0}{x_i - x_0} \), which is the slope of the secant line between \( p_0 \) and \( p_1 \). Similarly, the slope at \( u = 1 \) is the same as the secant line between the last two points.

3) The Bezier curve is contained in the \textbf{convex hull} determined by the four points.

The convex hull of a set of points is the smallest \textbf{convex set} that contains the points. A set, \( C \), is convex if and only if the line segment between any two points in the set lies entirely in set \( C \).

\begin{align*}
B_i(u) &= \sum_{k=-1}^{2} b_k p_{i+k}, \quad (3.19) \\
\text{where} \quad b_{-1} = \frac{1}{6}(1-u)^3, \quad b_0 = \frac{1}{6}(3u^3 - 6u^2 + 4), \quad b_1 = \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1), \quad b_2 = \frac{1}{6}u^3 \quad \text{with} \quad 0 \leq u \leq 1.
\end{align*}

B-Spline Curves

- B-Spline curves are like Bezier curves in that they do not ordinarily pass through the given data points. They can be of any degree, but we will concentrate on the cubic form.

Given the points \( p_i = (x_i, y_i), \ i = 0, 1, 2, \ldots, n \), the \textbf{cubic B-spline} for the interval \((p_i, p_{i+1}), \ i = 1, 2, \ldots, n-1\), is

\[ B_i(u) = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{bmatrix}. \quad (3.20) \]

If we write out the equations for \( x \) and \( y \), we get
The \( u \)-cubics act as weighting factors on the coordinates of the four successive points to generate the curve. For example, at \( u = 0 \), the weights applied are \( \frac{1}{6}, \frac{2}{3}, \frac{1}{6} \), and 0; at \( u = 1 \), they are 0, \( \frac{1}{6}, \frac{2}{3} \), and \( \frac{1}{6} \). These values vary throughout the interval from \( u = 0 \) to \( u = 1 \). Figure 3.8a and b show the effect of varying just one of the points, say \( p_2 \).

Because a set of four points is required to generate only a portion of the B-spline, we must consider how to get the B-spline for more than four points. The conditions that we impose on the B-spline are exactly the same as for the cubic splines: \textit{continuity of the curve and its first and second derivatives}.

Figure 3.9 shows how three successive parts of a B-spline might look (\( B_i \), \( B_{i+1} \) and \( B_{i+2} \)). And we summarize the properties of B-splines as follows:

1) Like the cubic splines, B-splines are pieced together so they agree at their joints in three ways:
   a) \( B_i(1) = B_{i+1}(0) = \frac{1}{6} (p_i + 4p_{i+1} + p_{i+2}) \),
   b) \( B_i'(1) = B_{i+1}'(0) = \frac{1}{2} (p_{i+2} - p_i) \),
   c) \( B_i''(1) = B_{i+1}''(0) = p_i - 2p_{i+1} + p_{i+2} \).

\[
x(u) = \frac{1}{6} (1 - u)^3 x_{i-1} + \frac{1}{6} (3u^3 - 6u^2 + 4)x_i + \frac{1}{6} (-3u^3 + 3u^2 + 3u + 1)x_{i+1} + \frac{1}{6} u^3 x_{i+2},
\]
\[
y(u) = \frac{1}{6} (1 - u)^3 y_{i-1} + \frac{1}{6} (3u^3 - 6u^2 + 4)y_i + \frac{1}{6} (-3u^3 + 3u^2 + 3u + 1)y_{i+1} + \frac{1}{6} u^3 y_{i+2}.
\]
2) The B-spline curve determined by each group of four points is within the convex hull of these points.

Now consider how to generate the ends of the jointed B-spline if we have points from \( p_1 \) to \( p_n \). We can add fictitious four points \( p_{-2} \), \( p_{-1} \), \( p_{n+1} \), and \( p_{n+2} \), with \( p_{-2} = p_{-1} = p_0 \) and \( p_{n+1} = p_{n+2} = p_n \). Then we will find that the new curves not only join properly with the portions already made, but start and end at the extreme points as we wanted.

- **B-Spline curves differ from Bezier curves in three ways:**
  1) For a B-spline, the curve in general does not begin and end at the extreme points.
  2) The slopes of the B-splines do not have any simple relationship to lines drawn between the points.
  3) The endpoints of the B-splines are in the vicinity of the two intermediate given points, but neither the \( x \)- nor the \( y \)-coordinates of these endpoints normally equal the coordinates of the intermediate points.

Figure 3.10 show B-splines that are defined by the same sets of points as the Bezier curves in Figure 3.7. There are significant differences.
Consider the following ten points,

<table>
<thead>
<tr>
<th>No.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>10</td>
<td>50</td>
<td>75</td>
<td>90</td>
<td>105</td>
<td>150</td>
<td>180</td>
<td>190</td>
<td>160</td>
<td>130</td>
</tr>
<tr>
<td>y</td>
<td>10</td>
<td>15</td>
<td>60</td>
<td>100</td>
<td>140</td>
<td>200</td>
<td>140</td>
<td>120</td>
<td>100</td>
<td>80</td>
</tr>
</tbody>
</table>

**a)** Plot the connected Bezier curves for points 0 to 3, 3 to 6 and 6 to 9.

**b)** Rewrite the Bezier equations so that the parameter $u$ is defined on $[0, 1]$ for points 0 to 3, on $[1, 2]$ for points 3 to 6, and on $[2, 3]$ for points 6 to 9.

**c)** Plot the connected B-spline curves from points 0 to 9.
3.5 Interpolating on a Surface

- An interpolation problem, faced by the National Weather Service, is to interpolate from scattered data to get values at points on a uniform grid. This very multi-dimensional problem is not an easy one.

There is a simplified form of interpolation that might be used to estimate a predicted value for the temperature at a grid point from data from weather stations located in its neighborhood. Suppose that the stations where the temperature is known are listed in Table 3.13:

<table>
<thead>
<tr>
<th>Station</th>
<th>Coordinates</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14.2, 25.6</td>
<td>56°F</td>
</tr>
<tr>
<td>2</td>
<td>22.7, 12.1</td>
<td>62°F</td>
</tr>
<tr>
<td>3</td>
<td>33.6, -2.5</td>
<td>59°F</td>
</tr>
<tr>
<td>4</td>
<td>-3.7, -8.3</td>
<td>64°F</td>
</tr>
<tr>
<td>5</td>
<td>13.4, 15.7</td>
<td>61°F</td>
</tr>
</tbody>
</table>

The coordinates of the known temperatures are relative to our desired grid point. This simplified weather problem gets the predicted temperature at the grid point by
\[
\frac{1}{8}(56 \times 2 + 62 \times 2 + 59 \times 2 + 64 \times 1 + 61 \times 1) = 59.9°F.
\]

The data from stations in the west are given double weight because weather comes generally from that direction.

Using Cubic Splines

- Another alternative is to use cubic splines for interpolation in multivariate cases. It is perhaps best to hold one variable constant while constructing one-way splines, then combine the results from these in the second phase.

An example of using MATLAB:
```matlab
>> y = [ 0  0.9  1.8  3.2];
>> x = 0;
>> z1 = 2*x*y + exp(x-y);
>> x = 1.2;
>> z2 = 2*x*y + exp(x-y);
>> x = 2.5;
>> z3 = 2*x*y + exp(x-y);
>> x = 2.9;
```
\[
\begin{align*}
\text{z4} &= 2x^2y + \exp(x-y); \\
\text{x} &= \begin{bmatrix} 0 & 1.2 & 2.5 & 2.9 \end{bmatrix}; \\
\text{z} &= \begin{bmatrix} z1 & z2 & z3 & z4 \end{bmatrix} \\
\text{z} &= \\
&= \begin{bmatrix} 1.0000 & 0.4066 & 0.1653 & 0.0408 \\
& 3.3201 & 3.5099 & 4.8688 & 7.8153 \\
\end{bmatrix} \\
\text{zlinear} &= \text{interp2(x,y,z,1.7,2.0, \text{'linear'})} \\
\text{zlinear} &= 10.4643 \\
\text{zicubic} &= \text{interp2(x,y,z,1.7,2.0, \text{'cubic'})} \\
\text{zicubic} &= 9.0978 \\
\end{align*}
\]

The true value of \(z\) is 7.5408. Use of the cubic splines is better than the linear interpolation. We would get results closer to the true value if the table were more closely spaced. A plot of the function shows that the \(z\)-values change rapid at \((x, y) = (1.7, 2.0)\).

**Creating a B-Spline Surface**

- From the previous section, we know that a cubic B-spline curve segment starting near the point \(p_i\) to near the point \(p_{i+1}\).

Analogous to the cubic B-spline curve, the interpolating cubic B-spline surface patch depends on 16 points as Figure 3.12 shows. Here \(p_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j})\) are points in \(E^3\). This patch is generated by computing the points \(p_{i,j}(u,v) = (x_{i,j}(u,v), y_{i,j}(u,v), z_{i,j}(u,v))\) for \(0 \leq u \leq 1\) and \(0 \leq v \leq 1\).

![Figure 3.12](image)

For simplicity, we will consider only the \(x\)-coordinate in detail. Comparable formulations
hold for the $y$- and $z$-coordinates. The simplest formulation for $x(u, v)$ is based on Eq. (3.20) and is given by

$$x_{i,j}(u,v) = \frac{1}{36} [u^3, u^2, u, 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{i,j-1-1} \\ x_{i,j-1} \\ x_{i+1,j-1} \\ x_{i+2,j-1} \end{bmatrix} \begin{bmatrix} v^3 \\ v^2 \\ v^1 \end{bmatrix}.$$ 

where

$$X_{i,j} = \begin{bmatrix} x_{i-1,j-1} & x_{i-1,j} & x_{i-1,j+1} & x_{i-1,j+2} \\ x_{i,j-1} & x_{i,j} & x_{i,j+1} & x_{i,j+2} \\ x_{i+1,j-1} & x_{i+1,j} & x_{i+1,j+1} & x_{i+1,j+2} \\ x_{i+2,j-1} & x_{i+2,j} & x_{i+2,j+1} & x_{i+2,j+2} \end{bmatrix}$$

as $u$ and $v$ range between 0 and 1. It is easily verified that the weights applied to each of the 16 points at the end points are

$$x_{i,j}(u = 0, v = 0) = \frac{1}{36} \begin{bmatrix} 1 & 4 & 1 & 0 \\ 4 & 16 & 4 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$x_{i,j}(u = 1, v = 1) = \frac{1}{36} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 4 & 16 & 4 \\ 0 & 1 & 4 & 1 \end{bmatrix}.$$ 

### 3.6 Least-Squares Approximations (最小誤差平方近似法)

- Until now, we have assumed that the data are accurate, but when these values are derived from an experiment, there is some error in the measurements. This section explains the usual method of treating such inaccurate data.

We would like a way of fitting a line to experimental data that is unambiguous and that, in some sense, minimizes the deviations of the points from the line. The usual method for doing this is called the **least-square method**.
Linear Data

- Let $Y_i$ represent an experimental value and $y_i$ be a value from the equation $y_i = ax_i + b$.

where $x_i$ is a particular value of the variable assumed to be free of error. We wish to determine the best values for $a$ and $b$ so that the $y_i$’s predict the function values that correspond to $x$-values. Also let $e_i = Y_i - y_i$. The least-squares criterion requires that

$$S = e_i^2 + e_2^2 + \cdots + e_N^2 = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} (Y_i - ax_i - b)^2$$

be a minimum. $N$ is the number of $(x, y)$-pairs. We reach the minimum by proper choice of the parameters $a$ and $b$. At a minimum for $S$, it is required

$$\frac{\partial S}{\partial a} = 0 \Rightarrow \sum_{i=1}^{N} 2(Y_i - ax_i - b)(-x_i) = 0,$$

$$\frac{\partial S}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N} 2(Y_i - ax_i - b)(-1) = 0.$$  

Dividing each of these equations by $-2$, we have

$$a \sum_{i=1}^{N} x_i^2 + b \sum_{i=1}^{N} x_i = \sum_{i=1}^{N} x_i Y_i$$ and

$$a \sum_{i=1}^{N} x_i + bN = \sum_{i=1}^{N} Y_i.$$  

Solving these equations simultaneously gives the values for slope and intercept $a$ and $b$.

An example by using MATLAB

```matlab
>> x = [20.5 32.7 51.0 73.2 95.7];
>> y = [765 826 873 942 1032];
>> eq = polyfit(x, y, 1)
```

```
ans =
3.3949 702.1721
```

Figure 3.13
**Nonlinear Data**

- Whenever data from experimental tests are not linear, we need to fit to them some function other than a first-degree polynomial. Popular forms that are tried are the power form

\[ y = ax^b \]

or the exponential form

\[ y = ae^{bx} \]

Since such nonlinear equations are much more difficult to solve than linear equations, they are usually linearized by taking logarithms before determining the parameters:

\[ \ln y = \ln a + b \ln x, \]

or

\[ \ln y = \ln a + bx. \]

We now fit the new variable \( z = \ln y \) as a linear function of \( \ln x \) or \( x \) as described earlier.

In cases when such linearization of the function is not desirable, or when no method of linearization can be discovered, graphical methods are frequently used; one merely plots the experimental values and sketches in a curve that seems to fit well.

**Least-Squares Polynomials (最小誤差平方多項式)**

- In the development, we use \( n \) as the degree of the polynomial and \( N \) as the number of data pairs \((x_i, y_i)\). We will always have \( N > n + 1 \) in the following.

Assume the functional relationship for fitting

\[ Y(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \]

with errors defined by

\[ e_i = y_i - Y(x_i) = y_i - a_0 - a_1 x_i - a_2 x_i^2 - \cdots - a_n x_i^n, \]

in which \( i = 1, 2, 3, \ldots, N. \)

We minimize the sum of error squares,

\[ S = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} (y_i - a_0 - a_1 x_i - a_2 x_i^2 - \cdots - a_n x_i^n)^2. \]

At the minimum, all the first partial derivatives with respect to \( a_i \)'s vanish. We have
\[
\frac{\partial S}{\partial a_0} = 0 = 2 \sum_{i=1}^{N} (y_i - a_0 - a_1 x_i - a_2 x_i^2 - \cdots - a_n x_i^n)(-1),
\]

\[
\frac{\partial S}{\partial a_1} = 0 = 2 \sum_{i=1}^{N} (y_i - a_0 x_i - a_2 x_i^2 - \cdots - a_n x_i^n)(-x_i),
\]

\[
\frac{\partial S}{\partial a_2} = 0 = 2 \sum_{i=1}^{N} (y_i - a_0 x_i - a_2 x_i^2 - \cdots - a_n x_i^n)(-x_i^2),
\]

\[\vdots\]

\[
\frac{\partial S}{\partial a_n} = 0 = 2 \sum_{i=1}^{N} (y_i - a_0 x_i - a_2 x_i^2 - \cdots - a_n x_i^n)(-x_i^n),
\]

Rearrange them to get

\[
a_0 N + a_1 \sum_{i=1}^{N} x_i + a_2 \sum_{i=1}^{N} x_i^2 + \cdots + a_n \sum_{i=1}^{N} x_i^n = \sum_{i=1}^{N} y_i,
\]

\[
a_0 \sum_{i=1}^{N} x_i + a_1 \sum_{i=1}^{N} x_i^2 + a_2 \sum_{i=1}^{N} x_i^3 + \cdots + a_n \sum_{i=1}^{N} x_i^{n+1} = \sum_{i=1}^{N} x_i y_i,
\]

\[
a_0 \sum_{i=1}^{N} x_i^2 + a_1 \sum_{i=1}^{N} x_i^3 + a_2 \sum_{i=1}^{N} x_i^4 + \cdots + a_n \sum_{i=1}^{N} x_i^{n+2} = \sum_{i=1}^{N} x_i^2 y_i,
\]

\[\vdots\]

\[
a_0 \sum_{i=1}^{N} x_i^n + a_1 \sum_{i=1}^{N} x_i^{n+1} + a_2 \sum_{i=1}^{N} x_i^{n+2} + \cdots + a_n \sum_{i=1}^{N} x_i^{2n} = \sum_{i=1}^{N} x_i^n y_i,
\]

or, in matrix form,

\[
\begin{bmatrix}
N \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \cdots & \sum_{i=1}^{N} x_i^n \\
\sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \cdots & \sum_{i=1}^{N} x_i^{n+1} \\
\sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i^3 & \cdots & \sum_{i=1}^{N} x_i^{n+2} \\
\vdots & \vdots & \ddots & \vdots \\
\sum_{i=1}^{N} x_i^n & \sum_{i=1}^{N} x_i^{n+1} & \cdots & \sum_{i=1}^{N} x_i^{2n}
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_n
\end{bmatrix}
=
\begin{bmatrix}
\sum_{i=1}^{N} y_i \\
\sum_{i=1}^{N} x_i y_i \\
\sum_{i=1}^{N} x_i^2 y_i \\
\vdots \\
\sum_{i=1}^{N} x_i^n y_i
\end{bmatrix}.
\]

Equations (3.28) represents a linear system. However, this system is usually ill-conditioned and round-off errors can distort the solution of \(a_i\)'s. Up to degree-3 or 4, the problem is not too great. It is very infrequent to use a degree higher than 4.

Example  A Quadratic Polynomial for Fitting

The data are actually a perturbation of the relation \(y = 1 - x + 0.2 x^2\).
The equations to be solved are

\[ \begin{align*}
11a_0 + 6.01a_1 + 4.6545a_2 &= 5.905, \\
6.01a_0 + 4.6545a_1 + 4.1150a_2 &= 2.1839, \\
4.6545a_0 + 4.1150a_1 + 3.9161a_2 &= 1.3357.
\end{align*} \]

The result is \( a_0 = 0.998, \quad a_1 = -1.018, \quad a_2 = 0.225 \).

So the least squares method gives

\[ Y(x) = 0.998 - 1.018x + 0.225x^2, \]

which is comparable to \( y = 1 - x + 0.2x^2 \).

What Degree of Polynomial Should Be Used? (多項式次數之選擇)

- The answer to this problem is found in statistics. One can increase the degree of polynomial as long as there is a statistically significant decrease in the variance,

\[ \sigma^2 = \frac{\sum_{i=1}^{N} e_i^2}{N - n - 1}. \]

For the preceding example in Table 3.14, when the degree of polynomial made to fit the points is varied from 1 to 5, we obtain the results shown in Table 3.15. The optimal degree chosen is 2.

<table>
<thead>
<tr>
<th>Degree</th>
<th>Equation</th>
<th>(\sigma^2) (Eq. 3.27)</th>
<th>(\sum e^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( y = 0.95228 - 0.76041x )</td>
<td>0.00106</td>
<td>0.00915</td>
</tr>
<tr>
<td>2</td>
<td>( y = 0.99800 - 1.0180x + 0.22468x^2 )</td>
<td>0.00023</td>
<td>0.00187</td>
</tr>
<tr>
<td>3</td>
<td>( y = 1.00337 - 1.0794x + 0.35137x^2 - 0.06894x^3 )</td>
<td>0.00026</td>
<td>0.00181</td>
</tr>
<tr>
<td>4</td>
<td>( y = 0.98810 - 0.83690x - 0.52680x^2 + 1.0461x^3 - 0.45635x^4 )</td>
<td>0.00027</td>
<td>0.00165</td>
</tr>
<tr>
<td>5</td>
<td>( y = 1.0369 - 1.8241x + 4.8953x^2 - 10.753x^3 + 10.537x^4 - 3.6594x^5 )</td>
<td>0.00013</td>
<td>0.00067</td>
</tr>
</tbody>
</table>
4 Approximation of Functions (函數之近似)

• It is wondered for us to know how the computer gets the value of sin(2.113) or \( e^{-3.5} \). It doesn’t need to look these up in tables and interpolate. The computer approximates every function other than polynomials from some polynomial that is tailored to give very accurate values.

• This chapter describes how such approximating polynomials are developed. We want the approximation to be efficient in that it obtains the values with the smallest error in the least number of arithmetic operations (以最少的算數運算次數來達到最準確的近似值).

4.1 Chebyshev Polynomials and Chebyshev Series

Chebyshev Polynomials
• Chebyshev’s differential equation is

\[
(1 - x^2)y'' - xy' + \lambda y = 0
\]

in which \(-1 \leq x \leq 1\).

For \( \lambda = n^2 \) \((n=0, 1, 2,\ldots)\), solutions of the differential equation, \( T_n(x) \), are called the Chebyshev polynomials. With arbitrary constant chosen so that \( T_n(1) = 1 \), we have the first 11 of Chebyshev polynomials as

\[
\begin{align*}
T_0(x) & = 1, \\
T_1(x) & = x, \\
T_2(x) & = 2x^2 - 1,
\end{align*}
\]

Figure 4.1
The members of this series of polynomials can be generated from the two-term recursion formula

\[ T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \quad \text{with} \quad T_0(x) = 1, \quad T_1(x) = x. \]

These polynomials form an orthogonal set in that

\[
\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} T_n(x)T_m(x)dx = \begin{cases} 0, & n \neq m \\ \pi, & n = m = 0 \\ \pi/2, & n = m \neq 0 \end{cases}
\]

in which \( n \) and \( m \) are distinct non-negative integers.

**Smallest Upper Bound / Smallest Maximum Error**

- It is most important to note that “the polynomial

  \[ \frac{1}{2^{n-1}}T_n(x), \]

  owning a coefficient of one on \( x^n \) term, has a smaller upper bound (有最小上限) to its magnitude in the interval \([-1, 1]\) than any other”. It means that for all polynomials of degree \( n \), whose highest power of \( x \) has a coefficient of one, \((1/2^{n-1})T_n(x)\) has the smallest error bounds on \([-1, 1]\). This is important because we will be able to write power function approximations to functions whose maximum errors are given in terms of this upper bound.

The proof is given by **contradiction** (矛盾法):

1. Let \( P_n(x) \) be a polynomial whose leading term is \( 1 \cdot x^n \) and suppose that its maximum magnitude on \([-1, 1]\) is less than that of \((1/2^{n-1})T_n(x)\).
2. Define

\[ P_{n-1}(x) \equiv \frac{1}{2^{n-1}} T_n(x) - P_n(x), \]

where \( P_{n-1}(x) \) is a polynomial of degree \( n - 1 \) or less, as the \( x^n \) terms cancel. Note that the polynomial \( T_n(x) \) has \( n + 1 \) extremes (including endpoints), each of magnitude 1. Therefore, \( (1/2^{n-1})T_n(x) \) has \( n + 1 \) extremes each of magnitude \( 1/2^{n-1} \), and these successive extremes alternative in sign.

3. By our supposition about \( P_n(x) \), the magnitude of \( P_n(x) \) is less than \( 1/2^{n-1} \) at each of these maxima and minima. Hence, \( P_{n-1}(x) \) must change its sign at least \( n \) times within \( n + 1 \) extremes of \( T_n(x) \). However, it is impossible because \( P_{n-1}(x) \) is only of degree \( n - 1 \).

Economizing a Power Series (次方級數省去法)

- Consider the Maclaurin series for \( e^x \):

\[
e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} + \ldots
\]

\[
\approx 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} \quad \text{(Maclaurin series)}
\]

\[
\approx 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} - \frac{1}{720} \left( \frac{1}{2^{n-1}} \right) T_6(x)
\]

\[
\approx 1.000043 + x + 0.499219x^2 + \frac{x^3}{6} + 0.043750x^4 + \frac{x^5}{120} \quad \text{(Economized series)}
\]

<table>
<thead>
<tr>
<th>( x )</th>
<th>( e^x )</th>
<th>( x )</th>
<th>( e^x )</th>
<th>( x )</th>
<th>( e^x )</th>
<th>( x )</th>
<th>( e^x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>1.000000</td>
<td>6</td>
<td>1.000000</td>
<td>6</td>
<td>1.000000</td>
<td>6</td>
<td>1.000000</td>
</tr>
<tr>
<td>0.2</td>
<td>1.22140</td>
<td>1.22140</td>
<td>1.22140</td>
<td>1.22140</td>
<td>1.22140</td>
<td>1.22140</td>
<td>1.22140</td>
</tr>
<tr>
<td>0.4</td>
<td>1.49182</td>
<td>1.49182</td>
<td>1.49182</td>
<td>1.49182</td>
<td>1.49173</td>
<td>1.49173</td>
<td>1.49173</td>
</tr>
<tr>
<td>0.6</td>
<td>1.82212</td>
<td>1.82212</td>
<td>1.82205</td>
<td>1.82205</td>
<td>1.822140</td>
<td>1.822140</td>
<td>1.822140</td>
</tr>
<tr>
<td>0.8</td>
<td>2.22554</td>
<td>2.22549</td>
<td>2.32513</td>
<td>2.32513</td>
<td>2.22240</td>
<td>2.22240</td>
<td>2.22240</td>
</tr>
<tr>
<td>1.0</td>
<td>2.71828</td>
<td>2.71806</td>
<td>2.71667</td>
<td>2.71667</td>
<td>2.70833</td>
<td>2.70833</td>
<td>2.70833</td>
</tr>
<tr>
<td><strong>Maximum error</strong></td>
<td>0.00023</td>
<td>0.00162</td>
<td>0.00995</td>
<td>0.00995</td>
<td>0.00027</td>
<td>0.00027</td>
<td>0.00027</td>
</tr>
</tbody>
</table>

Notice that near \( x = 0 \), the economized polynomials are less accurate. In effect, we permit errors at points within the range but get a smaller maximum error.
Chebyshev Series

- By rearranging the Chebyshev polynomials, we can express powers of $x$ in terms of them:

  \[
  1 = T_0, \\
  x = T_1, \\
  x^2 = \frac{1}{2}(T_0 + T_2), \\
  x^3 = \frac{1}{4}(3T_1 + T_3), \\
  x^4 = \frac{1}{8}(3T_0 + 4T_2 + T_4), \\
  x^5 = \frac{1}{16}(10T_1 + 5T_3 + T_5), \\
  x^6 = \frac{1}{32}(10T_0 + 15T_2 + 6T_4 + T_6), \\
  x^7 = \frac{1}{64}(35T_1 + 21T_3 + 7T_5 + T_7), \\
  x^8 = \frac{1}{128}(35T_0 + 56T_2 + 28T_4 + 8T_6 + T_8), \\
  x^9 = \frac{1}{256}(126T_1 + 84T_3 + 36T_5 + 9T_7 + T_9).
  \]

Consider the Maclaurin series for $e^x$:

\[
e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} + \cdots \quad \text{(Maclaurin series)}
\]

\[
= T_0 + T_1 + \frac{1}{2}(T_0 + T_2) + \frac{1}{3!}(3T_1 + T_3) + \frac{1}{5!}(3T_0 + 4T_2 + T_4) + \cdots \\
+ \frac{1}{1920}(10T_1 + 5T_3 + T_5) + \frac{1}{23040}(10T_0 + 15T_2 + 6T_4 + T_6) + \cdots \\
= 1.2661T_0 + 1.1302T_1 + 0.2715T_2 + 0.04433T_3 + \cdots \\
= 1.2661 + 1.1302x + 0.2715(2x^2 - 1) + 0.0443(4x^3 - 3x) + \cdots \\
= 0.9946 + 0.9973x + 0.5430x^2 + 0.1772x^3 + \cdots \quad \text{(Chebyshev series)}
\]

Table 4.2 and Figure 4.2 compare the error of the Chebyshev expansion with the Maclaurin series, using terms through $x^3$ in each case.
The computational economy to be gained by economizing a Maclaurin series or by using a Chebyshev series is even more dramatic when the Maclaurin series is slowly convergent.

Example 4.1 A Maclaurin series for \( \frac{1}{1+x} \) is

\[
\frac{1}{1+x} = 1 - x + x^2 - x^3 + x^4 - \cdots \quad (-1 < x < 1).
\]

Find the economized and the Chebyshev series for it.

Table 4.3 compares the accuracy of truncated Maclaurin series with the economized series derived from them.
In Tabel 4.3, we see that the error of the Maclaurin series is small for $x = 0.2$, whereas the economized polynomial has less accuracy. At $x = 0.8$, the situation is reversed. As for the Chebyshev series, it can give relatively accurate results

1) at significant savings of computational effort and

2) with smaller storage requirements in a computer’s memory for the coefficients of the polynomials.

<table>
<thead>
<tr>
<th>Degree</th>
<th>Maclaurin Value</th>
<th>Maclaurin Error</th>
<th>Chebyshev Value</th>
<th>Chebyshev Error</th>
<th>Economized Value</th>
<th>Economized Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.840000</td>
<td>0.006667</td>
<td>0.841035</td>
<td>0.007702</td>
<td>0.758600</td>
<td>-0.074733</td>
</tr>
<tr>
<td>4</td>
<td>0.833600</td>
<td>0.000267</td>
<td>0.83316</td>
<td>-0.000017</td>
<td>0.764594</td>
<td>-0.068739</td>
</tr>
<tr>
<td>6</td>
<td>0.833344</td>
<td>$1 \times 10^{-5}$</td>
<td>0.833359</td>
<td>0.000026</td>
<td>0.803646</td>
<td>-0.029687</td>
</tr>
<tr>
<td>8</td>
<td>0.833334</td>
<td>$1 \times 10^{-6}$</td>
<td>0.833365</td>
<td>0.000032</td>
<td>0.822786</td>
<td>-0.010547</td>
</tr>
<tr>
<td>10</td>
<td>0.833333</td>
<td>0</td>
<td>0.833365</td>
<td>0.000031</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

At $x = 0.2$

At $x = 0.8$

<table>
<thead>
<tr>
<th>Degree</th>
<th>Maclaurin Value</th>
<th>Maclaurin Error</th>
<th>Chebyshev Value</th>
<th>Chebyshev Error</th>
<th>Economized Value</th>
<th>Economized Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.840000</td>
<td>0.284445</td>
<td>0.549866</td>
<td>-0.005899</td>
<td>0.812600</td>
<td>0.257045</td>
</tr>
<tr>
<td>4</td>
<td>0.737600</td>
<td>0.182045</td>
<td>0.555518</td>
<td>-0.000038</td>
<td>0.738314</td>
<td>0.122759</td>
</tr>
<tr>
<td>6</td>
<td>0.672064</td>
<td>0.116509</td>
<td>0.555561</td>
<td>0.000006</td>
<td>0.628558**</td>
<td>0.073003</td>
</tr>
<tr>
<td>8</td>
<td>0.630121</td>
<td>0.074566</td>
<td>0.555568</td>
<td>0.000012</td>
<td>0.602106***</td>
<td>0.046551</td>
</tr>
<tr>
<td>10</td>
<td>0.603277</td>
<td>0.047722</td>
<td>0.555568</td>
<td>0.000012</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.2 Rational Function Approximations (有理式函數之近似)

- Approximating a known function with a Chebyshev series is much better than with a Taylor series in that it has a smaller maximum error in the interval $[-1, 1]$. However, there is still a way to improve further.

Pade Approximations

- A Pade approximation is a rational function, the quotient of two polynomials, the numerator of degree $n$ and the denominator of degree $m$. It can be written as

$$R_n(x) = \frac{a_0 + a_1x + a_2x^2 + \cdots + a_nx^n}{1 + b_1x + b_2x^2 + \cdots + b_mx^m}, \quad N = n + m.$$  

The constant term in the denominator can be taken as unity without loss of generality.
And note that the number of constants in \( R_N(x) \) is \( n + m + 1 \) or \( N + 1 \). The most useful of the Pade approximations are those with the degree of the numerator equal to, or one more than, the degree of the denominator (即 \( n=m \) 或 \( n=m+1 \)).

We begin to use \( R_N(x) \) (a Pade approximation) to approximate \( f_N(x) \) (a Maclaurin series of terms through \( x^N \)). Consider

\[
\frac{f_N(x) - R_N(x)}{f_N(x) - R_N(x)} = \frac{a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n}{1 + b_1 x + b_2 x^2 + \cdots + b_m x^m}
\]

Set the coefficients of the \( x \)-term through \( x^N \) in the numerator to zero. This gives \( N+1 \) equations to solve for the \( a \)'s and \( b \)'s:

\[
c_0 - a_0 = 0,
\]

\[
b_1 c_0 + c_1 - a_1 = 0,
\]

\[
b_2 c_0 + b_1 c_1 + c_2 - a_2 = 0,
\]

\[
b_3 c_0 + b_2 c_1 + b_1 c_2 + c_3 - a_3 = 0,
\]

\[
\vdots
\]

\[
b_m c_{n-m} + b_{m-1} c_{n-m+1} + \cdots + c_n - a_n = 0,
\]

\[
b_m c_{n-m+1} + b_{m-1} c_{n-m+2} + \cdots + c_{n+1} = 0,
\]

\[
b_m c_{n-m+2} + b_{m-1} c_{n-m+3} + \cdots + c_{n+2} = 0,
\]

\[
\vdots
\]

\[
b_m c_{N-m} + b_{m-1} c_{N-m+1} + \cdots + c_N = 0.
\]

**Example 4.2** Find the Pade approximation \( R_{10}(x) \) for \( \tan^{-1}(x) \). Use degree-5 in both numerator and denominator (\( n=5 \) and \( m=5 \)).

The Maclaurin series through \( x^{10} \) is

\[
\tan^{-1}(x) \approx f_{10}(x) = x - \frac{1}{3} x^3 + \frac{1}{5} x^5 - \frac{1}{7} x^7 + \frac{1}{9} x^9.
\]

We form

\[
f_{10}(x) - R_{10}(x)
\]
If we multiply out in the numerator and set the coefficients of the $x$-terms through $x^{10}$ to zero, we get for the $a$'s:

\[
\begin{align*}
\frac{1}{a_0} &= 1, \\
\frac{1}{a_1} &= \frac{1}{a_2} = b_1, \\
\frac{1}{a_3} &= -\frac{1}{a_4} = b_2, \\
\frac{1}{a_5} &= -\frac{1}{a_6} = b_3 \\
\end{align*}
\]

and for the $b$'s:

\[
\begin{align*}
\frac{1}{a_0} - \frac{1}{a_1} &= 0, \\
\frac{1}{a_1} + \frac{1}{a_2} - \frac{1}{a_3} &= 0, \\
\frac{1}{a_2} + \frac{1}{a_3} - \frac{1}{a_4} &= 0, \\
\frac{1}{a_3} - \frac{1}{a_4} + \frac{1}{a_5} &= 0, \\
\frac{1}{a_4} - \frac{1}{a_5} + \frac{1}{a_6} &= 0.
\end{align*}
\]

The last five equations are used to solve the $b$'s:

\[
\begin{align*}
b_1 &= 0, \\
b_2 &= 0, \\
b_3 &= 0, \\
b_4 &= \frac{5}{21}, \\
b_5 &= 0.
\end{align*}
\]

Then the first six equations are used to solve the $a$'s:

\[
\begin{align*}
a_0 &= 0, \\
a_1 &= 1, \\
a_2 &= 0, \\
a_3 &= \frac{7}{9}, \\
a_4 &= 0, \\
a_5 &= \frac{64}{945}.
\end{align*}
\]

Thus we have the Pade approximation

\[
R_{10}(x) = \frac{x + \frac{7}{9} x^3 + \frac{64}{945} x^5}{1 + \frac{10}{9} x^2 + \frac{5}{21} x^4}.
\]

<table>
<thead>
<tr>
<th>$x$</th>
<th>True value</th>
<th>Padé (Eq. 4.12)</th>
<th>Error</th>
<th>Maclaurin (Eq. 4.10)</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.19740</td>
<td>0.19740</td>
<td>0.00000</td>
<td>0.19740</td>
<td>0.00000</td>
</tr>
<tr>
<td>0.4</td>
<td>0.38051</td>
<td>0.38051</td>
<td>0.00000</td>
<td>0.38051</td>
<td>0.00000</td>
</tr>
<tr>
<td>0.6</td>
<td>0.54042</td>
<td>0.54042</td>
<td>0.00000</td>
<td>0.54067</td>
<td>-0.00025</td>
</tr>
<tr>
<td>0.8</td>
<td>0.67477</td>
<td>0.67477</td>
<td>-0.00003</td>
<td>0.67982</td>
<td>-0.00508</td>
</tr>
<tr>
<td>1.0</td>
<td>0.78540</td>
<td>0.78558</td>
<td>-0.00018</td>
<td>0.83492</td>
<td>-0.04952</td>
</tr>
</tbody>
</table>
In Table 4.4 we compare the errors for this Pade approximation to the Maclaurin series expansion.

Figure 4.3 shows how closely the Pade approximation matches \( \tan^{-1}(x) \), especially on \([-1, 1]\).
4.3 Fourier Series (傅立葉級數)

- Polynomials are not the only functions that can be used to approximate known functions. Another means for representing known functions are approximations that use sines and cosines, called Fourier series after the French mathematician who first proposed, in the early 1800s, that “any function can be represented by an infinite sum of sine and cosine terms.”

- Representing a function as a trigonometric series is important in solving some partial differential equations analytically.

Fourier Series in Standard Form

- Let \( f(x) \) be defined on \([-L, L]\). Then the Fourier series of \( f(x) \) is

\[
\frac{1}{2} a_0 + \sum_{n=1}^{\infty} \left[ a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right) \right].
\]

\[ (13.8) \]

where

\[
a_0 = \frac{1}{L} \int_{-L}^{L} f(x)dx,
\]

\[
a_n = \frac{1}{L} \int_{-L}^{L} f(x) \cos\left(\frac{n\pi x}{L}\right)dx \quad \text{for } n=1, 2, 3,\ldots
\]

\[
b_n = \frac{1}{L} \int_{-L}^{L} f(x) \sin\left(\frac{n\pi x}{L}\right)dx \quad \text{for } n=1, 2, 3,\ldots
\]

are the Fourier coefficients of \( f(x) \).

Proof:

1) To find \( a_0 \), integrate equation (13.8) term by term over the interval \([-L, L]\):

\[
\frac{1}{L} \int_{-L}^{L} f(x)dx = \frac{1}{L} \left[ \frac{1}{2} a_0 + \sum_{n=1}^{\infty} \left[ a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right) \right] \right]dx
\]

\[
= \frac{1}{2} a_0 \int_{-L}^{L} dx + \sum_{n=1}^{\infty} \left[ a_n \int_{-L}^{L} \cos\left(\frac{n\pi x}{L}\right)dx + b_n \int_{-L}^{L} \sin\left(\frac{n\pi x}{L}\right)dx \right]
\]

\[-89-\]
\[
\frac{1}{2} a_0 \cdot 2L = a_0 \cdot L
\]

\[\therefore a_0 = \frac{1}{L} \int_{-L}^{L} f(x) dx\]

2) To find \(a_n\), multiply equation (13.8) by \(\cos(k\pi x/L)\) and integrate each term to get

\[
\int_{-L}^{L} f(x) \cos\left(\frac{k\pi x}{L}\right) dx = \int_{-L}^{L} \left[ \frac{1}{2} a_0 + \sum_{n=1}^{\infty} \left( a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right) \right) \right] \cos\left(\frac{k\pi x}{L}\right) dx
\]

\[= \frac{1}{2} a_0 \int_{-L}^{L} \cos\left(\frac{k\pi x}{L}\right) dx + \sum_{n=1}^{\infty} \left[ a_n \int_{-L}^{L} \cos\left(\frac{n\pi x}{L}\right) \cos\left(\frac{k\pi x}{L}\right) dx + b_n \int_{-L}^{L} \sin\left(\frac{n\pi x}{L}\right) \cos\left(\frac{k\pi x}{L}\right) dx \right]
\]

\[= a_k \int_{-L}^{L} \cos^2\left(\frac{k\pi x}{L}\right) dx = a_k \int_{-L}^{L} \frac{1 + \cos(2k\pi x/L)}{2} dx = a_k \cdot \frac{1}{2} \cdot 2L = a_k \cdot L
\]

\[\therefore a_n = \frac{1}{L} \int_{-L}^{L} f(x) \cos\left(\frac{n\pi x}{L}\right) dx\]

3) To find \(b_n\), multiply equation (13.8) by \(\sin(k\pi x/L)\) and integrate each term to get

\[
\int_{-L}^{L} f(x) \sin\left(\frac{k\pi x}{L}\right) dx = \ldots \ldots = b_k \cdot L
\]

\[\therefore b_n = \frac{1}{L} \int_{-L}^{L} f(x) \sin\left(\frac{n\pi x}{L}\right) dx\]

**Fourier Series in Phase Angle Form**

- Consider the Fourier series of a function \(f(x)\).

\[
\frac{1}{2} a_0 + \sum_{n=1}^{\infty} \left[ a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right) \right]
\]

\[= \frac{1}{2} a_0 + \sum_{n=1}^{\infty} \sqrt{a_n^2 + b_n^2} \left[ \frac{a_n}{\sqrt{a_n^2 + b_n^2}} \cos\left(\frac{n\pi x}{L}\right) + \frac{b_n}{\sqrt{a_n^2 + b_n^2}} \sin\left(\frac{n\pi x}{L}\right) \right]
\]

let \(\theta_n = \tan^{-1}\left(\frac{b_n}{a_n}\right)\) and \(\delta_n = -\theta_n\),

then \(\cos(\delta_n) = \frac{a_n}{\sqrt{a_n^2 + b_n^2}}\), \(\sin(\delta_n) = \frac{-b_n}{\sqrt{a_n^2 + b_n^2}}\)

\[= \frac{1}{2} a_0 + \sum_{n=1}^{\infty} \sqrt{a_n^2 + b_n^2} \left[ \cos(\delta_n) \cos\left(\frac{n\pi x}{L}\right) - \sin(\delta_n) \sin\left(\frac{n\pi x}{L}\right) \right]
\]

- 90 -
with known

$$\cos(\alpha + \beta) = \cos \alpha \cos \beta - \sin \alpha \sin \beta$$

$$= \frac{1}{2} a_0 + \sum_{n=1}^{\infty} c_n \cos\left(\frac{n\pi x}{L} + \delta_n\right)$$

$$= \frac{1}{2} a_0 + \sum_{n=1}^{\infty} \left[ a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right) \right]$$

$$= \frac{1}{2} a_0 + \sum_{n=1}^{\infty} \left[ a_n \cdot \frac{1}{2} (e^{i n \omega x} + e^{-i n \omega x}) + b_n \cdot \frac{1}{2i} (e^{i n \omega x} - e^{-i n \omega x}) \right]$$

$$= \frac{1}{2} a_0 + \sum_{n=1}^{\infty} \left[ \frac{1}{2} (a_n - ib_n) e^{i n \omega x} + \frac{1}{2} (a_n + ib_n) e^{-i n \omega x} \right]$$

which is called the phase angle form of the Fourier series.

### Fourier Series in Complex Form

- Consider the Fourier series of a function \( f(x) \).

$$\frac{1}{2} a_0 + \sum_{n=1}^{\infty} \left[ a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right) \right]$$

$$= \frac{1}{2} a_0 + \sum_{n=1}^{\infty} \left[ a_n \cdot \frac{1}{2} (e^{i n \omega x} + e^{-i n \omega x}) + b_n \cdot \frac{1}{2i} (e^{i n \omega x} - e^{-i n \omega x}) \right]$$

$$= \frac{1}{2} a_0 + \sum_{n=1}^{\infty} \left[ \frac{1}{2} (a_n - ib_n) e^{i n \omega x} + \frac{1}{2} (a_n + ib_n) e^{-i n \omega x} \right]$$

Let

$$d_0 = \frac{1}{2} a_0, \quad d_n = \frac{1}{2} (a_n - ib_n) \quad \text{and} \quad \overline{d}_n = \frac{1}{2} (a_n + ib_n),$$

then we have

$$d_n = \frac{1}{2} (a_n - ib_n) = \frac{1}{2} \left[ \int_{-L}^{L} f(x)[\cos(n \omega_0 x) - i \sin(n \omega_0 x)] dx \right]$$

$$= \frac{1}{2L} \left[ \int_{-L}^{L} f(x)[\cos(-n \omega_0 x) + i \sin(-n \omega_0 x)] dx \right]$$

$$= \frac{1}{2L} \int_{-L}^{L} f(x) e^{-i n \omega_0 x} dx$$

$$= d_0 + \sum_{n=1}^{\infty} \left[ d_n e^{i n \omega_0 x} + \overline{d}_n e^{-i n \omega_0 x} \right]$$

$$= d_0 + \sum_{n=1}^{\infty} d_n e^{i n \omega_0 x} + \sum_{n=1}^{\infty} \overline{d}_n e^{-i n \omega_0 x}$$

$$= d_0 + \sum_{n=1}^{\infty} d_n e^{i n \omega_0 x} + \sum_{n=1}^{\infty} \overline{d}_n e^{-i n \omega_0 x} = d_0 + \sum_{n=1}^{\infty} d_n e^{i n \omega_0 x}$$

$$= \sum_{n=-\infty}^{\infty} d_n e^{i n \omega_0 x} \quad \text{and} \quad d_n = \frac{1}{2L} \int_{-L}^{L} f(x) e^{-i n \omega_0 x} dx ,$$

which is called the complex form of the Fourier series.
The Gibbs Phenomenon (Gibbs 現象)

The behavior of Fourier series at jump discontinuities of the function is known as the Gibbs phenomenon. It means that the peaks in neighbor of the jump discontinuities do not become smaller as $N \to \infty$. Instead, the peaks maintain roughly the same height, but move closer to the discontinuities as $N$ increases.

**Example 4.3** Find the partial sum of the Fourier series of the function

$$f(x) = \begin{cases} 0 & \text{for } -2 < x < 0 \\ 2 - x & \text{for } 0 \leq x \leq 2 \end{cases}$$

The Fourier series is

$$\frac{1}{2}a_0 + \sum_{n=1}^{\infty} \left[ a_n \cos\left( \frac{n\pi x}{2} \right) + b_n \sin\left( \frac{n\pi x}{2} \right) \right],$$

where $a_0 = \frac{1}{2} \int_{-2}^{2} f(x)dx$, $a_n = \frac{1}{2} \int_{-2}^{2} f(x)\cos\left( \frac{n\pi x}{2} \right)dx$ and $b_n = \frac{1}{2} \int_{-2}^{2} f(x)\sin\left( \frac{n\pi x}{2} \right)dx$.

```c
#include <stdio.h>
#include <math.h>
#define Npart 10
#define Ndiv 40

float funcx( float );

int main()

{  
    int i , n;
    float L=2.0 , dx , x , fx , fs , thetax , pi;
    float a0 , an[Npart] , bn[Npart];
    FILE *fPtr1 , *fPtr2;

    pi = 4.0 * atanf( 1.0 );

    fPtr1 = fopen( "coeff.dat" , "w" );
    fprintf( fPtr1 , "%s %8s %10s \n" , "n" , "an" , "bn" );
    fPtr2 = fopen( "output.dat" , "w" );
    fprintf( fPtr2 , "%s %11s %13s \n" , "x" , "fx" , "fseries" );
```
/* Find a0, an, bn */
a0 = 0.0;
for ( n = 1 ; n <= Npart ; n++ ) {
an[ n ] = 0.0;
bn[ n ] = 0.0;
}
dx = 2.0 * L / Ndiv;
x = -L + 0.5 * dx;
for ( i = 1 ; i <= Ndiv ; i++ ) {
    fx = funcx( x );
a0 += fx;
    for ( n = 1 ; n <= Npart ; n++ ) {
        thetax = n * pi * x / L;
        an[ n ] += fx * cosf( thetax );
        bn[ n ] += fx * sinf( thetax );
    }
    x += dx;
}
a0 = a0 * dx / L;
fprintf( fPtr1 , "%6d %10.6f
", 0 , a0 );
for ( n = 1 ; n <= Npart ; n++ ) {
an[ n ] = an[ n ] * dx / L;
bn[ n ] = bn[ n ] * dx / L;
fprintf( fPtr1 , "%6d %10.6f %10.6f\n", n , an[n] , bn[n] );
}
/* inverse transform */
x = -L + 0.5 * dx;
for ( i = 1 ; i <= Ndiv ; i++ ) {
    fs = 0.5 * a0;
    for ( n = 1 ; n <= Npart ; n++ ) {
        thetax = n * pi * x / L;
        fs += an[ n ] * cosf( thetax ) + bn[ n ] * sinf( thetax );
    }
    fprintf( fPtr2 , "%10.6f %10.6f %10.6f\n", x , funcx(x) , fs );
    x += dx;
}
fclose( fPtr1 );
fclose( fPtr2 );
return 0;

/* Function evaluation */
float funcx( float x ) {
    if ( -2.0 <= x && x < 0.0 )
        return 0.0;
    else
        return 2.0-x;
}

\n
執�行結果 — coeff.dat
n an bn
0 1.000000 0
1 0.404876 0.637275
2 -0.000000 0.319623
3 0.044607 0.214183
4 -0.000000 0.161803
5 0.015771 0.130656
6 -0.000000 0.110134
7 0.007808 0.095694
8 -0.000000 0.085065
9 0.004507 0.076989
10 -0.000000 0.070711

執行動結果 — output.dat
Example 4.4  Find the Fourier series expansion of the function

\[ f(x) = 3\cos(\frac{3\pi x}{2}) - 5\sin(\frac{5\pi x}{2}) + 7\cos(\frac{7\pi x}{2}) \quad \text{for} \quad -2 \leq x \leq 2. \]

```c
/* Function evaluation */
float funcx( float x )
{
    float pi, a1;
    pi = 4.0 * atanf( 1.0 );
    a1 = 0.5 * pi * x;
    return 3.0*cosf(3.0*a1) - 5.0*sinf(5.0*a1) + 7.0*cos(7.0*a1);
}
```

<table>
<thead>
<tr>
<th>x</th>
<th>fx</th>
<th>fseries</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.950000</td>
<td>0.000000</td>
<td>0.028351</td>
</tr>
<tr>
<td>-1.850000</td>
<td>0.000000</td>
<td>0.015397</td>
</tr>
<tr>
<td>-1.750000</td>
<td>0.000000</td>
<td>-0.027640</td>
</tr>
<tr>
<td>-1.650000</td>
<td>0.000000</td>
<td>-0.014181</td>
</tr>
<tr>
<td>-1.550000</td>
<td>0.000000</td>
<td>-0.039097</td>
</tr>
<tr>
<td>-1.450000</td>
<td>0.000000</td>
<td>-0.019121</td>
</tr>
<tr>
<td>-1.350000</td>
<td>0.000000</td>
<td>-0.05585</td>
</tr>
<tr>
<td>-1.250000</td>
<td>0.000000</td>
<td>-0.041823</td>
</tr>
<tr>
<td>-1.150000</td>
<td>0.000000</td>
<td>-0.08941</td>
</tr>
<tr>
<td>-1.050000</td>
<td>0.000000</td>
<td>-0.048471</td>
</tr>
<tr>
<td>-0.950000</td>
<td>0.000000</td>
<td>-0.001011</td>
</tr>
<tr>
<td>-0.850000</td>
<td>0.000000</td>
<td>0.057420</td>
</tr>
<tr>
<td>-0.750000</td>
<td>0.000000</td>
<td>-0.023959</td>
</tr>
<tr>
<td>-0.650000</td>
<td>0.000000</td>
<td>-0.070939</td>
</tr>
<tr>
<td>-0.550000</td>
<td>0.000000</td>
<td>0.015256</td>
</tr>
<tr>
<td>-0.450000</td>
<td>0.000000</td>
<td>0.005305</td>
</tr>
<tr>
<td>-0.350000</td>
<td>0.000000</td>
<td>0.001054</td>
</tr>
<tr>
<td>-0.250000</td>
<td>0.000000</td>
<td>0.031767</td>
</tr>
<tr>
<td>-0.150000</td>
<td>0.000000</td>
<td>0.015397</td>
</tr>
<tr>
<td>-0.050000</td>
<td>0.000000</td>
<td>-0.048471</td>
</tr>
<tr>
<td>0.050000</td>
<td>1.950000</td>
<td>1.471649</td>
</tr>
<tr>
<td>0.150000</td>
<td>1.850000</td>
<td>2.015978</td>
</tr>
<tr>
<td>0.250000</td>
<td>1.750000</td>
<td>1.859015</td>
</tr>
<tr>
<td>0.350000</td>
<td>1.650000</td>
<td>1.549295</td>
</tr>
<tr>
<td>0.450000</td>
<td>1.550000</td>
<td>1.503346</td>
</tr>
<tr>
<td>0.550000</td>
<td>1.450000</td>
<td>1.524572</td>
</tr>
<tr>
<td>0.650000</td>
<td>1.350000</td>
<td>1.371552</td>
</tr>
<tr>
<td>0.750000</td>
<td>1.250000</td>
<td>1.189745</td>
</tr>
<tr>
<td>0.850000</td>
<td>1.150000</td>
<td>1.142337</td>
</tr>
<tr>
<td>0.950000</td>
<td>1.050000</td>
<td>1.100941</td>
</tr>
<tr>
<td>1.050000</td>
<td>0.950000</td>
<td>0.948470</td>
</tr>
<tr>
<td>1.150000</td>
<td>0.850000</td>
<td>0.805827</td>
</tr>
<tr>
<td>1.250000</td>
<td>0.750000</td>
<td>0.758418</td>
</tr>
<tr>
<td>1.350000</td>
<td>0.650000</td>
<td>0.688845</td>
</tr>
<tr>
<td>1.450000</td>
<td>0.550000</td>
<td>0.535824</td>
</tr>
<tr>
<td>1.550000</td>
<td>0.450000</td>
<td>0.415629</td>
</tr>
<tr>
<td>1.650000</td>
<td>0.350000</td>
<td>0.369681</td>
</tr>
<tr>
<td>1.750000</td>
<td>0.250000</td>
<td>0.280229</td>
</tr>
<tr>
<td>1.850001</td>
<td>0.149999</td>
<td>0.123267</td>
</tr>
<tr>
<td>1.950001</td>
<td>0.049999</td>
<td>0.028351</td>
</tr>
</tbody>
</table>
執行結果 — output.dat

<table>
<thead>
<tr>
<th>x</th>
<th>fx</th>
<th>fseries</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.95</td>
<td>-6.972178</td>
<td>-6.972217</td>
</tr>
<tr>
<td>-1.85</td>
<td>2.887379</td>
<td>2.887370</td>
</tr>
<tr>
<td>-1.75</td>
<td>9.938501</td>
<td>9.938507</td>
</tr>
<tr>
<td>-1.65</td>
<td>7.471641</td>
<td>7.471642</td>
</tr>
<tr>
<td>-1.55</td>
<td>-1.980037</td>
<td>-1.980042</td>
</tr>
<tr>
<td>-1.45</td>
<td>-8.868067</td>
<td>-8.868066</td>
</tr>
<tr>
<td>-1.35</td>
<td>-6.174778</td>
<td>-6.174771</td>
</tr>
<tr>
<td>-1.25</td>
<td>3.537015</td>
<td>3.537016</td>
</tr>
<tr>
<td>-1.15</td>
<td>10.840187</td>
<td>10.840182</td>
</tr>
<tr>
<td>-1.05</td>
<td>8.977217</td>
<td>8.977211</td>
</tr>
<tr>
<td>-0.95</td>
<td>0.261558</td>
<td>0.261563</td>
</tr>
<tr>
<td>-0.85</td>
<td>-7.013356</td>
<td>-7.013354</td>
</tr>
<tr>
<td>-0.75</td>
<td>-7.363833</td>
<td>-7.363835</td>
</tr>
<tr>
<td>-0.65</td>
<td>-3.064002</td>
<td>-3.064001</td>
</tr>
<tr>
<td>-0.55</td>
<td>-0.370728</td>
<td>-0.370726</td>
</tr>
<tr>
<td>-0.45</td>
<td>-1.846802</td>
<td>-1.846801</td>
</tr>
<tr>
<td>-0.35</td>
<td>-3.644802</td>
<td>-3.644802</td>
</tr>
<tr>
<td>-0.25</td>
<td>-0.699694</td>
<td>-0.699690</td>
</tr>
<tr>
<td>-0.15</td>
<td>6.351420</td>
<td>6.351422</td>
</tr>
<tr>
<td>-0.05</td>
<td>10.799010</td>
<td>10.799006</td>
</tr>
<tr>
<td>0.05</td>
<td>6.972152</td>
<td>6.972149</td>
</tr>
<tr>
<td>0.15</td>
<td>-2.887422</td>
<td>-2.887421</td>
</tr>
<tr>
<td>0.25</td>
<td>-9.938512</td>
<td>-9.938511</td>
</tr>
<tr>
<td>0.35</td>
<td>-7.471617</td>
<td>-7.471619</td>
</tr>
<tr>
<td>0.45</td>
<td>1.980064</td>
<td>1.980063</td>
</tr>
<tr>
<td>0.55</td>
<td>8.868073</td>
<td>8.868073</td>
</tr>
<tr>
<td>0.65</td>
<td>6.174762</td>
<td>6.174760</td>
</tr>
<tr>
<td>0.75</td>
<td>-3.537037</td>
<td>-3.537035</td>
</tr>
<tr>
<td>0.85</td>
<td>-10.840194</td>
<td>-10.840187</td>
</tr>
<tr>
<td>0.95</td>
<td>-8.977199</td>
<td>-8.977193</td>
</tr>
<tr>
<td>1.05</td>
<td>-0.261539</td>
<td>-0.261543</td>
</tr>
<tr>
<td>1.15</td>
<td>7.013362</td>
<td>7.013354</td>
</tr>
<tr>
<td>1.25</td>
<td>7.363830</td>
<td>7.363825</td>
</tr>
<tr>
<td>1.35</td>
<td>3.063996</td>
<td>3.064004</td>
</tr>
<tr>
<td>1.45</td>
<td>0.370727</td>
<td>0.370727</td>
</tr>
<tr>
<td>1.55</td>
<td>1.846807</td>
<td>1.846807</td>
</tr>
<tr>
<td>1.65</td>
<td>3.644801</td>
<td>3.644804</td>
</tr>
<tr>
<td>1.75</td>
<td>0.699677</td>
<td>0.699688</td>
</tr>
<tr>
<td>1.85</td>
<td>-6.351444</td>
<td>-6.351445</td>
</tr>
<tr>
<td>1.95</td>
<td>-10.799011</td>
<td>-10.799043</td>
</tr>
</tbody>
</table>
5 **Numerical Differentiation and Integration**

- The heart of calculus is to find derivatives and integrals of functions that are exploited in many applications. We show in this chapter how derivatives and definite integrals can be computed with a computer program.

5.1 **Differentiation with a Computer**

- In calculus, the derivative of a function \( f(x) \) at \( x = x_0 \) is defined as

\[
f'(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}
\]

(前进差分; **forward difference**)  

\[
= \lim_{\Delta x \to 0} \frac{f(x_0) - f(x_0 - \Delta x)}{\Delta x}
\]

(后退差分; **backward difference**)  

A computer can approximate the derivative with smaller values of \( \Delta x \). But how small \( \Delta x \) should be? We should expect to find an optimal value for \( \Delta x \) because round-off errors in the numerator (分子) will become great as \( \Delta x \to 0 \).

Table 5.1 gives the results from a computer program for the computation of derivative of \( e^x \sin x \) at \( x = 1.9 \).

<table>
<thead>
<tr>
<th>( \Delta x )</th>
<th>Approximation</th>
<th>Error</th>
<th>Ratio of errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>4.05010</td>
<td>-0.11528</td>
<td></td>
</tr>
<tr>
<td>0.05/2</td>
<td>4.10955</td>
<td>-0.05583</td>
<td>2.06</td>
</tr>
<tr>
<td>0.05/4</td>
<td>4.13795</td>
<td>-0.02743</td>
<td>2.04</td>
</tr>
<tr>
<td>0.05/8</td>
<td>4.15176</td>
<td>-0.01362</td>
<td>2.01</td>
</tr>
<tr>
<td>0.05/16</td>
<td>4.15863</td>
<td>-0.00675</td>
<td>2.02</td>
</tr>
<tr>
<td>0.05/32</td>
<td>4.16199</td>
<td>-0.00389</td>
<td>1.99</td>
</tr>
<tr>
<td>0.05/64</td>
<td>4.16382</td>
<td>-0.00156</td>
<td>2.18</td>
</tr>
<tr>
<td>0.05/128</td>
<td><strong>4.16504</strong></td>
<td><strong>0.000034</strong></td>
<td><strong>4.67</strong></td>
</tr>
<tr>
<td>0.05/256</td>
<td>4.16504</td>
<td>-0.000034</td>
<td></td>
</tr>
<tr>
<td>0.05/512</td>
<td>4.16504</td>
<td>-0.000034</td>
<td></td>
</tr>
<tr>
<td>0.05/1024</td>
<td>4.16992</td>
<td>0.00454</td>
<td></td>
</tr>
<tr>
<td>0.05/2048</td>
<td>4.17969</td>
<td>0.01430</td>
<td></td>
</tr>
</tbody>
</table>

* At this point, round-off and truncation errors are in balance, but we still do not achieve six-digit accuracy.
Central-Difference Approximation (中間差分近似式) for \( f' \)

- Consider the Taylor series of \( f(x) \) at \( x = x_0 \):
  \[
  f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2} f''(x_0)(x-x_0)^2 + \cdots.
  \]

For \( x = x_0 + h \), we have

\[
 f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{1}{2} f''(x_0)h^2 + \cdots.
\]

\[
 \Rightarrow f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - \frac{1}{2} f''(x_0)h + \cdots = \frac{f(x_0 + h) - f(x_0)}{h} + O(h). \quad (1)
\]

For \( x = x_0 - h \), we have

\[
 f(x_0 - h) = f(x_0) - f'(x_0)h + \frac{1}{2} f''(x_0)h^2 + \cdots.
\]

\[
 \Rightarrow f'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h} + \frac{1}{2} f''(x_0)h - \cdots = \frac{f(x_0) - f(x_0 - h)}{h} + O(h). \quad (2)
\]

If we add Eqs. (1) and (2), then divide by 2, we get the central-difference approximation to the derivative:

\[
 f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f''(x_0)}{6}h^2 + \cdots = \frac{f(x+h) - f(x-h)}{2h} + O(h^2). \quad (central \ difference; 中間差分)
\]

The order of magnitude of error is \( O(h^2) \). It means that using a central-difference approximation is a much preferred way to estimate the derivative.

Table 5.2 shows that the errors decrease about four fold when \( \Delta x \) is halved and that a more accurate value is obtained.

<table>
<thead>
<tr>
<th>( \Delta x )</th>
<th>Approximation</th>
<th>Error</th>
<th>Ratio of errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>4.15831</td>
<td>-0.00708</td>
<td></td>
</tr>
<tr>
<td>0.05/2</td>
<td>4.16361</td>
<td>-0.00177</td>
<td>4.00</td>
</tr>
<tr>
<td>0.05/4</td>
<td>4.16496</td>
<td>-0.00042</td>
<td>4.21</td>
</tr>
<tr>
<td>0.05/8</td>
<td>4.16527</td>
<td>-0.00011</td>
<td>3.80</td>
</tr>
<tr>
<td><strong>0.05/16</strong></td>
<td><strong>4.16534</strong></td>
<td><strong>-0.00004</strong></td>
<td><strong>2.75</strong></td>
</tr>
<tr>
<td>0.05/32</td>
<td>4.16534</td>
<td>-0.00004</td>
<td></td>
</tr>
<tr>
<td>0.05/64</td>
<td>4.16565</td>
<td>-0.00027</td>
<td></td>
</tr>
</tbody>
</table>
Formulas for Computing Derivatives (Uniform spacing, \( \Delta x = h \))

- Formulas for the first derivatives:
  
  \[
  f'(x_0) = \frac{f_1 - f_0}{h} + O(h) \\
  f'(x_0) = \frac{f_1 - f_{-1}}{2h} + O(h^2) \\
  f'(x_0) = \frac{-f_2 + 4f_1 - 3f_0}{2h} + O(h^2) \\
  f'(x_0) = \frac{-f_2 + 8f_1 - 8f_{-1} + f_{-2}}{12h} + O(h^4)
  \]

  (forward difference)
  
  (central difference)
  
  (forward difference)
  
  (central difference)

- Formulas for the second derivatives:
  
  \[
  f''(x_0) = \frac{f_2 - 2f_1 + f_0}{h^2} + O(h) \\
  f''(x_0) = \frac{f_2 - 2f_0 + f_{-1}}{h^2} + O(h^2) \\
  f''(x_0) = \frac{-f_3 + 4f_2 - 5f_1 + 2f_0}{h^2} + O(h^2) \\
  f''(x_0) = \frac{-f_2 + 16f_1 - 30f_0 + 16f_{-1} - 2f_{-2}}{12h^2} + O(h^4)
  \]

  (forward difference)
  
  (central difference)
  
  (forward difference)
  
  (central difference)

- Formulas for the third derivatives:
  
  \[
  f'''(x_0) = \frac{f_3 - 3f_2 + 3f_1 - f_0}{h^3} + O(h) \\
  f'''(x_0) = \frac{f_2 - 2f_1 + 2f_{-1} - f_{-2}}{2h^3} + O(h^2)
  \]

  (forward difference)
  
  (central difference)

- Formulas for the fourth derivatives:
  
  \[
  f''''(x_0) = \frac{f_4 - 4f_3 + 6f_2 - 4f_1 + f_0}{h^4} + O(h) \\
  f''''(x_0) = \frac{f_2 - 4f_1 + 6f_0 - 4f_{-1} + f_{-2}}{h^4} + O(h^2)
  \]

  (forward difference)
  
  (central difference)

**Derivation of the Formula**  \( f'(x_0) = \frac{-f_2 + 8f_1 - 8f_{-1} + f_{-2}}{12h} + O(h^4) \)

- Consider again the Taylor series of \( f(x) \) at \( x = x_0 \):

  \[
  f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2} f''(x_0)(x-x_0)^2 + \cdots
  \]

  \[
  \Rightarrow \quad f_1 = f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{1}{2} f''(x_0)h^2 + \cdots 
  \]

  \[
  f_2 = f(x_0 + 2h) = f(x_0) + f'(x_0)(2h) + \frac{1}{2} f''(x_0)(2h)^2 + \cdots 
  \]
Extrapolation Techniques (外推技術)

• We found earlier that the errors of a central-difference approximation to \( f'(x) \) were of \( O(h^2) \). It suggests that the errors are proportional to \( h^2 \) although that is true only in the limit as \( h \to 0 \). So, from two computations with \( h \) being half, we can estimate the proportionality factor which we call \( C \). For example, in Table 5.2 we had

<table>
<thead>
<tr>
<th>( h )</th>
<th>approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>4.15831</td>
</tr>
<tr>
<td>0.025</td>
<td>4.16361</td>
</tr>
</tbody>
</table>

Then it is required

\[
\text{True value} = 4.15831 + C \cdot 0.05^2 \tag{7}
\]

\[
\text{True value} = 4.16361 + C \cdot 0.025^2 \tag{8}
\]

\[4 \cdot (8) - (7) \Rightarrow 3(\text{True value}) = 4 \cdot (4.16361 - 4.15831)
\]

\[
\text{True value} = 4.16361 + \frac{1}{3} \cdot (4.16361 - 4.15831) = 4.16538,
\]

which is very close to the exact value 4.165382.

The general formula for improving the estimate, when errors decrease by \( O(h^n) \):

\[
\text{Better estimate accurate} = \frac{1}{2^n - 1} \left( \frac{\text{more accurate}}{\text{less accurate}} \right).
\]

This extrapolation technique applies to any set of computations where the order of the error is known.

Richardson Extrapolation (Richardson 外差法)

• Consider the central-difference approximation to \( f'(x) \):

\[
\text{if } \Delta x = h \Rightarrow f'(x) = \frac{f(x + h) - f(x - h)}{2h} + a_1h^2 + a_2h^4 + a_3h^6 + \cdots \tag{9}
\]
if $\Delta x = \frac{1}{2} h \Rightarrow f'(x) = \frac{f(x + \frac{1}{2} h) - f(x - \frac{1}{2} h)}{h} + \frac{1}{2} a_2 h^2 + \frac{1}{2} a_4 h^4 + \frac{1}{2} a_6 h^6 + \cdots$ \hspace{0.5cm} (10)

if $\Delta x = \frac{1}{4} h \Rightarrow f'(x) = \frac{f(x + \frac{1}{4} h) - f(x - \frac{1}{4} h)}{\frac{1}{2} h} + \frac{1}{2} a_2 h^2 + \frac{1}{2} a_4 h^4 + \frac{1}{2} a_6 h^6 + \cdots$ \hspace{0.5cm} (11)

One can use Eqs. (9) and (10) to compute an extrapolation of $O(h^4)$; similarly, One can use Eqs. (10) and (11) to compute an extrapolation of $O(h^4)$. Then one can use these two extrapolation, by canceling the error term of $O(h^4)$, to compute another extrapolation of $O(h^6)$. Doing these successive extrapolations is called $\textbf{Richardson extrapolation}$.

Here is an example with $f(x) = x^2 \cos x$ for which $f'(1.0) = 0.23913363$. The original values of $f'(1.0)$ are from central difference so they are of $O(h^2)$.

<table>
<thead>
<tr>
<th>Value of $h$</th>
<th>$f'(1.0)$</th>
<th>First extrapolations</th>
<th>Second extrapolations</th>
<th>Third extrapolations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.226736</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.05</td>
<td>0.236031</td>
<td>0.239129</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.025</td>
<td>0.238258</td>
<td>0.239133</td>
<td>0.239134</td>
<td></td>
</tr>
<tr>
<td>0.0125</td>
<td>0.238940</td>
<td>0.239132</td>
<td>0.239132</td>
<td>0.239132</td>
</tr>
</tbody>
</table>

$\textbf{Extrapolation with Tabulated Values}$ (使用表格值之外差法)

- The function in Table 5.5 is for $f(x) = e^{-x} \sin x$ with $f'(2.4) = -0.128171$.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$x_i$</th>
<th>$f_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.0</td>
<td>0.123060</td>
</tr>
<tr>
<td>1</td>
<td>2.1</td>
<td>0.105706</td>
</tr>
<tr>
<td>2</td>
<td>2.2</td>
<td>0.089584</td>
</tr>
<tr>
<td>3</td>
<td>2.3</td>
<td>0.074764</td>
</tr>
<tr>
<td>4</td>
<td>2.4</td>
<td>0.061277</td>
</tr>
<tr>
<td>5</td>
<td>2.5</td>
<td>0.049126</td>
</tr>
<tr>
<td>6</td>
<td>2.6</td>
<td>0.038288</td>
</tr>
<tr>
<td>7</td>
<td>2.7</td>
<td>0.028722</td>
</tr>
<tr>
<td>8</td>
<td>2.8</td>
<td>0.020371</td>
</tr>
<tr>
<td>9</td>
<td>2.9</td>
<td>0.013164</td>
</tr>
<tr>
<td>10</td>
<td>3.0</td>
<td>0.007026</td>
</tr>
</tbody>
</table>

Suppose we want to compute $f'(2.4)$ with tabulated values. With use of the central-difference approximation, we can
i) compute \( f'(2.4) = -0.12819 \) from \( f(2.3) \) and \( f(2.5) \) with \( h = 0.1 \),

ii) compute \( f'(2.4) = -0.12824 \) from \( f(2.2) \) and \( f(2.6) \) with \( h = 0.2 \).

Since we know that they both are of \( O(h^2) \), an extrapolation for \( f'(2.4) \) will be

\[
\frac{1}{2}(-0.12819 - (-0.12824)) = -0.12817.
\]

Which is very close to the true value.

5.2 Numerical Integration – The Trapezoidal Rule (數值積分—梯形法)

- Given the function \( f(x) \), the antiderivative is a function \( F(x) \) such that \( F'(x) = f(x) \).

By the fundamental theorem of Calculus, the definite integral can be evaluated from the antiderivative.

\[
\int_{a}^{b} f(x) \, dx = F(b) - F(a)
\]

Still, there are functions that do not have an antiderivative expressible in terms of ordinary functions.

If we ask, “Is there any way that the definite integral can be found when the antiderivative is unknown?”, the answer is “Yes, we can do it numerically.”

You learned that the definite integral is the area between the curve of \( f(x) \) and the \( x \)-axis. That is the principle behind all numerical integration – we divide the distance from \( x = a \) to \( x = b \) into vertical strips and add the areas of these strips.

- One way to draw the strips is to make the top of the curve, either at the left corner or the right corner, but this is less accurate. We can think of these lines as interpolating polynomial of degree zero.
The Trapezoidal Rule (梯形法)

- A much better way is to approximate the curve with a sequence of straight lines. This gives us the trapezoidal rule using interpolating polynomials of degree-1. Figure 5.2 illustrates this.

It is intuitively clear that the area of the strip from \( x_i \) to \( x_{i+1} \) gives an approximation to the area under the curve:

\[
\int_{x_i}^{x_{i+1}} f(x) \, dx \approx \frac{1}{2} (f_i + f_{i+1})(x_{i+1} - x_i).
\]

If we are getting the integral of a known function from \( x = a \) to \( x = b \), we subdivide \([a, b]\) into \( n \) smaller intervals with \( \Delta x = h = \frac{1}{n}(b - a) \). Then the composite trapezoidal rule gives

\[
\int_{a}^{b} f(x) \, dx \approx \sum_{i=0}^{n-1} \frac{h}{2}(f_i + f_{i+1})h = \frac{h}{2}(f_0 + 2f_1 + 2f_2 + \cdots + 2f_{n-1} + f_n).
\]

In a computer program, one should use \( h(\frac{1}{2}f_0 + f_1 + \frac{1}{2}f_2 + \cdots + f_{n-1} + \frac{1}{2}f_n) \) in order to reduce the number of operations.

Romberg Integration

- The Romberg integration, similar to the Richardson extrapolation, can be used to improve the accuracy of the trapezoidal rule integral.

Because the integral determined with the trapezoidal method has an error of \( O(h^2) \), we can combine two estimates of the integral that have \( h \)-values in a 2:1 ratio to get an improved estimate by using
Better estimate \[= \frac{\text{more accurate}}{2^n - 1} + \frac{1}{2^n - 1} \left( \frac{\text{more accurate}}{2^{n+1} - 1} - \text{less accurate} \right). \]

Example 5.2 Use Romberg integration to find the integral of \( e^{-x^2} \) between \( x = 0.2 \) to \( x = 1.5 \). Take the initial subinterval size as \( x = \frac{1}{2} (1.5 - 0.2) \).

Our first estimate is

\[
\int_{0.2}^{1.5} e^{-x^2} \, dx \approx \frac{0.65}{2} [f(0.2) + 2f(0.85) + f(1.5)] = 0.66211.
\]

Table 5.7 exhibits the calculations when we repeat the estimates, halving the \( h \)-value each time.

<table>
<thead>
<tr>
<th>( h )</th>
<th>0.65</th>
<th>0.65947</th>
<th>0.65859</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h )</td>
<td>0.325</td>
<td>0.65898</td>
<td>0.65881</td>
</tr>
<tr>
<td>( h )</td>
<td>0.1625</td>
<td>0.65886</td>
<td>0.65882</td>
</tr>
</tbody>
</table>
5.3 Numerical Integration – Simpson’s Rules (辛普森法)

- The trapezoidal rule is based on approximating the function with a linear polynomial. We can fit the function better if we approximate it with a quadratic or a cubic interpolating polynomial. Simpson’s rules are based on these approximations.

There are two of these rules: Simpson’s 1/3 rule and Simpson’s 3/8 rule, so-named because the values 1/3 and 3/8 appear in their formulas.

Simpson’s 1/3 Rule (辛普森 1/3 法則)

- We get Simpson’s 1/3 rule by integrating the second-degree Newton-Gregory forward polynomial (see Sec. 3.2), which fits \( f(x) \) at \( x \)-values of \( x_0, x_1, x_2 \), which are evenly spaced a distance \( h \) apart:

\[
\begin{align*}
\int_{x_0}^{x_1} f(x)dx & \approx \int_{x=0}^{s} \left[ f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 \right] \cdot (hds) \quad \text{with} \quad s = \frac{x-x_0}{h} \\
& = \int_{s=0}^{s=2} \left[ hf_0 + h(f_1 - f_0)s + \frac{1}{2}h(f_2 - 2f_1 + f_0)s(s-1) \right]ds \\
& = hf_0 \cdot s\int_{s=0}^{s=2} + \frac{1}{2}h(f_1 - f_0)s^2\int_{s=0}^{s=2} + \frac{1}{2}h(f_2 - 2f_1 + f_0)(\frac{1}{2}s^3 - \frac{1}{3}s^2)\int_{s=0}^{s=2} \\
& = h[2f_0 + 2(f_1 - f_0) + \frac{1}{2}(f_2 - 2f_1 + f_0)] \\
& = \frac{1}{2}h[6f_0 + 6(f_1 - f_0) + (f_2 - 2f_1 + f_0)] \\
& = \frac{1}{2}h(f_0 + 4f_1 + f_2)
\end{align*}
\]

Simpson’s 3/8 Rule (辛普森 3/8 法則)

- Similarly, we get Simpson’s 3/8 rule by integrating the third-degree Newton-Gregory forward polynomial, which fits \( f(x) \) at \( x \)-values of \( x_0, x_1, x_2, x_3 \), which are evenly spaced a distance \( h \) apart:

\[
\begin{align*}
\int_{x_0}^{x_1} f(x)dx & \approx \int_{x=0}^{s} \left[ f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!}\Delta^3 f_0 \right] \cdot (hds) \quad \text{with} \quad s = \frac{x-x_0}{h} \\
& = \frac{1}{8}h(f_0 + 3f_1 + 3f_2 + f_3)
\end{align*}
\]
Formulas for Integration (Uniform spacing, $\Delta x = h$)

- Trapezoidal rule:
  \[
  \int_{a}^{b} f(x) \, dx = \frac{h}{2} \left( f_1 + 2f_2 + 2f_3 + \ldots + 2f_n + f_{n+1} \right)
  \]
  \[- \frac{b-a}{12} h^2 f^{(n)}(\xi), \quad a \leq \xi \leq b.
  \]

Simpson’s $\frac{1}{4}$-rule: (requires an even number of panels)

\[
\int_{a}^{b} f(x) \, dx = \frac{h}{3} \left( f_1 + 4f_2 + 2f_3 + 4f_4 + 2f_5 + \ldots + 4f_n + f_{n+1} \right)
\]
\[- \frac{b-a}{180} h^4 f^{(4)}(\xi), \quad a \leq \xi \leq b.
\]

Simpson’s $\frac{3}{8}$-rule: (requires a number of panels divisible by 3)

\[
\int_{a}^{b} f(x) \, dx = \frac{3h}{8} \left( f_1 + 3f_2 + 3f_3 + 2f_4 + 3f_5 + \ldots + 3f_n + f_{n+1} \right)
\]
\[- \frac{b-a}{80} h^4 f^{(4)}(\xi), \quad a \leq \xi \leq b.
\]

It is convenient to think of the strips defined by successive $x$-values as panels (板塊). For Simpson’s $1/3$ rule, there must be an even number of panels; while for Simpson’s $3/8$ rule, the number of panels should be of 3 times.

You may wonder why we use the $3/8$ rule when it has a larger error. One useful application of it is to find the integral from a table of values that has an odd number of panels.

**Example** Find the integral of $e^{-x^2}$ between $x = 0.2$ to $x = 2.6$.

**Table 5.9** Comparison of integration methods for the integral of $\exp(-x^2)$ between $x = 0.2$ and 2.6

<table>
<thead>
<tr>
<th>Number of panels</th>
<th>Trapezoidal rule</th>
<th>Simpson’s $1/3$ rule</th>
<th>Simpson’s $3/8$ rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value</td>
<td>Error</td>
<td>Value</td>
</tr>
<tr>
<td>6</td>
<td>0.69378</td>
<td>-0.00513</td>
<td>0.68824</td>
</tr>
<tr>
<td>12</td>
<td>0.68992</td>
<td>-0.00127</td>
<td>0.68863</td>
</tr>
<tr>
<td>18</td>
<td>0.68921</td>
<td>-0.00056</td>
<td>0.68865</td>
</tr>
<tr>
<td>24</td>
<td>0.68897</td>
<td>-0.00031</td>
<td>0.68865</td>
</tr>
</tbody>
</table>
5.4 An Application of Numerical Integration – Fourier Series and Fourier Transforms

• In the next example, we compare the accuracy of computing Fourier coefficients by the trapezoidal rule and by Simpson’s 1/3 rule in a case where the analytic values are possible.

Example 5.5

Evaluate the coefficients for the half-range expansions for \( f(x) = x \) on \([0, 2]\) numerically and compare to the analytic values. Do this with both 20 intervals and 200 intervals.

For the even extension (the Fourier cosine series), we use Eq. (4.55) to get the \( A \)'s (all \( B \)'s are zero):

\[
A_n = \left( \frac{2}{2} \right) \int_0^2 x \cos \left( \frac{n \pi x}{2} \right) \, dx, \quad n = 0, 1, 2, \ldots
\]

For the even extension (the Fourier sine series), we use Eq. (4.56) to get the \( B \)'s (all \( A \)'s are zero):

\[
B_n = \left( \frac{2}{2} \right) \int_0^2 x \sin \left( \frac{n \pi x}{2} \right) \, dx, \quad n = 1, 2, 3, \ldots
\]

Tables 5.10 and 5.11 show the results. Observe that the accuracy is poorer as the value of \( n \) increases.

Table 5.10 Comparison of numerical integration with analytical results: 20 subdivisions of \([0, 2]\)

<table>
<thead>
<tr>
<th>( n )</th>
<th>( A_n )</th>
<th>( B_n )</th>
<th>( A_n )</th>
<th>( B_n )</th>
<th>( A_n )</th>
<th>( B_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td></td>
<td>2</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-0.81224</td>
<td>1.27062</td>
<td>-0.81056</td>
<td>1.27324</td>
<td>-0.81057</td>
<td>1.27323</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-0.63138</td>
<td>0</td>
<td>-0.63665</td>
<td>0</td>
<td>-0.63662</td>
</tr>
<tr>
<td>3</td>
<td>-0.09175</td>
<td>0.41653</td>
<td>-0.08999</td>
<td>0.42453</td>
<td>-0.09006</td>
<td>0.42441</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>-0.30777</td>
<td>0</td>
<td>-0.31860</td>
<td>0</td>
<td>-0.31831</td>
</tr>
<tr>
<td>5</td>
<td>-0.03414</td>
<td>0.24142</td>
<td>-0.03219</td>
<td>0.25523</td>
<td>-0.03242</td>
<td>0.25465</td>
</tr>
</tbody>
</table>

Table 5.11 Comparison of numerical integration with analytical results: 200 subdivisions of \([0, 2]\)

<table>
<thead>
<tr>
<th>( n )</th>
<th>( A_n )</th>
<th>( B_n )</th>
<th>( A_n )</th>
<th>( B_n )</th>
<th>( A_n )</th>
<th>( B_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td></td>
<td>2</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-0.81059</td>
<td>1.27321</td>
<td>-0.81057</td>
<td>1.27324</td>
<td>-0.81057</td>
<td>1.27323</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-0.63657</td>
<td>0</td>
<td>-0.63662</td>
<td>0</td>
<td>-0.63662</td>
</tr>
<tr>
<td>3</td>
<td>-0.09008</td>
<td>0.42433</td>
<td>-0.09006</td>
<td>0.42441</td>
<td>-0.09006</td>
<td>0.42441</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>-0.31821</td>
<td>0</td>
<td>-0.31831</td>
<td>0</td>
<td>-0.31831</td>
</tr>
<tr>
<td>5</td>
<td>-0.03244</td>
<td>0.25452</td>
<td>-0.03242</td>
<td>0.25465</td>
<td>-0.03242</td>
<td>0.25465</td>
</tr>
</tbody>
</table>
Example 5.6

An experiment (actually, these are contrived data) showed the displacements given in Table 5.12 when the system was caused to vibrate in its natural modes. The values represent a periodic function on the interval for \( t \) of \([2, 10]\) because they repeat themselves after \( t = 10 \).

<table>
<thead>
<tr>
<th>( t )</th>
<th>Displacement</th>
<th>( t )</th>
<th>Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.000</td>
<td>3.804</td>
<td>6.250</td>
<td>3.746</td>
</tr>
<tr>
<td>2.250</td>
<td>6.503</td>
<td>6.500</td>
<td>5.115</td>
</tr>
<tr>
<td>2.500</td>
<td>7.496</td>
<td>6.750</td>
<td>4.156</td>
</tr>
<tr>
<td>2.750</td>
<td>6.094</td>
<td>7.000</td>
<td>1.593</td>
</tr>
<tr>
<td>3.000</td>
<td>3.003</td>
<td>7.250</td>
<td>-0.941</td>
</tr>
<tr>
<td>3.250</td>
<td>-0.105</td>
<td>7.500</td>
<td>-1.821</td>
</tr>
<tr>
<td>3.500</td>
<td>-1.598</td>
<td>7.750</td>
<td>-0.329</td>
</tr>
<tr>
<td>3.750</td>
<td>-0.721</td>
<td>8.000</td>
<td>2.799</td>
</tr>
<tr>
<td>4.000</td>
<td>1.806</td>
<td>8.250</td>
<td>5.907</td>
</tr>
<tr>
<td>4.250</td>
<td>4.350</td>
<td>8.500</td>
<td>7.338</td>
</tr>
<tr>
<td>4.500</td>
<td>5.255</td>
<td>8.750</td>
<td>6.380</td>
</tr>
<tr>
<td>4.750</td>
<td>3.878</td>
<td>9.000</td>
<td>3.709</td>
</tr>
<tr>
<td>5.000</td>
<td>0.893</td>
<td>9.250</td>
<td>0.992</td>
</tr>
<tr>
<td>5.250</td>
<td>-2.048</td>
<td>9.500</td>
<td>-0.116</td>
</tr>
<tr>
<td>5.500</td>
<td>-3.280</td>
<td>9.750</td>
<td>1.047</td>
</tr>
<tr>
<td>5.750</td>
<td>-2.088</td>
<td>10.000</td>
<td>3.802</td>
</tr>
<tr>
<td>6.000</td>
<td>0.807</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We will use trapezoidal integration to find the Fourier series coefficients for the data. Doing so gives these values for the \( A \)'s and \( B \)'s:

<table>
<thead>
<tr>
<th>( n )</th>
<th>( A )</th>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.6015</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.5004</td>
<td>-0.5006</td>
</tr>
<tr>
<td>2</td>
<td>-0.0009</td>
<td>0.0016</td>
</tr>
<tr>
<td>3</td>
<td>-0.0017</td>
<td>0.0016</td>
</tr>
<tr>
<td>4</td>
<td>0.0008</td>
<td>4.0011</td>
</tr>
<tr>
<td>5</td>
<td>-0.0017</td>
<td>0.0000</td>
</tr>
<tr>
<td>6</td>
<td>-0.0009</td>
<td>0.0022</td>
</tr>
<tr>
<td>7</td>
<td>-0.0005</td>
<td>-0.0023</td>
</tr>
<tr>
<td>8</td>
<td>-0.0008</td>
<td>0.0009</td>
</tr>
</tbody>
</table>

This shows that only \( A_0, A_1, B_1, \) and \( B_4 \) are important. There would be no amplification of motion from forces that do not include the frequencies corresponding to these. (Table 5.12 was constructed from

\[
f(t) = 2.3 + 1.5 \cos(t) - 0.5 \sin(t) + 4 \sin(4t),
\]

plus a small random variation whose values ranged from \(-0.01\) to \(+0.01\). It is the random variations that cause nonzero values for the insignificant \( A \)'s and \( B \)'s.)
Example 5.9

Use the FFT algorithm to obtain the finite Fourier series coefficients for the same data as in Table 5.12. These are perturbed values from

\[ f(t) = 2.3 + 1.5 \cos(t) - 0.5 \sin(t) + 4 \sin(4t). \]

A computer program that implements the algorithm gave these results:

<table>
<thead>
<tr>
<th>n</th>
<th>( A_n )</th>
<th>( B_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.6017</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.4993</td>
<td>-0.4994</td>
</tr>
<tr>
<td>2</td>
<td>0.0017</td>
<td>-0.0010</td>
</tr>
<tr>
<td>3</td>
<td>0.0003</td>
<td>-0.0005</td>
</tr>
<tr>
<td>4</td>
<td>0.0015</td>
<td>3.9990</td>
</tr>
<tr>
<td>5</td>
<td>0.0019</td>
<td>0.0009</td>
</tr>
<tr>
<td>6</td>
<td>-0.0004</td>
<td>-0.0009</td>
</tr>
<tr>
<td>7</td>
<td>-0.0003</td>
<td>-0.0019</td>
</tr>
<tr>
<td>8</td>
<td>0.0017</td>
<td>-0.0008</td>
</tr>
<tr>
<td>9</td>
<td>-0.0023</td>
<td>0.0019</td>
</tr>
<tr>
<td>10</td>
<td>-0.0024</td>
<td>-0.0011</td>
</tr>
<tr>
<td>11</td>
<td>0.0003</td>
<td>0.0020</td>
</tr>
<tr>
<td>12</td>
<td>0.0008</td>
<td>-0.0033</td>
</tr>
<tr>
<td>13</td>
<td>-0.0004</td>
<td>0.0011</td>
</tr>
<tr>
<td>14</td>
<td>0.0025</td>
<td>0.0003</td>
</tr>
<tr>
<td>15</td>
<td>-0.0005</td>
<td>0.0013</td>
</tr>
<tr>
<td>16</td>
<td>-0.0010</td>
<td></td>
</tr>
</tbody>
</table>

The results are essentially the same as those of Example 5.6, which were computed by the trapezoidal rule.

**N-Point Discrete Fourier Transform**

- Remember that the complex Fourier series of a function \( f(x) \) is

\[
\sum_{n=-\infty}^{\infty} d_n e^{i(n\pi x/L)},
\]

in which \( d_n = \frac{1}{2L} \int_{-L}^{L} f(x)e^{-i(n\pi x/L)} dx \).

Let \( N \) be a positive integer and \( f = \{f_0, f_1, f_2, \ldots, f_{N-1}\} \) be a sequence of \( N \) complex numbers. Then the **N-point discrete Fourier transform** of \( f \) is the sequence \( d = \{d_0, d_1, d_2, \ldots, d_{N-1}\} \) defined by

\[
d_n = \sum_{j=0}^{N-1} f_j e^{-i(n(2\pi j)/N)}, \text{ for } n = 0, 1, 2, \ldots, N-1.
\]
The inverse N-point discrete Fourier transform of \( d \) is

\[
f_k = \frac{1}{N} \sum_{n=0}^{N-1} d_n e^{i(n\pi k)/N}, \quad \text{for } k = 0, 1, 2, \ldots, N-1.
\]

The Fast Fourier Transform (FFT/快速傅立葉轉換)

- The FFT is not a transform at all, but rather an efficient procedure for computing discrete Fourier transforms (DFTs).

The FFT is known to many users as a computer routine that they simply key into their analysis.

The problem addressed by the FFT is that of computing the \( N \)-point DFT

\[
d_n = \sum_{j=0}^{N-1} f_j e^{-i(n\pi j)/N} \quad \text{for a given sequence } f = \{f_0, f_1, f_2, \ldots, f_{N-1}\}.
\]

In applying the FFT, \( N \) is chosen as a positive integer power of 2.

The idea behind the FFT becomes clear if we examine in detail a simple case, namely, \( N = 2^2 = 4 \). In this case, the quantities to be computed in

\[
d_n = \sum_{j=0}^{N-1} f_j e^{-i(n\pi j)/N}
\]

are

\[
d_0 = f_0 + f_1 + f_2 + f_3, \\
d_1 = f_0 + f_1 \cdot e^{-i(2\pi/4)} + f_2 \cdot e^{-i(4\pi/4)} + f_3 \cdot e^{-i(6\pi/4)}, \\
d_2 = f_0 + f_1 \cdot e^{-i(4\pi/4)} + f_2 \cdot e^{-i(8\pi/4)} + f_3 \cdot e^{-i(12\pi/4)}, \\
d_3 = f_0 + f_1 \cdot e^{-i(6\pi/4)} + f_2 \cdot e^{-i(12\pi/4)} + f_3 \cdot e^{-i(18\pi/4)}.
\]

Let \( W = e^{-i(2\pi/4)} \). And notice that \( W^k = W^{\text{mod}(k,4)} \). We therefore have

\[
d_0 = f_0 + f_1 + f_2 + f_3, \\
d_1 = f_0 + f_1 \cdot W + f_2 \cdot W^2 + f_3 \cdot W^3, \\
d_2 = f_0 + f_1 \cdot W^2 + f_2 \cdot W^4 + f_3 \cdot W^6 = f_0 + f_1 \cdot W^2 + f_2 \cdot W^0 + f_3 \cdot W^2, \\
d_3 = f_0 + f_1 \cdot W^3 + f_2 \cdot W^6 + f_3 \cdot W^9 = f_0 + f_1 \cdot W^3 + f_2 \cdot W^2 + f_3 \cdot W^1.
\]

Write this system of equations in matrix form:
\[
\begin{bmatrix}
    d_0 \\
    d_1 \\
    d_2 \\
    d_3
\end{bmatrix}
= \begin{bmatrix}
    1 & 1 & 1 & 1 \\
    1 & W & W^2 & W^3 \\
    1 & W^2 & W^0 & W^2 \\
    1 & W^3 & W^2 & W
\end{bmatrix}
\begin{bmatrix}
    f_0 \\
    f_1 \\
    f_2 \\
    f_3
\end{bmatrix}
\]

A straightforward count shows that \( N^2 = 16 \) complex multiplications and \( N(N-1) = 12 \) complex additions are needed to compute this matrix product.

The crucial step is to recognize that the last matrix equation can be written in factored form as

\[
\begin{bmatrix}
    d_0 \\
    d_2 \\
    d_1 \\
    d_3
\end{bmatrix}
= \begin{bmatrix}
    1 & W^0 & 0 & 0 \\
    1 & W^2 & 0 & 0 \\
    0 & 0 & 1 & W \\
    0 & 0 & 1 & W^3
\end{bmatrix}
\begin{bmatrix}
    1 & 0 & W^0 & 0 \\
    0 & 1 & 0 & W^0 \\
    1 & 0 & W^2 & 0 \\
    0 & 1 & 0 & W^2
\end{bmatrix}
\begin{bmatrix}
    f_0 \\
    f_1 \\
    f_2 \\
    f_3
\end{bmatrix}
\]

This factorization is not obvious but, once given, is easily verified. A price to be paid for the factorization is that \( d_2 \) and \( d_1 \) have become switched.

Let

\[
\begin{bmatrix}
    d_0 \\
    d_2 \\
    d_1 \\
    d_3
\end{bmatrix}
= \begin{bmatrix}
    1 & W^0 & 0 & 0 \\
    1 & W^2 & 0 & 0 \\
    0 & 0 & 1 & W \\
    0 & 0 & 1 & W^3
\end{bmatrix}
\begin{bmatrix}
    b_0 \\
    b_1 \\
    b_2 \\
    b_3
\end{bmatrix}
\quad \text{and} \quad
\begin{bmatrix}
    b_0 \\
    b_1 \\
    b_2 \\
    b_3
\end{bmatrix}
= \begin{bmatrix}
    1 & 0 & W^0 & 0 \\
    0 & 1 & 0 & W^0 \\
    1 & 0 & W^2 & 0 \\
    0 & 1 & 0 & W^2
\end{bmatrix}
\begin{bmatrix}
    f_0 \\
    f_1 \\
    f_2 \\
    f_3
\end{bmatrix}
\]

In total, computation of the factored product costs 4 complex multiplications and 8 complex additions.

The FFT method reduces the number of complex computations from a number proportional to \( N^2 \) to a number proportional to \( N \log_2(N) \). For \( N = 4 \), this is a reduction by one-half; if \( N = 1024 \), the reduction in complex multiplies is a hundredfold.

It is convenient to represent the sequence of multiplications of the factored form by flow diagrams. Figure 5.4 is for \( N = 4 \) and Figure 5.5 is for \( N = 16 \).

The number of stages is \( \log_2(N) \). In each stage, we get \( x \)-values of the next stage from those of the present stage. Every new \( x \)-value is the sum of the two \( x \)-values from the
previous stage that connect to it, with one of those multiplies by a power of $W$.

The last columns in Figures 5.4 and 5.5 indicate how the final $x$-values ($x_2$ and $x_4$) are unscrambled to give the $x$-values of the last columns. The relationship can be found by expressing the index of the final $x$-values as a binary number and reversing the bits. For example, in Figure 5.5, we have

$$3 = 0011_2 \rightarrow 1100_2 = 12 \quad \text{and} \quad 11 = 1011_2 \rightarrow 1101_2 = 13.$$
An Algorithm to Perform a Fast Fourier Transform

Given \( n \) data points, \((x_i, f_i), i = 0, \ldots, n - 1\) (with \( n \) a power of 2) and \( x \) on \([0 \ldots 2\pi]\):

Set \( y_r_i = f_i, i = 0, \ldots, n - 1 \).

Set \( y_i_i = 0, i = 0, \ldots, n - 1 \).

Set \( c_i = \cos(2i\pi/n), i = 0, \ldots, n - 1 \).

Set \( s_i = \sin(2i\pi/n), i = 0, \ldots, n - 1 \).

Set \( \text{numstages} = \log_2(n) \).

Set \( p_i = 0, i = 0, \ldots, n - 1 \).

For stage = 1 To numstages Do

\[ \text{Set } p_{i+2} = p_i + 1, i = 0, \ldots, 2^{\text{stage-1}} \]

End Do (For stage)

Set stage = 1.

Set nsets = 1.

Set del = \( n/2 \).

Set \( k = 0 \).

Repeat

For set = 1 To nsets - 1 Do

For \( i = 0 \) To \( n/nsets - 1 \) Do

\[ \text{Set } j = i \text{ Mod del} + (\text{set} - 1) * \text{del} * 2 \]

\[ \text{Set } \ell = \text{Int}(\ell/\text{del}) \]

\[ \text{Set } y_{yj} = y_{jr} + c_{\ell} * y_{rj+\text{del}} - s_{\ell} * y_{ij+\text{del}} \]

\[ \text{Set } y_{yi} = y_{ji} + c_{\ell} * y_{ij+\text{del}} - s_{\ell} * y_{jr+\text{del}} \]

\[ \text{Set } k = k + 1 \]

End Do (For \( i \)).

End Do (For set).

Set \( y_r_i = y_{yj}, i = 0, \ldots, n - 1 \).

Set \( y_i_i = y_{yji}, i = 0, \ldots, n - 1 \).

Set stage = stage + 1.

Set nsets = nsets * 2.

Set del = del/2.

Set \( k = 0 \).

Until stage > numstages.

When terminated, the \( A \)'s and \( B \)'s of the Fourier series are contained in the \( y_r \) and \( y_i \) arrays. These must be divided by \( n/2 \) and should be unscrambled using the \( p \)-array values as indices.

Note: If the \( f_i \) are complex numbers, set the imaginary parts into array \( y_i \).
Homework #7

Use the FFT algorithm to obtain the finite Fourier series coefficients from 32 data points evaluated from

\[ f(x) = 2.3 + 1.5\cos(t) - 0.5\sin(t) + 4\sin(4t). \]

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NFFT 32
#define SWAP(a,b) temp=(a);(a)=(b);(b)=temp

int main()
{
    int i;
    float PI, dtheta, theta, c1;
    float func[NFFT+1];
    FILE *fPtr;
    PI = 4.0 * atan(1.0);

    // data generation
    dtheta = 2.0 * PI / NFFT;
    theta = 0.0;
    for ( i=1;i<=NFFT;i++ )
    {
        func[i] = 2.3 + 1.5*cos(theta) - 0.5*sin(theta) + 4.0*sin(4.0*theta);
        theta += dtheta;
    }

    // fast Fourier transform
    realft(func, NFFT, 1);
    c1 = dtheta / PI;
    func[1] = 0.5*c1*func[1];
    for ( i=2;i<=NFFT;i++ )
    {
        func[i] = c1*func[i];
    }

    // output
    fPtr = fopen( "output.dat" , "w" );
    for ( i=1;i<NFFT;i+=2 )
    {
        fprintf( fPtr , "%25.15f %25.15f\n" , func[i] , func[i+1] );
    }
    fclose( fPtr );

    system("PAUSE");
    return 0;
}

void realft(float data[], unsigned long n, int isign)
{
    void four1(float data[], unsigned long nn, int isign);
    unsigned long i,i1,i2,i3,i4,np3;
    float c1=0.5,c2,h1r,h1i,h2r,h2i;
    double wr,wi,wpr,wpi,wtemp,theta;

    theta=3.141592653589793/(double) (n>>1);
    if (isign == 1) {
        c2 = -0.5;
        four1(data,n>>1,1);
    } else {
        c2 = 0.5;
        four1(data,n>>1,0);
    }
}
```
\begin{verbatim}
} else {
  c2=0.5;
  theta = -theta;
}
  wtemp=sin(0.5*theta);
  wpr = -2.0*wtemp*wtemp;
  wpi=sin(theta);
  wr=1.0+wpr;
  wi=wpi;
  np3=n+3;
  for (i=2;i<=(n>>2);i++) {
    i4=1+(i3=np3-(i2=1+(i1=i+i-1)));
    h1r=c1*(data[i1]+data[i3]);
    h1i=c1*(data[i2]-data[i4]);
    h2r = -c2*(data[i2]+data[i4]);
    h2i=c2*(data[i1]-data[i3]);
    data[i1]=h1r+wr*h2r-wi*h2i;
    data[i2]=h1i+wr*h2i+wi*h2r;
    data[i3]=h1r-wr*h2r+wi*h2i;
    data[i4] = -h1i+wr*h2i+wi*h2r;
    wr=(wtemp=wr)*wpr-wi*wpi+wr;
    wi=wi*wpr+wtemp*wpi+wi;
  }
  if (isign == 1) {
    data[1] = (h1r=data[1])+data[2];
    data[2] = h1r-data[2];
  } else {
    data[1]=c1*(h1r=data[1])+data[2];
    data[2]=c1*(h1r-data[2]);
    four1(data,n>>1,-1);
  }
}

void four1(float data[], unsigned long nn, int isign) {
  unsigned long n,mmax,m,j,istep,i;
  double wtemp,wr,wpr,wpi,wi,theta;
  float tempr,tempi;

  n=nn << 1;
  j=1;
  for (i=1;i<n;i+=2) {
    if (j > i) {
      SWAP(data[j],data[i]);
      SWAP(data[j+1],data[i+1]);
    }
    m=nn;
    while (m >= 2 && j > m) {
      j -= m;
      m >>= 1;
    }
    j += m;
  }
  mmax=2;
  while (n > mmax) {
    istep=mmax << 1;
    theta=isign*(6.28318530717959/mmax);
    wtemp=sin(0.5*theta);
    wpr = -2.0*wtemp*wtemp;
    wpi=sin(theta);
    wr=1.0;
    wi=0.0;
    for (m=1;m<mmax;m+=2) {
      for (i=m;i<n;i+=istep) {
...
\end{verbatim}
\[
\begin{align*}
&j = i + \text{mmax;} \\
&\text{tempr} = \text{wr} \cdot \text{data}[j] - \text{wi} \cdot \text{data}[j+1]; \\
&\text{tempi} = \text{wr} \cdot \text{data}[j+1] + \text{wi} \cdot \text{data}[j]; \\
&\text{data}[j] = \text{data}[i] - \text{tempr}; \\
&\text{data}[j+1] = \text{data}[i+1] - \text{tempi}; \\
&\text{data}[i] += \text{tempr}; \\
&\text{data}[i+1] += \text{tempi}; \\
&\text{wr} = (\text{wtemp} = \text{wr}) \cdot \text{wpr} - \text{wi} \cdot \text{wpi} + \text{wr}; \\
&\text{wi} = \text{wi} \cdot \text{wpr} + \text{wtemp} \cdot \text{wpi} + \text{wi}; \\
&\text{mmax} = \text{istep};
\end{align*}
\]

```c
#define SWAP

```

```
  2.299999952316284  -0.000000953674316
  1.49999165534973  -0.500000178813934
 -0.000000735331014   0.000001086131419
  0.000001032933028   0.000004514623015
  0.000006976951681   3.99999523162842
  0.000000972873295  -0.000005252232768
 -0.000000441642868  -0.000001880910418
 -0.000000898570420  -0.000001435938771
 -0.000000715255737  -0.000001192092896
 -0.000000725656150  -0.000000884595750
 -0.000000679998493  -0.000000709890912
 -0.000000766032997  -0.000000491004471
 -0.000000778068511  -0.000000371403331
 -0.000000777837215  -0.000000260891255
 -0.000000765631967  -0.000000127034923
 -0.000000813761233  -0.000000097154079
```

- 115 -
5.6 Gaussian Quadrature (高斯求面积法)

- Gauss observed that if we remove the requirement that the function can be evaluated at some predetermined \( x \)-values, a \( n \)-term formula will contain \( 2n \) parameters (\( n \) unknown \( x \)-values plus \( n \) weights) and should correspond to an interpolating polynomial of degree-(\( 2n-1 \)). Formulas based on this principle are called **Gaussian Quadrature formulas**. They can be applied only when \( f(x) \) is known explicitly, so that it can be evaluated at any desired value of \( x \).

For a two-term formula containing four unknown parameters (包含四个未知数的二项式公式),

\[
\int_{-1}^{1} f(t)dt = af(t_1) + bf(t_2).
\]

We use an integration interval that is symmetrical about the origin, from \( t = -1 \) to \( t = 1 \) to simplify the arithmetic. It is requested that the two-term formula is to be valid for any polynomial of degree-3, that is,

- when \( f(t) = t^3 \), \( at_1^3 + bt_2^3 = \int_{-1}^{1} t^3 dt = 0 \);
- when \( f(t) = t^2 \), \( at_1^2 + bt_2^2 = \int_{-1}^{1} t^2 dt = \frac{2}{3} \);
- when \( f(t) = t \), \( at_1 + bt_2 = \int_{-1}^{1} t dt = 0 \);
- when \( f(t) = 1 \), \( a + b = \int_{-1}^{1} 1 dt = 2 \).

Multiplying the third equation by \( t_1^2 \) and subtracting from the first to eliminate terms containing \( t_1 \), we have

\[
b(t_2)(t_2 - t_1)(t_2 + t_1) = 0.
\]

Only the condition \( t_2 + t_1 = 0 \) is valid because the other possibilities reduce our formula to only a single term. We then find that

\[
a = b = 1 \quad \text{and} \quad t_1 = -t_2 = -\sqrt{\frac{2}{3}} = -0.5773.
\]

It means that

\[
\int_{-1}^{1} f(t)dt = f(-0.5773) + f(0.5773)
\]
for any cubic polynomial function $f(t)$.

In case our limits of integration are from $t = a$ to $t = b$, we can replace the given variable by another according to the following scheme:

$$x = \frac{(b-a)t+b+a}{2}$$

so that $dx = \left(\frac{b-a}{2}\right)dt$.

Then

$$\int_a^b f(t) dt = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{(b-a)t+b+a}{2}\right) dt.$$ 

Figure 5.13 lists values that we need for Gaussian Quadrature where the equivalent polynomial is up to degree-9.

<p>| Table 5.13 | Values for Gaussian quadrature |
|-----------------------------|</p>
<table>
<thead>
<tr>
<th>Number of terms</th>
<th>Values of $t$</th>
<th>Weighting factor</th>
<th>Valid up to degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>-0.57735027</td>
<td>1.0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0.57735027</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-0.77459667</td>
<td>0.55555555</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.88888889</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.77459667</td>
<td>0.55555555</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-0.86113631</td>
<td>0.34785485</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>-0.33998104</td>
<td>0.65214515</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.33998104</td>
<td>0.65214515</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.86113631</td>
<td>0.34785485</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-0.90617975</td>
<td>0.23692689</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>-0.53846931</td>
<td>0.47862867</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.56888889</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.53846931</td>
<td>0.47862867</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.90617975</td>
<td>0.23692689</td>
<td></td>
</tr>
</tbody>
</table>

Note that the $t_i$'s for a given number of terms $n$ are the zero roots of the $n$th-degree Legendre polynomial.

**Example 5.11**

Evaluate $I = \int_0^\pi \sin x\; dx$. (It is not hard to show that $I = 1.0$, so we can readily see the error of our estimate.)

To use the two-term Gaussian formula, we must change the variable of integration to make the limits of integration from $-1$ to $1$. 

- 117 -
Let

\[ x = \frac{(\pi/2)t + \pi/2}{2}, \quad \text{so} \quad dx = \frac{\pi}{4} dt. \]

Observe that when \( t = -1, x = 0 \); when \( t = 1, x = \pi/2 \). Then

\[ I = \frac{\pi}{4} \int_{-1}^{1} \sin \left( \frac{\pi t + \pi}{4} \right) dt. \]

The Gaussian formula calculates the value of the new integral as a weighted sum of two values of the integrand, at \( t = -0.5773 \) and \( t = 0.5773 \). Hence,

\[ I = \frac{\pi}{4} \left[ (1.0)(\sin(0.10566\pi)) + (1.0)(\sin(0.39434\pi)) \right] \]

\[ = 0.99847. \]

The error is \( 1.53 \times 10^{-3} \).

The power of the Gaussian method derives from the fact that we need only two functional evaluations. If we had used the trapezoidal rule, which also requires only two evaluations, our estimate would have been \( (\pi/4)(0.0 + 1.0) = 0.7854 \), an answer quite far from the mark. Simpson’s \( \frac{1}{3} \) rule requires three functional evaluations and gives \( I = 1.0023 \), with an error of \( -2.3 \times 10^{-3} \), somewhat greater than for Gaussian quadrature.

\[ \text{Example 5.12} \]

Repeat Example 5.4, but use the four-term Gaussian formula. Compare to the result of Example 5.4. We are to evaluate

\[ I = \int_{0.2}^{2.6} e^{-x} \, dx. \]

We change to variable \( t \) for limits \([-1, 1] \):

\[ x = \frac{(2.6 - 0.2)t + 2.6 + 0.2}{2} = 1.2t + 1.4. \]

So that

\[ I = \frac{2.6 - 0.2}{2} \int_{-1}^{1} e^{-(1.2t+1.4)} \, dt \]

\[ = 1.2[0.3478 \ldots e^{[-1.2(-0.8611 \ldots)+1.4]} + 0.6521 \ldots e^{[-1.2(-0.3398 \ldots)+1.4]} + 0.6521 \ldots e^{[-1.2(0.3398 \ldots)+1.4]} + 0.3478 \ldots e^{[-1.2(0.8611 \ldots)+1.4]}] \]

\[ = 0.68833, \text{ whose error is } -0.00032. \]

This error is less than the error from Simpson’s \( 1/3 \) Rule with six intervals (its error is \(-0.00041\)) and less than the error with the trapezoidal rule with 18 intervals (its error is \(-0.00056\)).
5.7 Multiple Integrals (多重積分)

- Recall that any one of the integration formulas is just a linear combination of values of the function, evaluated at varying values of the independent variable. For a multiple integral, we can easily adapt the previous integration formulas in each direction.

There is no reason to use the same integration formula in each direction.

Example 5.13
Evaluate the integral of the function \( f(x, y) \), tabulated in Table 5.14, over a rectangular region bounded by \( 1.5 \leq x \leq 3.0, \ 0.2 \leq y \leq 0.6 \).

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( 0.1 )</th>
<th>( 0.2 )</th>
<th>( 0.3 )</th>
<th>( 0.4 )</th>
<th>( 0.5 )</th>
<th>( 0.6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td></td>
<td>0.165</td>
<td>0.428</td>
<td>0.687</td>
<td>0.942</td>
<td>1.190</td>
<td>1.643</td>
</tr>
<tr>
<td>1.0</td>
<td></td>
<td>0.271</td>
<td>0.640</td>
<td>1.003</td>
<td>1.359</td>
<td>1.703</td>
<td>2.035</td>
</tr>
<tr>
<td>1.5</td>
<td></td>
<td>0.447</td>
<td>0.990</td>
<td>1.524</td>
<td>2.045</td>
<td>2.549</td>
<td>3.031</td>
</tr>
<tr>
<td>2.0</td>
<td></td>
<td>0.738</td>
<td>1.568</td>
<td>2.384</td>
<td>3.177</td>
<td>3.943</td>
<td>4.672</td>
</tr>
<tr>
<td>2.5</td>
<td></td>
<td>1.216</td>
<td>2.520</td>
<td>3.800</td>
<td>5.044</td>
<td>6.241</td>
<td>7.379</td>
</tr>
<tr>
<td>3.0</td>
<td></td>
<td>2.005</td>
<td>4.090</td>
<td>6.136</td>
<td>8.122</td>
<td>10.030</td>
<td>11.841</td>
</tr>
</tbody>
</table>

Let us use the trapezoidal rule in the \( x \)-direction and Simpson’s \( \frac{1}{3} \) rule in the \( y \)-direction. (Because the number of panels in the \( x \)-direction is not even, Simpson’s \( \frac{1}{3} \) rule does not apply readily.) It is immaterial which integral we evaluate first. Suppose we start with \( y \) being constant:

\[
y = 0.2: \quad \int_{1.5}^{3.0} f(x, 0.2) \, dx = \frac{h}{2} (f_1 + 2f_2 + 2f_3 + f_4)
\]
\[
= \frac{0.5}{2} \left[ 0.990 + 2(1.568) + 2(2.520) + 4.090 \right]
\]
\[
= 3.3140;
\]

\[
y = 0.3: \quad \int_{1.5}^{3.0} f(x, 0.3) \, dx = \frac{0.5}{2} \left[ 1.524 + 2(2.384) + 2(3.800) + 6.136 \right]
\]
\[
= 5.0070.
\]

Similarly, at

\[
y = 0.4, \quad I = 6.6522;
\]
\[
y = 0.5, \quad I = 8.2368;
\]
\[
y = 0.6, \quad I = 9.7435.
\]

We now sum these in the \( y \)-direction according to Simpson’s rule:
\[
f(x, y) dx = \frac{0.1}{3} [3.3140 + 4(5.0070) + 2(6.6522) + 4(8.2368) + 9.7435] = 2.6446
\]

In this example, the answer does not check well with the analytical value of 2.5944 because the \( x \)-intervals are large.

- The previous example shows that double integration by numerical methods reduces to a double summation of weighted function values. The calculations we have made could be written in the following weighted form

\[
\iint f(x, y) dx dy = \sum_{j=1}^{n} \left[ v_j \left( \sum_{i=1}^{m} w_{ij} f_{ij} \right) \right]
\]

\[
= \frac{\Delta x \Delta y}{2} [(f_{1,1} + 2f_{2,1} + 2f_{3,1} + f_{4,1}) + 4(f_{1,2} + 2f_{2,2} + 2f_{3,2} + f_{4,2}) + 2(f_{1,3} + 2f_{2,3} + 2f_{3,3} + f_{4,3}) + 4(f_{1,4} + 2f_{2,4} + 2f_{3,4} + f_{4,4}) + (f_{1,5} + 2f_{2,5} + 2f_{3,5} + f_{4,5})]
\]

\[
= \frac{\Delta x \Delta y}{2} \begin{bmatrix} 1 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 8 & 8 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 2 & 4 & 4 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 4 & 8 & 8 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 1 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} & f_{1,4} & f_{1,5} \\ f_{2,1} & f_{2,2} & f_{2,3} & f_{2,4} & f_{2,5} \\ f_{3,1} & f_{3,2} & f_{3,3} & f_{3,4} & f_{3,5} \\ f_{4,1} & f_{4,2} & f_{4,3} & f_{4,4} & f_{4,5} \end{bmatrix}.
\]
6 Numerical Solution of Ordinary Differential Equations
(常微分方程式之數值解)

• Most Problems in the real world are modeled with differential equations because it is
easier to see the relationship in terms of a derivative. (真實世界上, 大多數的問題都是利用微分程式來加以描述，這是因為問題的關係式較容易使用導數項來建立。)

Many differential equations can be solved analytically and you probably learned how to
do this in a previous course. The general analytical solution will include arbitrary
constants in a number equal to the order of the equation. If the same number of conditions
on the solution is given, these constants can be evaluated.

When all of the conditions on the problem are specified at the same value for the
independent variable, the problem is termed an initial-value problem (當所有的條件都給
定在同一個變數值上，該問題叫做初始值問題). If these are at two different values for
the independent variable, usually at the boundaries of some region of interest, it is called a
boundary-value problem (如果條件是給定在兩個以上的變數值上，該問題便叫做邊界
值問題).

This chapter describes techniques for solving ordinary differential equations by numerical
methods. To solve the problem numerically, the required number of conditions must be
known and these values are used in the numerical solution.

6.1 The Taylor-Series Method (泰勒級數法)

• Here is an example:

\[
\frac{dy}{dx} = -2x - y, \quad y(0) = -1.
\]

The analytic solution is

\[
y(x) = -3e^{-x} - 2x + 2.
\]

The Taylor series of \( y(x) \) expanded about \( x = 0 \) can be written as

\[
y(x) = y(0) + y'(0)x + \frac{1}{2} y''(0)x^2 + \frac{1}{6} y'''(0)x^3 + \frac{1}{24} y^{(4)}(0)x^4 + error.
\]
We can get the derivatives by successively differentiating the equation for the first derivative such as
\[
y'(0) = -2(0) - (-1) = 1.
\]
\[
y''(0) = -2 - y'(0) = -2 - 1 = -3.
\]
\[
y'''(0) = -y''(0) = -(3) = 3.
\]
\[
y^{(4)}(0) = -y'''(0) = -3.
\]
Table 6.1 shows how the computed solutions compare to the analytical between \( x = 0 \) and \( x = 0.6 \). At the start, the Taylor-series solution agrees well, but beyond \( x = 0.3 \) they differ increasingly. More terms in the series would extend the range of good agreement.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y(x) )</th>
<th>Anal</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00000</td>
<td>-1.00000</td>
<td>-1.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>0.10000</td>
<td>-0.91451</td>
<td>-0.91451</td>
<td>0.00000</td>
</tr>
<tr>
<td>0.20000</td>
<td>-0.85620</td>
<td>-0.85619</td>
<td>0.00001</td>
</tr>
<tr>
<td>0.30000</td>
<td>-0.82251</td>
<td>-0.82245</td>
<td>0.00006</td>
</tr>
<tr>
<td>0.40000</td>
<td>-0.81120</td>
<td>-0.81096</td>
<td>0.00024</td>
</tr>
<tr>
<td>0.50000</td>
<td>-0.82031</td>
<td>-0.81959</td>
<td>0.00072</td>
</tr>
<tr>
<td>0.60000</td>
<td>-0.84820</td>
<td>-0.84643</td>
<td>0.00177</td>
</tr>
</tbody>
</table>

### 6.2 The Euler Method and Its Modifications (尤拉法及其修正)

**The Euler Method (尤拉法)**

- The Euler method is the first truly numerical method that we discuss. It comes from using just one term from the Taylor series for \( y(x) \) expanded about \( x = x_0 \).

We can solve the differential equation
\[
\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0,
\]
by using just one term of the Taylor-series method
\[
y(x) = y(x_0) + y'(x_0)(x - x_0) + O((x - x_0)^2)
\]
or
\[
y_{n+1} = y_n + y'_n h + O(h^2). \tag{4}
\]
If the increment to \( x \) \( (x - x_0 \) or \( h \) is small enough, the error will be small. This method is easy to program provided that we know the formula for \( y'(x) \) and a starting
value \( y(x_0) = y_0 \).

For the previous example,
\[
\frac{dy}{dx} = -2x - y, \quad y(0) = -1.
\]
The computation can be done rather simply and the results are listed in Table 6.2. Here we take \( h = 0.1 \).

<table>
<thead>
<tr>
<th>( x_n )</th>
<th>( y_n )</th>
<th>( y'_n )</th>
<th>( h y'_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>-1.00000</td>
<td>1.00000</td>
<td>0.10000</td>
</tr>
<tr>
<td>0.1</td>
<td>-0.90000</td>
<td>0.70000</td>
<td>0.07000</td>
</tr>
<tr>
<td>0.2</td>
<td>-0.83000</td>
<td>0.43000</td>
<td>0.04300</td>
</tr>
<tr>
<td>0.3</td>
<td>-0.78700</td>
<td>0.18700</td>
<td>0.01870</td>
</tr>
<tr>
<td>0.4</td>
<td>-0.76830</td>
<td>-0.03170</td>
<td></td>
</tr>
</tbody>
</table>

(Analytical answer is -0.81096, error is -0.04266.)

Comparing the last result to the analytical answer \( y(0.4) = -0.81096 \), there is only one-decimal accuracy. To get an error less than 0.00024, we will need to reduce the step size about 426-fold.

**The Modified Euler Method (修正之尤拉法)**

- The trouble with this most simple method is its lack of accuracy, requiring an extremely small step size. Figure 6.1 suggest how we might improve this method with just a little effort.

The modified Euler method can be derived from using two terms of the Taylor series:
\[
y_{n+1} = y_n + y'_n h + \frac{1}{2} y''_n h^2 + O(h^3)
\]
\[
= y_n + y'_n h + \frac{1}{2} \left( y'_{n+1} - y'_n \right) + O(h^2) + O(h^3)
\]

\[
= y_n + h \left( y'_n + \frac{1}{2} y'_{n+1} - \frac{1}{2} y'_n \right) + O(h^3)
\]

\[
= y_n + h \left( \frac{y'_n + y'_{n+1}}{2} \right) + O(h^3).
\]

Table 6.3 shows the results of this modified Euler method on the same problem, \(dy/dx = -2x - y\), \(y(0) = -1\).

<table>
<thead>
<tr>
<th>(x_n)</th>
<th>(y_n)</th>
<th>(ky_n')</th>
<th>(y_{n+1,p})</th>
<th>(hy'_{n+1,p})</th>
<th>(hyn')</th>
<th>(y_{n+1,c})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>-1.0000</td>
<td>0.1000</td>
<td>-0.9000</td>
<td>0.0700</td>
<td>0.0850</td>
<td>-0.9150</td>
</tr>
<tr>
<td>0.1</td>
<td>-0.9150</td>
<td>0.0715</td>
<td>-0.8435</td>
<td>0.0444</td>
<td>0.0579</td>
<td>-0.8571</td>
</tr>
<tr>
<td>0.2</td>
<td>-0.8571</td>
<td>0.0457</td>
<td>-0.8114</td>
<td>0.0211</td>
<td>0.0334</td>
<td>-0.8237</td>
</tr>
<tr>
<td>0.3</td>
<td>-0.8237</td>
<td>0.0224</td>
<td>-0.8013</td>
<td>0.0001</td>
<td>0.0112</td>
<td>-0.8124</td>
</tr>
<tr>
<td>0.4</td>
<td>-0.8124</td>
<td>0.0012</td>
<td>-0.8112</td>
<td>-0.0189</td>
<td>-0.0088</td>
<td>-0.8212</td>
</tr>
<tr>
<td>0.5</td>
<td>-0.8212</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[y(0.5) = -0.81959, the analytical value]

6.3 Runge-Kutta Method

- What if we use more terms of the Taylor series? Two German mathematicians, Runge and Kutta, developed algorithms from using more than two terms of the series.

Second-Order Runge-Kutta Methods (2 階 Runge-Kutta 法)

- The modified Euler method is, in fact, a second-order Runge-Kutta method. Second-order Runge-Kutta methods are obtained by using a weighted average of two increments to \(y(x)\), say \(k_1\) and \(k_2\).

For the differential equation \(dy/dx = f(x, y)\), we have

\[
y_{n+1} = y_n + ak_1 + bk_2
\]

\[
k_1 = hf(x_n, y_n)
\]

\[
k_2 = hf(x_n + \alpha h, y_n + \beta k_1)
\]

\[
(6.8)
\]
We may think of the values \( k_1 \) and \( k_2 \) as estimates of the change in \( y \) when \( x \) advances by \( h \), because they are the product of the change in \( x \) and a value for the slope of the curve. The Runge-Kutta methods always \( k_1 = hf(x_n, y_n) \) as the first estimate of \( \Delta y \); the other estimate is taken with \( x \) and \( y \) stepped up by the fractions \( \alpha \) and \( \beta \) of \( h \) and the earlier estimate of \( \Delta y \), say \( k_1 \) in this case.

The choice of the four parameters, \( a \), \( b \), \( \alpha \) and \( \beta \), is finished by making Eq. (6.8) agree as well as possible with the Taylor-series expansion:

\[
y_{n+1} = y_n + h y'_n + \frac{1}{2} h^2 y''_n + \cdots
\]

with \( f'(x, y) = \frac{dy}{dx} + \frac{\partial f}{\partial y} \)

\[
y_{n+1} = y_n + hf(x_n, y_n) + \frac{1}{2} h^2 f'(x_n, y_n) + \cdots
\]

Rewrite Eq. (6.8) as

\[
y_{n+1} = y_n + a hf(x_n, y_n) + b hf(x_n + \alpha h, y_n + \beta k_1) = y_n + a hf_n + b hf_n + \beta hf_n
\]

with \( f(x + \Delta x, y + \Delta y) \approx f + \Delta x \frac{\partial f}{\partial x} + \Delta y \frac{\partial f}{\partial y} \)

\[
y_{n+1} + a hf_n + b hf_n + \beta hf_n
\]

Eq. (6.12) should be identical to Eq. (6.9). Then we have

\[
a + b = 1, \quad \alpha b = \frac{1}{2}, \quad \beta b = \frac{1}{2}.
\]

Note that there are four unknowns, but only three equations need to be satisfied. We can choose one value arbitrarily. Hence, we have a set of second-order methods such as

- The midpoint method: \( a = 0, \quad b = 1, \quad \alpha = \frac{1}{2} \) and \( \beta = \frac{1}{2} \).
- The modified Euler method: \( a = \frac{1}{2}, \quad b = \frac{1}{2}, \quad \alpha = 1 \) and \( \beta = 1 \).
- Minimum error bound method: \( a = \frac{1}{3}, \quad b = \frac{3}{4}, \quad \alpha = \frac{3}{4} \) and \( \beta = \frac{3}{4} \).

Fourth-Order Runge-Kutta Methods (4階 Runge-Kutta 法)

- Fourth-order Runge-Kutta methods are most widely used and are derived in similar fashion. Greater complexity results from having to compare terms through \( h^4 \), and this gives a set of 11 equations in 13 unknowns. The set of 11 equations can be solved with 2
unknowns being chosen arbitrarily. The most commonly used set of values leads to the selection:

\[ y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \]

\[ k_1 = hf(x_n, y_n), \]
\[ k_2 = hf \left( x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1 \right), \]
\[ k_3 = hf \left( x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2 \right), \]
\[ k_4 = hf(x_n + h, y_n + k_3). \]  \hspace{1cm} (6.13)

For the previous problem, \( \frac{dy}{dx} = -2x - y, \ y(0) = -1 \) with \( h = 0.1 \), We obtain the results shown in Table 6.4. The results here are very impressive compared to those given in Table 6.1, where we computed the values using the terms of the Taylor series up to \( h^4 \). Table 6.4 agrees to five decimals with the analytical result – illustrating a further gain in accuracy with less effort.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( k_1 )</th>
<th>( k_2 )</th>
<th>( k_3 )</th>
<th>( k_4 )</th>
<th>( k_{\text{avg}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>-1.00000</td>
<td>0.1000</td>
<td>0.0850</td>
<td>0.0858</td>
<td>0.0714</td>
<td>0.0855</td>
</tr>
<tr>
<td>0.1</td>
<td>-0.91451</td>
<td>0.0715</td>
<td>0.0579</td>
<td>0.0586</td>
<td>0.0456</td>
<td>0.0584</td>
</tr>
<tr>
<td>0.2</td>
<td>-0.85619</td>
<td>0.0456</td>
<td>0.0333</td>
<td>0.0340</td>
<td>0.0222</td>
<td>0.0338</td>
</tr>
<tr>
<td>0.3</td>
<td>-0.82246</td>
<td>0.0222</td>
<td>0.0111</td>
<td>0.0117</td>
<td>0.0011</td>
<td>0.0115</td>
</tr>
<tr>
<td>0.4</td>
<td>-0.81096</td>
<td>0.0011</td>
<td>-0.0090</td>
<td>-0.0085</td>
<td>-0.0181</td>
<td>-0.0086</td>
</tr>
<tr>
<td>0.5</td>
<td>-0.81959</td>
<td>-0.0180</td>
<td>-0.0271</td>
<td>-0.0267</td>
<td>-0.0354</td>
<td>-0.0268</td>
</tr>
<tr>
<td>0.6</td>
<td>-0.84644</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(The analytical value of \( y(0.6) \) is \(-0.846434\).)

Figure 6.2 illustrates the four slope values that are combined in the four \( k \)'s of the Runge-Kutta method.
The Runge-Kutta techniques have been very popular, especially the fourth-order method just presented. Because going from second to fourth order was so beneficial, we may wonder whether we should use a still higher order of formula.

**Fifth-Order Runge-Kutta Methods**

- As an example, the Runge-Kutta-Fehlberg method are listed:

\[
y_{n+1} = y_n + \left( \frac{16}{135} k_1 + \frac{6656}{12825} k_2 + \frac{28561}{56430} k_3 - \frac{9}{50} k_4 + \frac{2}{55} \right),
\]

\[
k_1 = h \cdot f(x_n, y_n),
\]

\[
k_2 = h \cdot f(x_n + \frac{h}{4}, y_n + \frac{k_1}{4}),
\]

\[
k_3 = h \cdot f(x_n + \frac{3h}{8}, y_n + \frac{3k_1}{32} + \frac{9k_2}{32}),
\]

\[
k_4 = h \cdot f(x_n + \frac{12h}{13}, y_n + \frac{1932k_1}{2197} - \frac{7200k_2}{2197} + \frac{7296k_3}{2197}),
\]

\[
k_5 = h \cdot f(x_n + h, y_n + \frac{439k_1}{216} - 8k_2 + \frac{3680k_3}{513} - \frac{845k_4}{513}),
\]

\[
k_6 = h \cdot f(x_n + \frac{h}{2}, y_n - \frac{8k_1}{27} + 2k_2 - \frac{5544k_3}{2565} + \frac{1859k_4}{4104} - \frac{11k_5}{4104}),
\]

with global error \(O(h^5)\).

- A summary and comparison of the numerical methods we have studied for solving \(dy/dx = f(x, y)\) is presented in Table 6.5.

**Table 6.5**

<table>
<thead>
<tr>
<th>Method</th>
<th>Estimate of slope</th>
<th>Global error</th>
<th>Local error</th>
<th>Evaluations per step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euler</td>
<td>Initial value</td>
<td>(O(h))</td>
<td>(O(h^2))</td>
<td>1</td>
</tr>
<tr>
<td>Modified Euler</td>
<td>Average, initial and final</td>
<td>(O(h^2))</td>
<td>(O(h^3))</td>
<td>2</td>
</tr>
<tr>
<td>Midpoint</td>
<td>Midpoint of interval</td>
<td>(O(h^2))</td>
<td>(O(h^3))</td>
<td>2</td>
</tr>
<tr>
<td>Runge–Kutta (fourth-order)</td>
<td>Weighted average, four values</td>
<td>(O(h^3))</td>
<td>(O(h^5))</td>
<td>4</td>
</tr>
<tr>
<td>Runge–Kutta–Fehlberg</td>
<td>Weighted average, six values</td>
<td>(O(h^5))</td>
<td>(O(h^5))</td>
<td>6</td>
</tr>
</tbody>
</table>

To see empirically how the global errors of Table 6.5 hold, again we consider the example \(dy/dx = -2x - y, \ y(0) = -1\). Table 6.6 shows how the errors of \(y(0.4)\) decreases as \(h\) is halved.
How Do We Know If the Step-Size Is Right? (步伐大小之決定)

- One way to determine whether the Runge-Kutta values are sufficiently accurate is to recompute the value at the end of each interval with step size halved. If only a slight change in the value of \( y_{n+1} \) occurs, the results are acceptable; if not, the step must be halved again until the results are satisfactory. However, this procedure is very expansive.

A different approach, being of less function evaluations, uses two Runge-Kutta methods of different orders. For instance, we could use one fourth-order and one fifth-order method to move from \((x_n, y_n)\) to \((x_{n+1}, y_{n+1})\). We would then compare our results at \(y_{n+1}\).

### 6.4 Multi-step Methods (多步法)

- Runge-Kutta methods are called single-step methods because they use only the information from the last step computed. The most important advantages of the single-step methods are that
  1) they have the ability to perform the next step with a different step size,
  2) they are ideal for beginning the solution where only the initial conditions are available.

After the solution has begun, however, there is additional information available about the function if we are wise enough to retain it in the memory of the computer. A multi-step method is one that takes advantage of this fact.
Multi-Step Methods

- The principle behind a multi-step method is to utilize the past values of $y$ and/or $y'$ to construct a polynomial that approximates the derivative function, and extrapolate this into the next interval.

Since change of step size with the multi-step methods is considerably more awkward than the single-step methods, most multi-step methods use equal step size to make the construction of the polynomial easy. The following Adams multi-step methods are typical:

Adams third-order formula: \[ y_{n+1} = y_n + \frac{h}{12} (5 f_{n-2} - 16 f_{n-1} + 23 f_n) + O(h^4). \]

Adams fourth-order formula: \[ y_{n+1} = y_n + \frac{h}{24} (-9 f_{n-3} + 37 f_{n-2} - 59 f_{n-1} + 55 f_n) + O(h^5). \]

Observe that the above formulas resemble the single-step formulas in that the increment in $y$ is a weighted sum of the derivatives times the step size, but differ in that past values are used rather than estimates in the forward direction.

The Adams-Moulton Method

- An improvement over the Adams method is the Adams-Moulton method. It uses the Adams method as a predictor formula, then applies a corrector formula.

Predictor: \[ y_{n+1} = y_n + \frac{h}{24} (-9 f_{n-3} + 37 f_{n-2} - 59 f_{n-1} + 55 f_n + \frac{251}{720} h^5 y^{(5)}(\xi_1)). \]

Corrector: \[ y_{n+1} = y_n + \frac{h}{24} ( f_{n-2} - 5 f_{n-1} + 19 f_n + 9 f_{n+1}) - \frac{19}{720} h^3 y^{(5)}(\xi_2). \]

- The efficiency of Adams-Moulton method is about twice that of the Runge-Kutta and Runge-Kutta-Fehlberg methods. Only two function evaluations are needed per step for the former method, whereas four or six are required with the single-step alternatives. All have similar error terms.

Changing the Step size

- When the predicted and corrected values of the Adams-Moulton method agree to as many decimal as the desired accuracy, we can save computational effort by increasing the step size. We can conveniently double the step size by omitting every second one.
When the difference between predicted and corrected values exceeds the accuracy criterion, we should decrease step size. Convenient formulas for this are
\[
y_{n-\frac{1}{4}} = \frac{1}{128} \left( 3y_{n-4} - 20y_{n-3} + 90y_{n-2} + 60y_{n-1} - 5y_n \right),
\]
\[
y_{n+\frac{1}{4}} = \frac{1}{128} \left( -5y_{n-4} + 28y_{n-3} - 70y_{n-2} + 140y_{n-1} + 35y_n \right).
\]
Use of these values with \( y_{n-1} \) and \( y_n \) gives four values of the function at intervals of \( \Delta x = \frac{1}{2} h \).

**Homework #8**

Solve \( \frac{dy}{dx} = x + y - xy, \ y(0) = 1 \) at \( x = 0.1, 0.2, 0.3, 0.4, 0.5 \).

a) Use the Taylor series method with terms through \( x^4 \).

b) Use the fourth-order Runge-Kutta method with \( h = 0.1 \).

c) Use the Adams-Moulton method to solve
   i) \( y(0.4) \) with known \( y(0), \ y(0.1), \ y(0.2), \ y(0.3) \),
   ii) \( y(0.5) \) with known \( y(0.1), \ y(0.2), \ y(0.3), \ y(0.4) \).