簡單的演算法：以計算方程式的根爲例

last modified March 26, 2012

求解

$$f(x) = 0$$

計算方程式的根已經是老問題了, 不過卻是一個頗不簡單的問題。多年來各路英雄好漢提出各式各樣的方法, 針對不同的函數, 用紙筆推導或應用電腦及數值的方法。特別當電腦的速度逐漸提昇, 許多紙筆很難推導或甚至解不出來的問題, 慢慢在數值方法的演進中得到解決。本單元介紹 MATLAB 計算方程式根的指令與使用方法, 如 roots 及 fzero。另外是幾個著名的演算法, 如牛頓法、堪根法及定點法, 都是計算方程式根的典範。

本章將學到關於程式設計

演算法的程式寫作、迴圈中斷的技巧、程式偵錯(Debug) 的技巧及副程式與匿名函數的應用。

〈本章關於 MATLAB 的指令與語法〉

指令:roots, fzero, while, break, quad

語法: 無限迴圈的寫法

1 背景介紹

有些方程式的解可以透過數學的技巧推導出結果 (closed-form solution), 但有很多方程式根本寫不出完整的解。譬如: 求解

$$x - e^{-x/2} = 0$$

於是開始衍生以迭代法 (iteration) 的觀念, 逐步求解, 在牛頓的時代便已經有這樣的觀念與做法。不過受限於人力所能演算的能力, 往往祇能解決一些相對簡單的方程式。複雜一些, 或函式的計算比較難的問題, 直到電腦運算能力提昇, 才得到解決。而電腦取代人手, 只是以極快速度重複迭代的過程。但如何迭代? 如何判斷是否爲根? 何時停止迭代? 是否重根? 如何找到所有的根? 這些問題還是要回到紙筆的推導。不過不再是推導出完整的解, 而是退而求其次的, 去推導如何找到近似解的方法 (或稱演算法 algorithm)。凡事從簡單的入手, 假設要求解

$$f(x) = x^2 - 3x + 2 = 0 \tag{1}$$

這是一個有完整解 (closed-form solution) 的方程式, 可以透過數學求解的技巧找到答案。不過在此, 我們將透過數值的演算方法及電腦程式來解決。

以數值演算法求解的方法很多, 牛頓法是其中的代表, 是以遞迴迭代的方法求解的典型。其步驟假設 x_0 是任一的 x 值, 從 x_0 開始, 到 $x_1, x_2, \cdots$ 逐步往方程式的某個根移動。牛頓法定義了「步伐」移動的方式:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

或者寫成,

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, k = 0, 1, 2, \cdots \tag{2}$$

透過圖1可以清楚瞭解牛頓法的原理。跟著牛頓法的脚步, 很清楚的發現移動的方向 $(-f(x_k)/f'(x_k))$ 一定是朝向方程式的某一個根。至於要移動多少步才能到達「目的地,」在正確的執行程式後, 不難發現與初始值的選擇及函數的「樣子」有關。

以下的練習將協助初學者慢慢的探討這類演算法的過程。

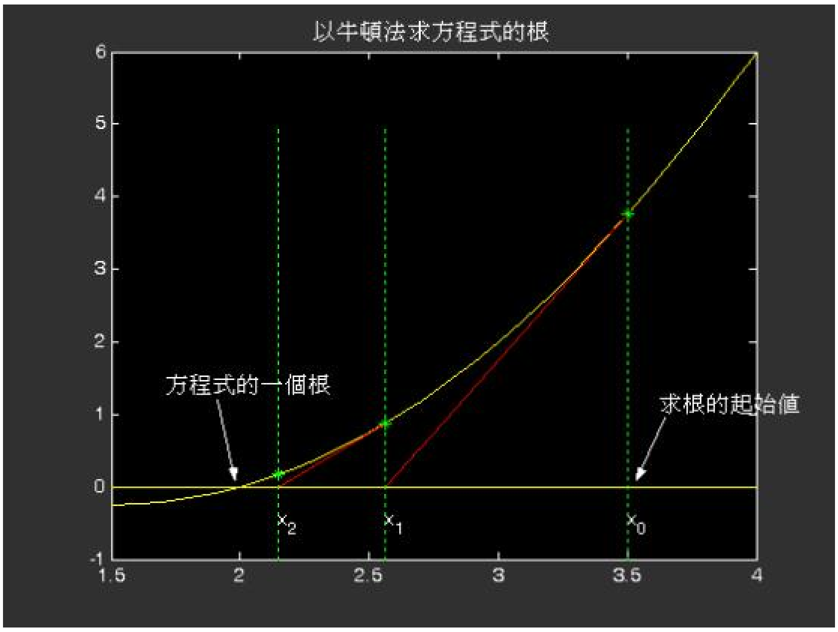

圖 1: 以牛頓法求方程式的根

2 練習

範例1: 寫一支程式, 按牛頓法的移動方式, 計算方程式(1) 的根。從初始值 $x_0 = 3.5$ 開始, 計算到第三步 x_3 , 並逐步列印出每一步的 x 與 $f(x)$ 值。

觀察 x_1, x_2, x_3 是否逐漸往其中的一個根逼近 (或說 $f(x)$ 是否愈趨近 0)? 本範例之初始值 $x_0 = 3.5$ 位在所有根的右方, 當初始值改爲 -1(在所有根的左邊) 時, 移動的方向是否改變了呢? 是否也是朝著某個根呢? 如果初始值選在兩個根的中間, 下一步會朝那個方向移動呢? 不妨實際試試看。

程式是一連串的指令 (動作) 按照某些邏輯組合而成, 初學者往往不知從何開始。而 MATLAB 是學習程式寫作很好的工具, 其動作邏輯與指令的表達方式非常接近數學式, 因此初學者不妨依數學式的解題順序來寫作程式, 再從觀摩他人的程式

學習到所謂的程式技巧。以本題爲例, 你如何用紙筆推導牛頓法到第三步, 將過程直接轉換成相對的 MATLAB 指令即可, 雖然過程嫌瑣碎了些, 但是先完成目標才是首要任務, 做出正確的結果後, 再來慢慢修理程式, 讓它更精簡、更有效率、更好維護。相信程式技巧一定會在多次的練習中逐漸進步。

數學推導	MATLAB 程式 (指令的對應組合)
$f(x) = x^2 - 3x + 2$	f=@(x) polyval([1 -3 2],x);
$f'(x) = 2x - 3$	fp=@(x) polyval([2 -3],x);
let $x_0 = 3.5$	x0=3.5
$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$	x1=x0-f(x0)/fp(x0)
$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$	x2=x1-f(x1)/fp(x1)
$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}$	x3=x2-f(x2)/fp(x2)

多項式的根一般而言相對的單純,MATLAB 提供的指令 roots 可以直接計算多項式方程式所有的根, 以本範例爲例, 指令如下

$p = [1\ \ -3\ \ 2]$	%多項式的係數
$roots(p)$	%呼叫 roots

範例2: 由於牛頓法的每個「步伐」的計算方式是固定的, 加上到達「目的地」的步數不同, 程式的寫作上必須利用迴圈的方式, 重複同樣的動作。請改寫範例1的程式, 以迴圈的方式讓 x_k 從 $k = 0, 1, 2, \cdots$ 不斷的遞迴迭代, 每個迴圈走一步。

程式中遞迴迭代的方式是典型的程式設計技巧。由於程式中不可能維護 $x_0, x_1, \cdots$ 這樣不知確定數量的變數, 且後面的值由前一個產生, 在程式設計上通常只用兩個變數配合迴圈來運作, 下面的程式片段說明這個技巧。

$N = 10$	% 代表迴圈數, 先設定爲 10
$x_k = 3.5$	%先給定初始值, 迴圈中則代表現值
$for\ \ i = 1 : N$	%開始迭代迴圈
$\quad x_{k1} = x_k - f(x_k)/fp(x_k)$	%根據牛頓法從現值計算新值
$\quad x_k = x_{k1}$	%進行下一步前, 新值換成現值。
end	

程式中的迴圈數先設定為 10, 其實是個權宜之計, 我們事先並不知道牛頓法何時收斂, 暫時先執行幾步並觀察 x_{k1} 的變化。稍後會討論在程式中加入收斂判斷的技巧。

範例3: 延續上一個範例, 將求解的過程逐步畫出來, 如圖1所示。不但將 $x_0, x_1, x_2, \cdots$ 一一畫出, 連同過程中的切線 (切線方程式先以紙筆導出適當的公式, 再放入程式中) 也要畫出來。每條線或每個文字畫完之後可以利用 pause(秒數) 的指令讓過程呈現暫停數秒的效果。否則以現在電腦的速度, 眼睛還來不及反應的瞬間, 圖上所有的內容都畫好了, 看不出「過程」與收斂的情況。

在圖上將演算過程畫出來是件令人興奮的事, 可以清楚的觀察到演算法的演進, 提高對數學的興趣。不過卻是要一步一步來, 譬如圖1上的垂直線、切線, 甚至 x_0, x_1, x_2 的文字, 都要仔細琢磨, 一個個畫上去。首先必須先確定程式大致上沒問題, 再簡單的畫上 x 的演進過程並在圖上標示出位置, 譬如,

$text(x, y,' X')$ %在座標 (x,y) 處畫上文字符號 X

圖2展示這個作法, 在演算的迴圈裡面執行上述指令 (當然得使用正確的 x, y 變數)。成功之後, 再慢慢加上垂直線及切線。

3 觀察

1. 以演算法求解通常面臨『何時停止演算』的問題, 否則程式會繼續下去, 沒完沒了, 也就是迴圈的「圈數」如何決定。不過剛開始可以先將迴圈數設大一點, 再觀察程式, 大約在演算多少迴圈後答案會趨於『穩定』?

2. 迴圈指令除了 for 之外, while 是另一個常用的方式, 他的優點是不涉及迴圈數的決定。有關 while 的用法, 請自行參考 help 的範例, 或參考稍候的程式範例。

3. 即便事先不知道迴圈數依然可以使用 for 指令, 只不過要設定大一點。然後迴圈內必須做出跳出迴圈的條件判斷, 當條件滿足時, 利用 break 指令跳出

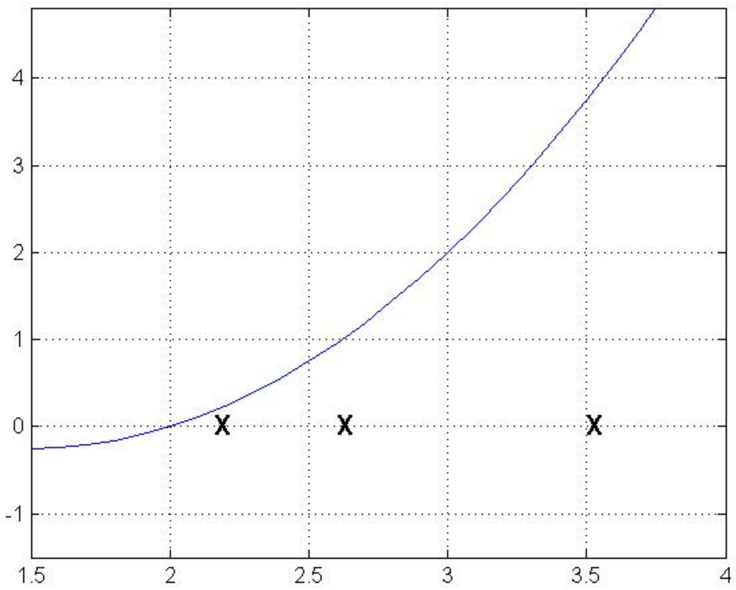

圖 2: 演算過程的註記

迴圈。譬如:

```
for  i = 1 : 10000        %開始迭代迴圈
         ⋮
     if 條件判斷式         %判斷式否已經到達目的地
         break;           %跳出迴圈
     end
         ⋮
end
```

4. 在逐步迭代的演算法中, 設定停止點或迴圈的中斷點是必要的, 否則迭代的過程將無止盡的進行。什麼時候該停止迭代呢? 換句話說, 該如何決定已經到達目的地, 找到根了呢? 是在第 $k+1$ 步的函數值 $f(x_{k+1})$ 很接近 0, 還是 $f(x_{k+1}) \approx f(x_k)$, 或是 $x_{k+1} \approx x_k$? 哪一個比較恰當? 與方程式有關嗎? 上述程式中的「條件判斷式」便是依這些原則來判斷式否該終止迭代,

找到答案 (根)。試著在演算法裡面訂一個停止點。譬如

$$
\begin{array}{lll}
if \;\; abs(f(x_{k1})) < 0.00001 & & \%\text{多小的值才適當必須依函數而定} \\
\qquad\quad break; & & \%\text{跳出迴圈} \\
end & &
\end{array}
$$

5. 演算法通常也面臨『起始值』的選擇問題。有時候起始值的選擇是有根據的, 但有時候卻是盲目的!試試看不同起始值, 是否會得到不同的答案 (根)?

6. 通常我們會從圖形去取得適當的起始值, 不過實際的狀況可能連圖都畫不出來。無法先透過圖形的判斷取得起始值。想想看有沒有一些簡單有效的方法, 可以讓程式自動找尋好的起始值? 譬如

$$
\begin{array}{ll}
while \; 1 & \%\text{開始無限迴圈迴圈} \\
\qquad ab = unifrnd(N_1, N_2, 1, 2) & \%\text{在}(N_1, N_2)\text{的範圍內隨意找兩個值} \\
\qquad a = ab(1); b = ab(2); & \%\text{ 一個放在變數 } a \text{ 另一個放在 } b \\
\qquad fa = f(a); & \%\text{計算函數值} \\
\qquad fb = f(b); & \%\text{計算函數值} \\
\qquad if \;\; fa * fb < 0 & \%\text{判斷 a 與 b 之間是否存在一個根} \\
\qquad\quad break; & \%\text{跳出迴圈} \\
\qquad end & \\
\qquad\quad \vdots & \\
end & \\
x_k = (a+b)/2; & \%\text{初始值}
\end{array}
$$

圖 3展示這樣的概念。這個方式保證可以找到一個不太離譜的初始值 (最壞的情況就是 a 與 b 距離很遠, 且其中一個離根很近), 但也有機會落在根的附近。這個方式如果能進一步改良, 讓 $(a+b)/2$ 不會離根太遠, 甚至還可以逐步「夾擊」進逼到根的位置。想想看、試試看有哪些作法。

7. 當方程式有重根時, 如何把所有的根都找出來呢?

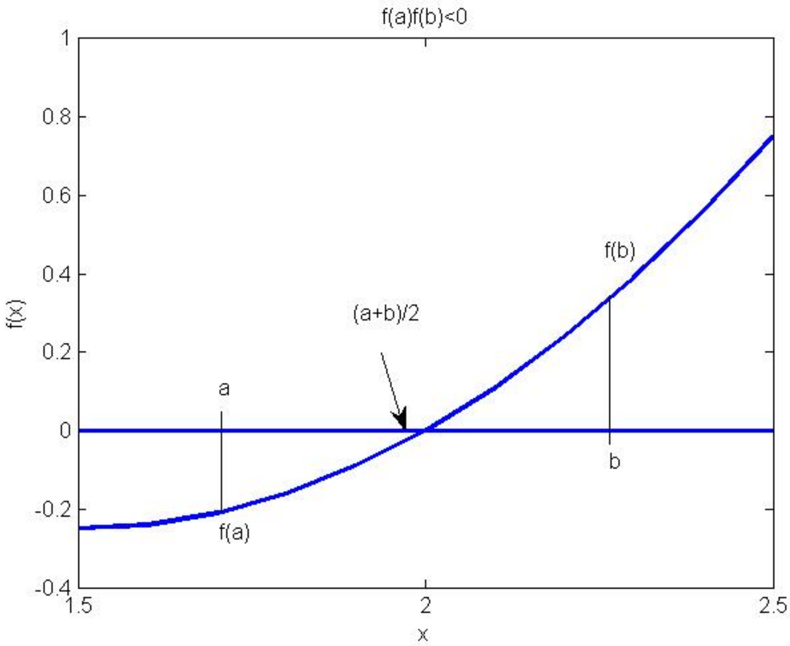

圖 3: 自動搜尋初始值的概念

8. MATLAB 除提供計算多項式方程式根的指令 roots 外,fzero 是另一個方式, 其做法與之前學過的積分指令 quad 類似。[1] 以下舉幾個使用方式, 計算方程式 (1) 的根。

[1]fzero 或 quad 等指令在 MATLAB 7.x 版有不同於 6.x 版的作法, 更具彈性, 本範例以 7.x 版爲主

```
方式1                              %適用6.x 以上
fzero('x^2-3*x+2',2)               %第2個參數代表計算在2附近的根

方式2                              %適用7.x 以上
fzero(@(x)x^2-3*x+2,2);

方式3                              %適用7.x 以上
f=@(x)x^2-3*x+2;                   % 使用匿名函數 (anonymous function)
fzero(f,2);

方式4                              %適用7.x 以上
f=@(x)polyval([1 -3 2],x);         % 使用匿名函數 (anonymous function)
fzero(f,2);
```

以上幾個方式適用在函數比較單純, 可以一行指令表示者。若函數複雜, 最好還是以副程式 (函數) 的方式比較周延, 譬如以下的範例

```
fzero(@(x)myfun(x),2)


其中呼叫的副程式如下
function  y=myfun(x)
          y=x^2-3*x+2;
```

MATLAB 7.x 以上還可以允許輸入額外的參數到副程式中, 讓計算根的功能更具彈性, 譬如

```
p=[1 -3 2]                         %多項式的係數
fzero(@(x)myfun(x,p),2)


其中的呼叫的副程式如下
function  y=myfun(x,p)
          y=p(1)*x^2+p(2)*x+p(3);
```

以上的範例說明當變更變數 p 時, 可以計算任何二項式方程式的根。這個程式當然可以被改寫, 擴充爲可以針對任何多項式函數。此外,MATLAB 7.x 提供一種 nested function 的概念, 將一個副程式隱含在另一個副程式裡面, 避免使用過多的程式, 這個功能類似前一個範例使用的匿名函數。看看這個新的作法:

```
p = [1 -3 2]                 %多項式的係數
findzero(p,2)                % 呼叫副程式 findzero 計算2附近的根

副程式 findzero
function y = findzero(p,x0)
        y = fzero(@(x)myfun(x,p),x0);

        function f = myfun(x,p)
                f = p(1)*x^2 + p(2)*x + p(3);
        end
end
```

依這個方式, 不難寫出與 roots 相同功能的副程式, 有興趣者不妨一試。

9. 解方程式的方法很多, 除本單元介紹的之外, 其他如「割線法、」「定點法」... 等, 都值得去找來研究, 以增加自己程式寫作的能力。其中的「定點法」很有趣, 將 $f(x) = 0$ 改寫成 $g(x) = x$, 以迴圈的方式迭代, 當近似根在 $|g'(x)| < 1$ 範圍內均可以收斂。有趣的地方在 $g(x) = x$ 可以有許多種寫法, 有些滿足收斂條件, 有些則否。適當的選擇往往可以讓很棘手的問題變得簡單, 扮演小兵力大功的角色。

   以解方程式 $f(x) = x - e^{-x/2} = 0$ 爲例, 定點法將之改寫爲

$$x = e^{-x/2} = g(x)$$

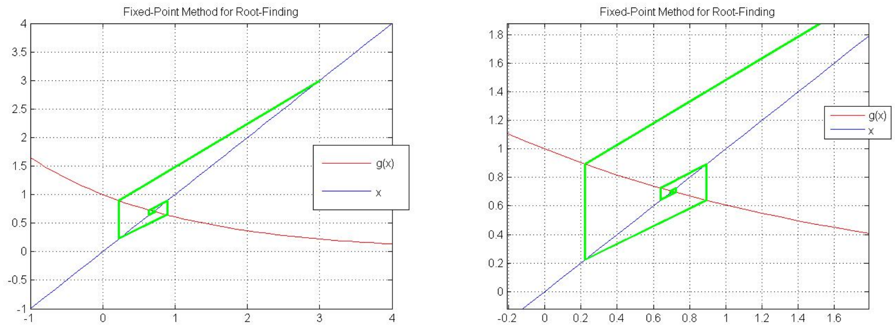

圖 4: 定點法計算根的過程

其幾何意義如圖4的斜線 $(y=x)$ 與曲線 $(y=g(x))$, 其根為兩線的交點。利用定點法的「定點意義,」以迭代的方式

$$x_{k+1}=g(x_k)$$

當 $|g'(x^o)|<1$ 時, x_{k+1} 將收斂到方程式根 x^o。圖4左圖展示迭代的過程, 確實逐步往交點逼近, 右圖放大交點附近區域, 更容易看清楚收斂的情形。

10. 牛頓法 (2) 並不保證收斂, 在某些情況下, 也會出狀況; 譬如, 計算 $f(x)=x^3-5x$ 的根, 若選擇初始值為 $x=1$, 將發生震盪現象, 不論迴圈進行多少遍,x 值始終在 1 及 -1 來回交替。不妨試試看, 觀察這個有趣的現象。

4 作業

1. 從圖1牛頓法的步驟, 推導出式 (2) 的方向,$d_k=-\frac{f(x_k)}{f'(x_k)}$。

2. 試寫出牛頓法計算方程式根的演算法步驟。

3. 方程式

$$f(x)=-x+e^{-x/2}=0$$

- 在適當的範圍內畫出圖形。
- 應用牛頓法計算方程式的根。
- 將程式設計成可以秀出求解過程的演進。(即標示出$x_0,x_1,x_2,\cdots$)。

4. 試寫出勘根法計算方程式根的演算法步驟。

5. 利用圖3勘根法的概念, 寫程式尋找上一題方程式的解。

6. 利用 MATLAB 指令$fzero$求上述方程式的解。

7. 利用定點法計算上述方程式的解。

8. 利用 MATLAB 的 fzero 與 quad 求出下列方程式的解 x。

$$0.9 = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{\theta^2}{2}} d\theta$$

注意:$-\infty$ 在數值計算上並不可行, 可以試著以一個相當的數字代替, 譬如 -10。

9. 計算圖 5 中兩個 β 分配的交點。

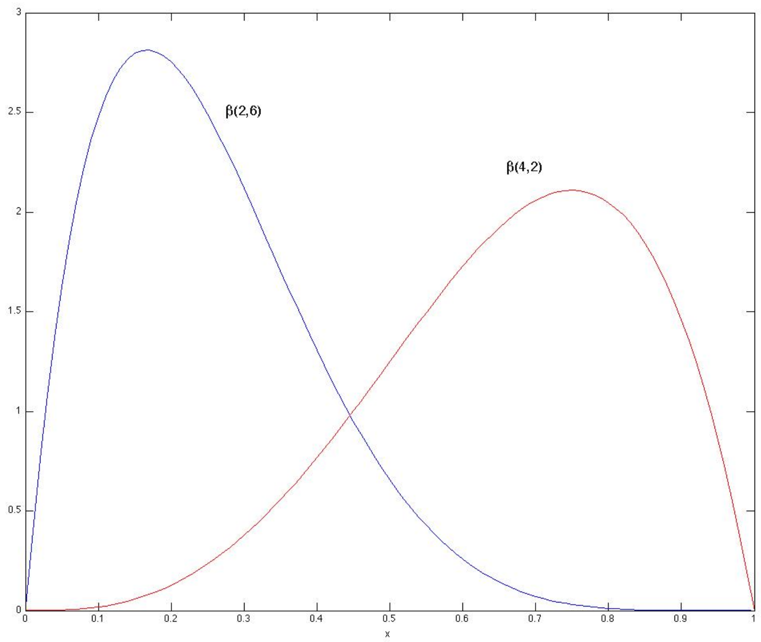

圖 5: 兩個 β 分配:$\beta(2,6)$ 與 $\beta(4,2)$.