詳論C語言

與8086組合語言呼叫

■深入探討IBM PC/MS-DOS系統呼叫
、鍵盤、螢幕、CGA/EGA ROM BIOS繪圖常式

蔡明峯　編譯

Supercharging C
with Assembly Language

數值方法

譯　者：姚　修　慎
發行人：楊　鏡　秋
出版者：儒林圖書有限公司
地　址：台北市重慶南路一段111號
電　話：3118971-3　3144000
郵政劃撥：0106792-1號
吉豐印刷廠有限公司承印
板橋市三民路二段居仁巷一弄五十三號
行政院新聞局局版台業字第1492號

中華民國七十四年五月初版
定價新台幣 220 元正

序

本書的目的是希望以最清晰的方式，爲您提供一個關於數值方法的邏輯性結構，書中所提到的方法，都很適合透過使用數位計算機來解決科學與工程方面的許多問題。我們儘可能地把進展神速的數值方法上的那些最新、最有效率的技術加到書中，但却也不漏掉那些仍然在被廣泛使用且已經有了良好基礎的老方法。

因爲我們的重點是在於了解以及使用各種方法，所以只有在爲了對某一個方法做進一步了解或提供進一步的動機時才會對它做出證明；當提出一個方法之後，如果在能夠立即透過使用這個方法而加強對它的認識時，我們也加進了不少例題。另外，在每一章的後面都加上了一些詳細解說了的題目，它們幾乎涵蓋了該章中的任何論題，並且說明了各種方法的優點，以及它們的潛在困難。在這本書中唯一的數學背景，就是初等微積分，正因爲這樣，這本書不但可以做爲一本有計劃的教學教材，而且也可以做爲其它書本的輔助自修手册。

在這本書中我們並沒有順著目前的趨勢，把每一個方法都附上一個程式，因爲依作者的經驗，這樣會造成學生只不過把程式摘下來，輸入並執行而已，而不會實際地嘗試去了解那一個方法的眞諦，不過爲了方便起見，我們還是加上了一些參考程式，但並不是每一個方法都有；此外，雖然我們加上了BASIC的程式，但是却並不意味著這樣要侷限在一個語言上，因爲基於不同的理由，有時候我們得要使用其它的語言，像FORTRAN、PL／I，APL等來寫作程式。

雖然本書中有很多例子都對應著物理現象的數學模式，不過它們通常是以數學的面目出現。因爲這本書並不限定在工程或科學訓練上，因此這樣做的好處就方法與問題方面的關聯而言，是可以讓授課人自行提出該例題在某一範疇中。不過，作者却不希望在學生能夠了解基礎數值方法之前，就提出很複雜的，包含了很多物理現象方面的問題，因爲這樣做就會導致學生迷失或者因而錯

導在物理問題上頭，而不能得到數值方法上的經驗。

第一章介紹了數值方法的效力與限制，對於工程師科學家研究這些方法的動機以及從使用人觀點出發對數位計算機的一個簡介。至於數值方法的基石，泰勒級數與有限差分別在第二與第三章中討論。

第四章的課題是內插法。雖然這一項課題一直是相當重要，並且也是數值分析中大部分的理論基石，但是很多課本却因爲某些理由而全部避而不談，或者是討論得不夠。第五章的重點在於解方程式的根，很多方法都可以直接從泰勒級數發展出來，但是反內插法卻也是相當有用的。

第六章中討論直接以及遞回的技巧來解線性聯立方程式組，特別地我們會討論條件不良的問題，以及數目異常龐大的方程式組。

第七章則是把內插法的概念擴充到函數的逼近上去，透過第六章的技巧，最少平方法用來配合資料的技巧才能夠用比較有效率的方法來討論。

數值積分在第八章討論，我們提出了幾個高精確度，高效率的方法，並且正說到一些奇異點的方法。

到了第九章，我們討論到爲了解微分方程式的好些個數值技巧，並且也考慮到了解初值與邊界值的論題；對於每一個方法的精確度與效率也仔細地討論到了。

第十章則是代數特徵值（或固有值）的討論，我們特別強調了對某一類問題選擇最有效的技巧的論題。

第十一章中我們加進了用數值方法解偏微分方程式的簡介特別是拋物式與橢圓式類型；我們也簡單地討論到有限元素的威力與潛力。

這本書的內容對於工程或科學方面的大三或大四階層一學期的課程來說是非常充足的，但是若時間不足以涵蓋整本書內容的話，我們建議您跳過 7.2節以及第十章中比較深的部分，另外第十一章因爲與本書的邏輯關聯不大，因此也可以不教。

目　錄

第1章
重點介紹

1.0 前言

本章將簡短的討論數值方法的目的、能力和限度；並且提供了對這些數值方法詳細研究的正確指引。

1.1 什麼是數值方法？

數值方法是數學中一種可以處理廣泛問題的一套方法。這些問題都源自於實際問題的數學模型。數值方法尤其特別的是它僅利用到數學與邏輯兩種運算，因此可以用數位計算機來執行。

雖然以嚴格定義來說，手指頭與算盤都可算是數位電腦，但在此我們都指1950年代中期以來使用的電子式儲存程式型電腦。數值方法的歷史比數位電腦要早很多，事實上數值方法中有許多是在現代數學萌芽初期就已經發展出來。這些方法的使用在開始時頗受限制，直到桌上型計算器的發明，而後數位計算機的產生，才使數值方法大大的發生作用。

數值方法和數位電腦的結合創造了數學分析上強而有力的工具。比如說，數值方法可以處理非線性，複雜的幾何學，以及模擬實際問題的複雜模型（通常是一組龐大的方程式）。傳統的數學，即使是那些最精妙的應用數學，也無法克服現代技術所須解決問題的程度，所以，在許多方面，數值方法取代了傳統的數學分析。在工業界及研究應用方面，傳統分析方法都較少被採用；即使分析解可以得到，但由於數值方法通常成本較低，且有現成的程式可利用，所

以大家都採用數值方法。

1.2 數值方法有它的限度嗎？

這答案是肯定的。對於門外漢或是許多知識淵博的科學家及工程師而言如果沒有別的方法，唯一可能的就是讓電腦來解。這種看法無疑的是因爲前一節中所談及的數值方法巨大的能力所造成的。不幸的，有些問題用數值方法也一樣沒有辦法解決（有時我們用“不實際（impractical）”的字眼），因有些問題中是沒有精確且完善的數學模型可加以解釋，換句話說，它們根本沒有數學解。另外有些問題則因超過現今的電腦能力，而變得不可解。比如說，你想用電腦解一個時變暴風流體問題，同時考慮流體中所有漩渦的效應，如果要詳細解出結果，那麼電腦可能要花 30 年的時間來解。這項估計是以 1968 年的技術來衡量，若以今天的技術來比較，也不會快過 5 倍。然而，實際上是否能得完全解，和你所願意花的代價有關。有些問題實在是太重要，以致於工業界及政府願意花下數百萬元去得到所需的電腦容量及速度，將從前被認爲實際不可能解出的問題解出。雖然這種能力界限一直不斷地在擴大，但在大部分情況下。許多問題仍無法由數學模式的原理或現代的技術解決。

1.3 何以要學數值方法？

這似乎很奇特，雖然數值方法現在已經廣泛的使用在科學界、技術界及政府，作者本人仍感覺到有義務必要去證實學習數值的方法。以現在或將來的眼光來看，數值分析師及電腦學家無疑仍是需要的。

但對工程師和科學家而言，也有些人對數值方法不太清楚。近些年來，許多大的電腦程式，都是經過許多人年（man-year）的工作量而完成的，以用來模擬實際的複雜世界問題。這些程式的設計使用，通常不須過於了解程式內在的運作。並且有一個副程式庫（libraries of subprograms），它能執行許多非常複雜的數值方法以解決不同的數學問題。由於這種原因，有人懷疑工程師或科學家是否有必要要求去了解數值方法之理論。但是，一個工程師及科學家如果只是想要使用事先寫好的程式或副程式庫的程式，他將會失望的。選擇和利用數值方法，許多地方仍是藝術性而非科學性的工作，且如果一個

電腦使用者沒有辦法對特殊問題具設計和選擇數值的方法，他在執行程式時，將會發現所能處理的問題相當有限。

當套裝程式或可利用的副程式恰適合於你的工作時，叫用它們顯然是最有效的過程。即是在此種情形下，具有數值方法之知識仍是非常有用的。不論如何，運用這種現成的程式集及副程式仍會碰到許多問題，可概分如下：

(a) 沒有辦法用數學模型去精確的模擬實際的世界。（這點極爲重要，但不是本書討論之範圍）

(b) 沒有一個數值方法在任何情況下都不會產生問題（trouble-free）。

(c) 沒有一個數值方法是完全不產生誤差（error-free）的。

(d) 沒有一個數值方法在任何情況下是最佳的（optimal）。

（在上述(b)，(c)，(d)中有重疊的部分，我們在此只是說明一般的觀念，並不給予嚴格精確的定義）困難在於事先的套裝程式或庫存副程式也有可能會得到錯誤的結果，甚至沒有結果。並且，當使用者在程式庫中搜尋時。會有很多可用的副程式；但這種可用只是一般性的，這些副程式在所附的說明中，很少會提及副程式本身的效率或者在解決特殊問題上的適用性。

如果使用者遇到了這些問題，而又不懂數值方法，他就必須尋求協助了可能是數值分析師）；當然前提是此種磋商者必需要存在。在此種情形下，使用者也很難問出正確的問題，磋商的人也不一定給予有用的答案，這是由於二者知識背景不同所造成的。

可見，許多證據顯示科學家和工程師必須要有數值方法的工作知識。這種知識能讓使用者選擇，修改及撰寫適合某特定工作的程式，並且能幫助使用者選擇套裝程式及副程式。當遇到困難時，此種知識也能幫助使用者和專家作溝通。最後，大部分的“方法發展”（methods development）——也就是用程式來模擬實際世界——是由科學家和工程師來作，而不是數值分析家。顯然的，在此類工作中，需要有效而精確的數值技巧，且科學家和工程師也需要具備廣泛數值方法的知識。

現在我們將開始討論許多和電腦有關而和數值方法較無關的問題，這對於想用數位電腦來執行數值方法的使用者將是很有趣的。

1.4 電腦語言

大部分的讀者可能都使用過像FORTRAN，ALGOL及BASIC 等高階語言來撰寫程式。這種高階語言提供使用者一種具有代數公式、英語式邏輯和輸出入指令的語言型態。這種高階語言實際上和執行此程式的機器是無關的（independent）。然而，用一種叫作編譯程式（compiler）將此種高階語言轉換成基本的機器碼，再用此機器碼來執行。

廣泛用來處理科學目的的高階語言是 FORTRAN IV。除了少數的例外，ALGOL 語言並不用來處理科學計算，而是做爲演算法 (althgorithm) 的描述。BASIC 語言是一個經常被使用在分時（time sharing）系統及簡單的程式上。其他被用來做科學使用的高階語言尚有APL （廣泛運用在分時系統上，可處理極簡單到極繁難的工作）。MAD（類似ALGOL 語言），及 PL-1（一種非常有能力的語言）。

每一種新語言出來時，對一般使用者都會造成恐慌，這意味著又需要重新學習，以及會和其他語言產生混淆。然而，任何腦筋靈活的人將會發現如果需要，學習新語言並不是很困難。引起最大爭議的是經濟因素，由於發展大型電腦程式是非常昂貴的，而由一種語言系統再去發展另一種語言系統更是需費時數月的大工作。這就是爲什麼 FORTRAN IV至今仍是“標準”科學用的語言。而且在最近的未來仍無法被它種語言取代的原因。

1.5 證實問題（程式的驗證（偵錯））

一個必須且最困難的工作，就是在求數值解時，必須證明你的結果是正確的。首先，程式必須如程式師撰寫時所期望的，也就是說，編碼（code）必須正確。這可以靠印出中間結果及對特定部分用手算或桌上型計算器來加以檢查而得到。驗證的第二部分是要證實演算法（algorithm）必須得出正確的結果。因爲一個問題的正確解不能事先預測得到（否則我們就不需要數值解了），所以證實過程變得非常間接。此種間接的方法比如說，如果問題的某些特殊情況已經知道正確解，於是用這個正確解來驗證數值方法的正確性。這種設定的方法可能是使某幾項變成 0，或使某些項成常數，或是將其定爲非常大或非

常小，或者略過某些程式段落，也可能在程式中插入某些段落等都可以達成此目的。

在許多情形下，證實過程比求答案還花時間。然而，你花在程式證實上的投資換取你對程式結果的信任。在求數值解時，由所需時間和所需成本的估計來作折衷，可以適度放寬“證實”問題的執行程度。

直到這裏，我們都在討論一般使用者特殊程式的證實問題。至於一般程式或是程式庫中的副程式，它們的證實問題過程和前述的很類似，但要花更多的心血。同時要進行許多“最壞可能”的測試，以證實是否能切合某些困難問題之需要。

1.6 電腦會犯錯嗎？

就某些方面而言，電腦是會犯錯。但我們必須知道，絕大部分在電腦中發生錯誤是由於使用者自己所造成的。有時人們難以接受這些錯誤是自己造成的觀念。不論如何，除非所有可能的錯誤都已經消除，否則最有效的偵錯過程都是假設自己犯錯。

假如眞是電腦錯誤，可以分成兩類——硬體及軟體錯誤。眞正的硬體錯誤很少見，在此不予討論。軟體錯誤可能來自三方面：作業系統、編譯程式以及程式庫中的副程式。

作業系統（以下爲其可能的英文名字：computer executive system／operating system／monitor／supervisor／exec）的錯誤會讓使用者迷惑。現代系統通常爲了能充分利用硬體，而採多重處理（multiprocessing）方法，一次可處理許多程式；也可以同時准許很多使用者計算或和系統“交談”（此稱爲分時系統——time sharing）。使用者所遭遇之問題大部分是出在作業系統發生了錯誤，而使兩個程式在無法事先預知的情況下產生了交互作用，這也會使整個系統的程式產生錯誤動作及結果。不過此種錯誤很少重複出現，你可以重新執行你的程式以去除錯誤。

編譯程式（compiler）的錯誤會使正確的高階語言產生不正確的機器語言，幸運的是，由於執行“證實”工作，許多此種錯誤都可避免掉。然而，如果編譯程式中包含最佳化的程序，許多不可預測的錯誤將會發生。最佳化工作

是把由高階語言翻譯來的機器碼加以修改，產生同功能而最有效率的碼。此種工作，可能會把操作加以對調，以減少電腦執行時間，而越有效的最佳化程式（optimizer），就越可能產生不正確的機器碼。解決此種錯誤的方法，可以只用編譯程式，而"關掉"最佳化程序，或是換一個沒有最佳化程序的編譯程式；或是採用無錯誤（error-free）的最佳化程式。在此有一個建議，在除錯（debugging）或起始執行程式時，不要使用最佳化程式，只有對效率要求極高時，才採用它。當然，程式結果在作最佳化和不作最佳化之間最好也相互比較。

在程式庫中的副程式錯誤通常是"證實"工作不徹底的結果。使用者無法自行處理，最好是將錯誤的程式告訴系統人員。

最後必須指出，由於電腦有限的數字會造成捨入誤差（round off error）。事實上任何數值方法都會產生這種誤差，以後在介紹每一種數值方法時都會詳細討論該種方法的誤差問題。

1.7 結語

數值方法不能單純的來讀它，必須用過才能領會。這需要讀者確實用本書所教方法去解問題。在本章結束之前作者以個人經驗指出，要知道自己是否懂數值方法的最好途徑是去寫電腦程式，而不是用手型計算器去執行（雖然這在剛開始去了解方法的邏輯時是很有用的），如此將因爲得出的結果精確不含混，而使混淆的觀念變得清晰。

第2章

泰勒級數

2.0 前言

泰勒級數是數值方法的基礎，許多數值技巧都是直接由它推導出來，且此法也包括了誤差的估計。有些讀者可能已經在微積分中學過了泰勒級數，但由於強調的主題不同，在此仍需作一些簡短的研究。

如果一個函數 f(x) 能夠在某一點 x = a 的附近展開成冪級數：

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a) + \frac{(x-a)^3}{3!}f'''(a) + \cdots + \frac{(x-a)^n}{n!}f^{(n)}(a) + \cdots \qquad (2.1)$$

則稱 f(x) 在 x = a 附近爲可解析的（analytic），且（2.1）式是唯一的，稱之爲 f(x) 在 x = a 隣域的泰勒級數展開（Taylor series expansion）。我們很難證明 f(x) 的（2.1）展式一定存在及收斂的一般條件，但 f(x) 在 x = a 的任意次導數必須是存在且爲有限值。假如泰勒級數存在，可由計算 f(a) 和 f(x) 在 a 點的導函數值而得到在 a 附近某一點 x 的精確近似 f(x) 值。

當 | x − a | 遞增，且達到某一點 x 時，（2.1）式不再收斂，我們稱此爲不再“充分靠近” x = a，或開始超過了收斂半徑（radius of convergence）。有些級數的收斂半徑爲無限長，有些則只有有限長度。當級數收斂時，將可精確的算出 f(x) 值，這對我們而言是非常有用的。我們的興趣只在於求（2.1）式的前幾項，它能如何逼近 f(x)？

我們用圖 2 - 1 來加以說明。假設我們欲求 f(b)，由(2.1)式，得：

$$f(b)=f(a)+(b-a)f'(a)+\frac{(b-a)^2}{2!}f''(a)+\frac{(b-a)^3}{3!}f'''(a)+\cdots \qquad (2.2)$$

假如只有(2.1)式的第一項被使用到，函數是一個常數，f(a)，如圖 2 - 1a 。假如首兩項被用到，則 f 函數在 x = a 的斜率將被用上，得到一個以 f'(a)爲斜率，經過 f(a)的直線，如圖 2 - 1b 。當加入第三項以後，

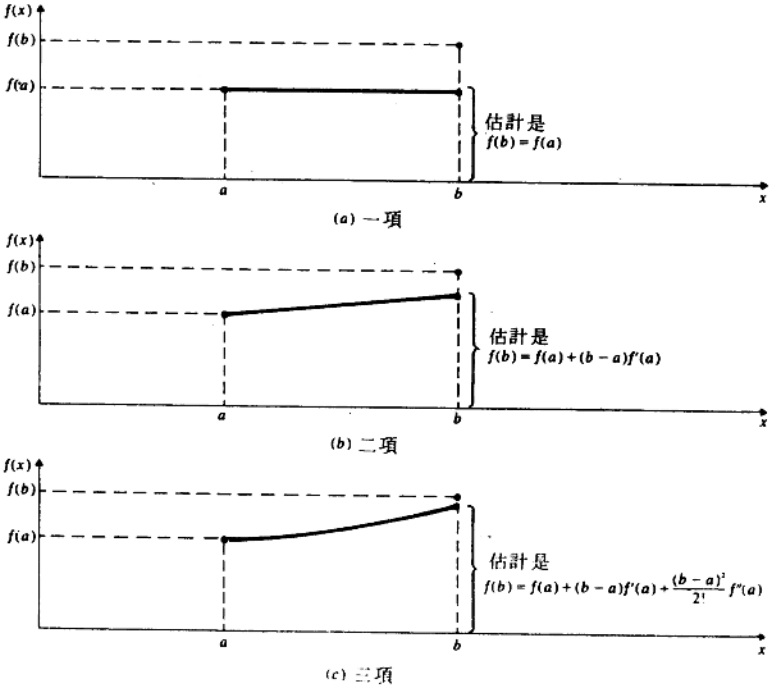

圖 2 − 1　使用部分泰勒級數的近似結果