淺談數值線性代數

賴玉玲

一. 介紹

什麼是數值線性代數呢? 從線性代數這一學科中, 我們學到了琳瑯滿目的方法來解決線性代數問題, 這些方法都有一個共同的特性, 計算量隨著矩陣維數 (dimension) 的增大而可能增加到非人力可以勝任, 以至於必須訴諸電子計算機的幫助。由於電腦的記憶體, 運算速度及算術誤差等因素, 我們不能單純的利用電腦來執行我們在線性代數學科中所學的方法, 因此我們得針對電腦的特性, 並藉助發展了好幾世紀的線性代數理論, 設計出快速有效及準確的演算法 (algorithms), 這也就是數值線性代數這一學科的主要研究課題。

衆所周知, 數學算子中最簡單的就是線性算子, 它也是在應用數學領域中最常見的模型, 所以其衍生出的線性代數問題值得探討。大體來說, 數值線性代數比較常處理其中的兩大基本問題

(1) 找線性系統 $Ax = b$ 的解

(2) 特徵值問題 (找特徵值 λ 及特徵向量 x 使得 $Ax = \lambda x$)

根據估計約有 75% 的應用數學問題會衍生出以上的第一個問題, 其中又以偏微分方程的數值解爲最常見, 其次爲函數的線性插值 (linear interpolation), 線性逼近 (linear approximation) 及最小方差 (least square) 等問題, 所以在此我們將對第一個問題做較深入的探討, 希望藉此可引導讀者們進入數值線性代數領域。

求線性系統解的數值方法衆多, 大致可粗略的分成兩大類 (1) 直接法 (2) 疊代法, 我們將於下列兩節分別詳述之。

二. 直接法

直接法的特性在於如果不考慮電腦計算的誤差, 則在有限步驟內就可得到正確解, 其中最常被採用的即是大家所熟知的高斯消去法 (Gaussian elimination) 及其變形 (variants), 其主要原因是它的觀念淺顯易懂及它的演算法可輕易地翻譯成電腦程式。

爲了詳細描述它的觀念, 我們考慮下列的線性系統

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \quad \vdots \qquad\qquad\qquad\qquad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \tag{1}$$

假如 $a_{11} \neq 0$, 則第 (i) 式減去第 (1) 式乘以

$\frac{a_{i1}}{a_{11}}$, 我們得到下列維數 $(n-1)$ 的線性系統

$$\begin{cases} a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 + \cdots + a_{2n}^{(2)}x_n = b_2^{(2)} \\ a_{32}^{(2)}x_2 + a_{33}^{(2)}x_3 + \cdots + a_{3n}^{(2)}x_n = b_3^{(2)} \\ \quad\vdots \qquad\qquad\qquad\qquad \vdots \\ a_{n2}^{(2)}x_2 + a_{n3}^{(2)}x_3 + \cdots + a_{nn}^{(2)}x_n = b_n^{(2)} \end{cases} \tag{2}$$

其中 $a_{ij}^{(2)} = a_{ij} - \frac{a_{i1}}{a_{11}}a_{1j}$, $b_i^{(2)} = b_i - \frac{a_{i1}}{a_{11}}b_1$, $i, j = 2, \ldots n$。很顯然地, 變數 x_1 被消去了, 整個問題被簡化成 $(n-1)$ 的線性系統, 此時如果 $a_{22}^{(2)} \neq 0$, 則可重覆上述的過程, 更甚者, 如果 $a_{kk}^{(k)} \neq 0$, $k = 2, 3, \ldots (n-1)$, 則可依序消法變數 $x_2, x_3, \ldots x_{n-1}$ 直到只剩下一個式子及一個變數。如果把每一步驟中的第一個式子收集起來, 我們得到下列三角 (triangular) 系統

$$\begin{cases} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ \qquad\qquad a_{22}^{(2)}x_2 + \cdots + a_{2n}^{(2)}x_n = b_2^{(2)} \\ \qquad\qquad\quad \ddots \qquad\qquad \vdots \\ \qquad\qquad\qquad\qquad a_{nn}^{(n)}x_n = b_n^{(n)} \end{cases} \tag{3}$$

其中爲了符號的一致性, 以 $a_{1j}^{(1)}$ 及 $b_1^{(1)}$ 分別代表 a_{ij} 及 b_1。以上的過程就是所謂的高斯消去法, 我們以下列的演算法做一個總結

```
for k = 1 to n − 1
    for i = k + 1 to n
        c := a_ik / a_kk
        b_i := b_i − c b_k
        for j = k + 1 to n
            a_ij := a_ij − c a_kj
        end
    end
end
```

利用高斯消去法, 原來的線性系統 $Ax = b$ 被簡化成三角線性系統 $Ux = \tilde{b}$, 其中 U 是一上三角矩陣。新的問題 $Ux = \tilde{b}$ 十分易解, 從方程組 (3) 可輕易觀察出 $x_n = b_n^{(n)}/a_{nn}^{(n)}$, 一旦 x_n 值知道了, 代入第 $(n-1)$ 式就可得到 x_{n-1} 的值, 依此類推, 可陸續得到 $x_{n-2}, x_{n-3}, \ldots x_1$ 的值, 此法被稱爲回置代入法 (back substitution)。

高斯消去法雖是一淺顯易懂的方法, 但是從它的推演過程中, 我們不難發現它並不是一個進行無礙的方法, 因爲途中若某個值 $a_{kk}^{(k)} = 0$ 時 ($a_{kk}^{(k)}$ 被稱爲 pivot), 則高斯消去法就被迫中斷。利用線性代數的理論, 不考慮算術誤差的情況下, 可推導出某些矩陣可以高枕無憂的使用高斯消去法, 然而由於無法避免電腦計算的誤差, 所以對那些理論上可使用高斯消去法的矩陣還是有可能遇到 pivot 爲零的情況而被迫中斷高斯消去法的使用, 因此我們不詳述那些矩陣理論上可無礙的使用高斯消去法, 然而我們將看是否有補救之道呢? 首先我們必須注意到高斯消去法中斷並不表示此線性系統無解, 例如考慮

$$\begin{cases} x_1 + x_2 + x_3 = 6 \\ x_1 + x_2 + 2x_3 = 9 \\ x_1 - x_2 + 4x_3 = 11 \end{cases} \tag{4}$$

其解爲 $x_1 = 1$, $x_2 = 2$ 及 $x_3 = 3$。應用一步高斯消去法後, 得到

$$\begin{cases} 0x_2 + x_3 = 3 \\ -2x_2 + 3x_3 = 5 \end{cases} \tag{5}$$

很顯然地, 因爲 $a_{22}^{(2)} = 0$, 無法再使用高斯消去法了。然而我們可以藉著交換行、交換列或

交換行及列來避免 pivot 爲零的情況, 這種方式通常被稱爲 pivoting。在更深入探討此法前, 首先我們要提醒讀者注意以下兩項事實: (1) 兩列互換的同時, 右邊向量 b 的兩個相對應元素也要互換, (2) 兩行互換時不要忘了相對應的兩個變數也要互換。

假如我們處理的矩陣是可逆的, 理論上高斯消去法配合 pivoting 的技巧絕對可以進行無礙地把解找到, 但實際上, 由於計算的誤差, 我們無可避免地去考慮如何 pivoting 才能眞正如理論上所說順暢地把解求出, 更甚者, 如何 pivoting 才能增加解的準確性, 有許多不同的 pivoting 技巧可以達到上述的目的, 在此我們列舉兩個最常被使用的

1. Complete pivoting: 在所處理矩陣的所有元素中, 找一個絕對值最大的元素, 再利用行、列交換把它搬到 pivot 的位置。以數學語言來說, 如果 r 及 s 使得 $|a_{rs}^{(k)}| = \max_{i,j=k\ldots n} |a_{ij}^{(k)}|$, 則第 r 列與 s 行分別與第 k 列及行互換。
2. Partial pivoting: 在所處理矩陣中的第一行元素中, 找一個絕對值最大的元素, 利用列交換把它搬到 pivot 的位置, 換言之, 如果 r 使得 $|a_{rk}^{(k)}| = \max_{i=k\ldots n} |a_{ik}^{(k)}|$, 則第 r 列與第 k 列互換。

因在 partial pivoting 的技巧中, 列交換比較常被使用, 所以我們對其略做了些描述, 實際上, 上述的觀念也可以考慮行交換來代替之。爲了使讀者更易了解 pivoting 的技巧, 我們考慮前述的例子, 利用 complete pivoting, 我們得到

$$\begin{cases} 3x_3 - 2x_2 = 5 \\ x_3 + 0x_2 = -1 \end{cases} \tag{6}$$

雖然線性系統不變, 但是處理的矩陣 A_2, 變成了 $\begin{bmatrix} 3 & -2 \\ 1 & 0 \end{bmatrix}$ 以 3 爲 pivot, 相對地, 如果考慮 partial pivoting, 則矩陣 A_2 變成了 $\begin{bmatrix} -2 & 3 \\ 0 & 1 \end{bmatrix}$ 以 -2 爲 pivot。

理論上高斯消去法配合 complete pivoting 所得到的解較配合 partial pivoting 而得的解來得精確, 但在衆多的例子中, 數值結果顯示配合 partial pivoting 的技巧所得到的解跟實際的解差異並不大, 而且從 complete pivoting 及 partial pivoting 的定義中, 可輕易看出 partial pivoting 所需的工作量遠少於 complete pivoting 的工作量, 所以一般的情況下, 除非想得到非常精確的解, partial pivoting 技巧還是較常被建議的。

三. 疊代法

大型電路 (large circuits) 分析、氣象學、地震學, 及太空科技等推演出的偏微分方程, 經離散化後所導出的線性系統, 一般來說都有成千上萬的未知數 (亦即線性系統的維數), 此時如果直接採用上一節的直接法是一很不明智的做法, 因爲如果這個線性系統的維數是 n 且沒有特殊的結構, 則需有足夠的記憶體來儲存約 n^2 個實數及約需 n^3 的運算才能得到線性系統的解, 舉例來說, 如果 $n = 10000$ 則約需 4 天的運算時間及 800 mega byte(以一個實數需 8 bytes 來計算)

的記憶體。到底要如何解這類的問題呢? 我們之所以用"這類"的字眼, 在於它們通常都有一共同的特性——稀疏 (sparsity), 亦即這類的線性系統的係數所構成的矩陣 A 中大部分元素都是零, 因此我們可以避免乘零或加減零的運算來減少運算時間及只存非零的元素以減少所需的記憶體量。我們可以直接法配合上述的技巧來解此類的線性系統, 但是這需要非常小心及有技巧的處理矩陣 A 的元素, 因爲高斯消去法會把 A 中原來爲零的元素變成非零元素 (亦即 fill-in 的問題), 所以必須非常技巧的利用 pivoting 來減少 fill-in 的量; 另外由於 fill-in 的問題, 而很難預估記憶體的需要量, 因此直接法通常並不被建議來處理大型且稀疏的線性系統, 也因此而發展出可輕易地避免無謂的乘零及加零的運算及使用最少記憶體量的迭代法。

根據一般說法, 約 170 年前 Carl Friedrich Gauss 是最早使用疊代法來解線性系統的人, 那時候他在處理最小方差問題時, 發現所導出的線性系統太大而無法以直接法中的高斯消法去求解, 他那時候所用的方法就是現今大家所熟悉的 Blockwise Gauss-Seidel 疊代。此後不久, 約1840 Carl Gustav Jacobi 也提出一非常類似的方法, 後來被稱爲 Jacobi 疊代。由於電腦的快速發展, 人類可以處理更大的線性系統後, 這時才有人發現及證明上述的兩個方法的收斂速度太慢, 所以才開始思考如何增加它的收斂速度, 其中最有名的就是1950, D. Young 所提出的 SOR(successive over relaxation) 疊代, 從那時候起, 陸陸續續有人從事這方面的研究, 提出更多有效且快速的方法。

對一線性系統 $Ax = b$, 首先選取任意初始向量 (initial vector) $x^{(0)}$, 利用 $x^{(0)}$ 推得 $x^{(1)}$, 再以 $x^{(1)}$ 推出 $x^{(2)}$, 依此類推而得到一向量數列 $\{x^{(m)}\}_{m=0}^{\infty}$, 這就是迭代法的主要觀念。隨著從 $x^{(m)}$ 推演得 $x^{(m+1)}$ 的方式的不同, 而有形形色色不同的迭代法, 如前述的 Jacobi 疊代的方式爲

$$x_i^{(m+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(m)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(m)})/a_{ii} \quad i = 1, 2, \ldots, n \tag{7}$$

Gauss-Seidel 疊代的方式

$$x_i^{(m+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(m)}\Big)/a_{ii} \quad i = 1, 2, \ldots, n \tag{8}$$

這兩個迭代方法的架構大同小異, 它們都源於下列的觀念: 每一矩陣 A 都可以唯一分解成 $A = D - L - U$ 其中 D, L, U 分別爲對角, 嚴格下三角及嚴格上三角矩陣, 且 $Ax = b$, $Dx = b + (L + U)x$ 及 $(D - L)x = b + Ux$ 三個線性系統是相等的 (equivalent), 所以把左邊及右邊向量 x 分別以 $x^{(m+1)}$ 及 $x^{(m)}$ 代替之, 則可分別得到上述的 Jacobi 迭代

$$Dx^{(m+1)} = b + (L + U)x^{(m)} \tag{9}$$

及 Gauss-Seidel 迭代

$$(D - L)x^{(m+1)} = b + Ux^{(m)} \tag{10}$$

從上述的式子, 我們可輕易地發現, 如果矩陣 D 是奇異的 (singular) 則這兩個迭代法都無法執行, 然而以上的觀念可推廣爲改考慮矩陣 A 的任意分解 (splitting) $A = M - N$ 而得到下列疊代式

$$Mx^{(m+1)} = Nx^{(m)} + b \qquad (11)$$

爲了能有效及輕易地執行上述迭代式, 通常我們會要求 M 是可逆的, 且它的反矩陣 M^{-1} 不難求得(例如: 如果 D 是可逆的, 則可取 $M = D$ 或 $M = D - L$ 亦即得到所謂的 Jacobi 及 Gauss-seidel 迭代式)。在 M 是可逆的情況下, 假設 x^* 是 $Ax = b$ 的正確解則由 $Mx^* = Nx^* + b$ 我們得到

$$\begin{aligned} & x^{(m+1)} - x^* \\ = \; & M^{-1}N(x^{(m)} - x^*) \qquad (12) \\ = \; & (M^{-1}N)^2(x^{(m-1)} - x^*) \\ = \; & \cdots \\ = \; & (M^{-1}N)^{m+1}(x^{(0)} - x^*) \end{aligned}$$

從式子 (12) 及利用線性代數的定理, 我們可歸納出, 如果 $M^{-1}N$ 的 spectral 半徑 $\rho(M^{-1}N) < 1$, 則 $\{x^{(m)}\}_{m=0}^{\infty}$ 是一收斂數列, 且其收斂向量 $\tilde{x} = x^*$, 如果 $\rho(M^{-1}N) > 1$ 則 $\{x^{(m)}\}_{m=0}^{\infty}$ 必爲一發散數列, 如果 $\rho(M^{-1}N) = 1$ 則 $\{x^{(m)}\}_{m=0}^{\infty}$ 可能收斂也可能發散。另外我們亦可以 $\rho(M^{-1}N)$ 的大小來粗略的估計其收斂速度, 前述所提 Jacobi 或 Gauss-Seidel 迭代的收斂太慢的原因就是它們相對應的迭代矩陣 $M^{-1}N$ 的 spectral 半徑 $\rho(D^{-1}(L+U))$ 及 $\rho((D-L)^{-1}U)$ 不夠小。爲了使其收斂快些, 我們引進參數 ω, 而改考慮線性系統 $\omega Ax = \omega b$, 且考慮分解 $M = D - \omega L$ 及 $N = (1-\omega)D + \omega U$ 而得到所謂的 SOR 迭代式

$$(D-\omega L)x^{(m+1)} = \omega b + [(1-\omega)D + \omega U]x^{(m)} \qquad (13)$$

亦即

$$\begin{aligned} & x_i^{(m+1)} \\ = \; & x_i^{(m)} - \omega\Big(\sum_{j=1}^{i-1} a_{ij}x_j^{(m+1)} + \sum_{j=i}^{n} a_{ij}x_j^{(m)} - b_i\Big)/a_{ii} \\ & i = 1, 2, \cdots, n \qquad (14) \end{aligned}$$

因爲 ω 是一任意實數, 且 $\omega = 1$ 即所謂的 Gauss-Seidel 迭代, 所以一定存在 ω 使得 $\rho((D-\omega L)^{-1}((1-\omega)D+\omega U)) \le \rho((D-L)^{-1}U)$, 其中使得 $\rho((D-\omega L)^{-1}((1-\omega)D+\omega U))$ 爲最小的 ω, 我們稱之爲最佳的 ω, 以 ω_{opt} 代表之。找 ω_{opt} 並非一簡單的工作, 在1950及1960年代有不少的學者從事這方面的研究, 於某些特定類型的矩陣找到了 ω_{opt}, 我們不在此多作描述, 有興趣的讀者可參照後面所列的參考資料。

四. 結語

進入數值線性代數的領域不難, 只要具備線性代數的基本概念及些許的程式設計能力即可, 一旦你登堂入室後, 你便可發現其內部的奧妙, 它並不如想像中的容易, 因爲還是有很多看似淺顯的問題尚未被解決出, 留待聰明的讀者們去解決。

在此我們僅介紹兩個淺顯易懂的方法給讀者們, 希望能引起讀者的興趣而動手寫些程式去執行上述的兩大演算法。唯有動手做問題, 您才會發現其實還有很多的問題我們在此尚未探討的, 諸如直接法中的捨入誤差對解的影響有多大, 迭代法中須如何判斷數列 $\{x^{(m)}\}_{m=0}^{\infty}$ 是否收斂到正確解了, 及怎樣的判斷得到的解最接近正確解等等的問題。希望做更深入的探討的讀者, 不妨參考我們所列的參考資料。

參考資料

1. G. Dahlquist and A. Bjorck, "Numerical Methods", Prentice-Hall, Englewood Cliffs, NJ, 1974.
2. G. H. Golub and C. F. Van Loan, "Matrix Computations", 3rd ed., Johns Hopkins University Press, Baltimore, 1989.
3. R. S. Varga, "Matrix Iterative Analysis", Prentice-Hall, Englewood Cliffs, NJ, 1962.
4. D. M. Young, "Iterative Solution of Large Linear Systems", Academic Press, New York, 1971.

—本文作者任教於中正大學應用數學系—