▶例 5.4 亂數產生器

在眞實的世界裡，我們很難進行一個完美的量測，因爲實際的量測總會摻雜著量測的雜訊。對控制系統設計來說，譬如飛機、煉油廠、核子反應爐等的操控系統，量測雜訊是控制系統設計中的重要考慮因素。一個良好的工程設計，必須考慮量測誤差的問題，避免這些量測雜訊導致系統的不當操作（不能有飛機失事、煉油廠爆炸或是核子反應爐熔化的狀況發生!）。

大部分的工程設計，會在系統建造之前，先用數值模擬去測試系統的運作情形。這些模擬包括了建立系統行爲的數學模型，並提供這些數學模型模擬的輸入資料。如果這些數學模型對模擬的輸入資料反應正確，我們才能有合理的自信，實際設計的系統也會對眞實的輸入資料做出同樣正確的反應。

爲了模擬系統眞實的操作環境，提供給模型的輸入資料必須含有模擬的量測雜訊，而這些模擬的輸入資料只是把一組亂數加在理想的輸入資料上。這種由亂數產生器（random number generator）所產生的亂數，常常用來模擬實際的量測雜訊。

當每次呼叫亂數產生器的函式時，亂數產生器都會傳回不同。而且明顯爲亂數的值。這些亂數事實上是從某種既定性的（deterministic）演算法所計算而來，它們只是看起來像是隨機出現而已[4]。然而只要用來產生亂數的演算法夠複雜，對模擬的使用來說，這些數字便能算是隨機出現的亂數。

一個簡單的亂數產生器演算法[5]是藉由模數函式（modulo function）處理大數字時所引起的不可預測性來產生亂數。記得在第四章學過的模數函式 mod 會傳回兩個整數相除的餘數，假如第一個整數來自於一個未知的級數，而第二個整數也是未知，則產生的餘數序列看起來就像是隨機亂數。

舉例而言，（5-6）式藉由在一個線性方程式裡乘上一個級數的前一個數值，然後取其模數，因而定義出此級數的下一個數值。

---

4 因為這個緣故，有些人稱這些函式為虛擬亂數 (pseudorandom number) 產生器。

5 這個演算法改寫自 1986 年劍橋大學出版，由 Press、Flannery、Teukolsky 和 Vetterling 所著的 *Numerical Recipes: The Art of Scientific Programming* 中的第七章。

$$n_{i+1} = \text{mod}(8121\, n_i + 28411, 134456) \qquad (5\text{-}6)$$

假設 n_i 是一個非負整數，因爲模數函式的關係，n_{i+1} 將會是一個介於 0 至 134455 之間的數字。接下來，n_{i+1} 可以輸入到方程式裡，產生 n_{i+2} 的數字，而 n_{i+2} 也是介於 0 至 134455。這過程可以一直重複下去，以產生一連串介於 [0,134455] 範圍內的數字。如果我們事先不知道 8121、28411 以及 134456 這些數字，我們便不可能去猜出 n 值產生的順序。 此外，對產生的結果來說，每個產生的數字出現在序列的機率是相等的（或是均勻分佈）。因爲這些性質，（5-6） 式可當作一個產生均勻分佈亂數的簡易基本演算法則。

我們現在將使用（5-6）式來設計一個亂數產生器，以輸出在 [0.0,1.0）範圍[6]的實數。

解答

我們將編寫一個函式，每次呼叫函式時，都會產生一個在 $0 \le ran < 1.0$ 範圍的實數。這個亂數將由下列方程式產生：

$$ran_i = \frac{n_i}{134456} \qquad (5\text{-}7)$$

其中 n_i 是由（5-6）式所產生介於 0 到 134455 間的數字。

這個由方程式（5-6）與（5-7）所產生的特定序列，與初始值 n_0（稱爲種子）的設定有很大的關係。我們必須提供使用者一個設定 n_0 的方法，使得每次執行程式時，都會產生不同的序列。

1. 宣告問題。

根據方程式（5-6）與（5-7），編寫一個函式 `random0`，使其能夠產生並傳回包含一個或多個均勻分佈亂數序列的陣列 `ran`，而且 $0 \le$ `ran` < 1.0。這個函式必須有一個或兩個輸入引數 `(m,n)`，用以設定傳回的陣列大小。如果只有一個輸入引數 `m`，函式會產生一個大小爲 `m` × `m` 的方形陣列。如果有兩個輸入引數，則函式會產生一個大小爲 `m` × `n` 的陣列。亂數種子的初始值 n_0 將由呼叫函式 `seed` 來指定。

6　[0.0, 1.0) 代表亂數值介於 0.0 與 1.0 之間，可以等於 0.0，但不等於 1.0。

2. **定義輸入和輸出。**

這問題中有兩個函式：seed 及 random0。函式 seed 的輸入值是一個整數，用來當作亂數序列的初始值，此函式不需要輸出。函式 random0 的輸入值是一個或兩個整數，用來指定產生的亂數陣列大小。假設只有提供一個引數 m，函式會產生大小爲 m × m 的方形陣列。假設提供了兩個引數 m 與 n，則函式會產生大小爲 n × m 的陣列。函式 random0 的輸出爲一個陣列，其元素值爲介於 [0.0,1.0）範圍內的亂數。

3. **設計演算法。**

函式 random0 的虛擬碼如下:

```
function ran = random0 (m, n)
Check for valid arguments
Set n <- m if not supplied
Create output array with "zeros" function
for ii = 1:number of rows
   for jj = 1:number of columns
      iseed <- mod (8121 * iseed + 28411, 134456)
      ran(ii,jj) <- iseed / 134456
   end
end
```

其中 iseed 爲一全域變數，其數值被存放在共用記憶體中。而函式 seed 的虛擬碼可簡單描述如下：

```
function seed (new_seed)
new_seed <- round(new_seed)
iseed <- abs(new_seed)
```

使用函式 round 是爲了避免使用者輸入一個非整數值，而使用絕對值函數是爲了避免使用者輸入一個負數值。使用者事先並不需要知道只有正整數才是合法的亂數種子。

全域變數 iseed 將被存放在共用記憶體中，也因此它能被這兩個函式所存取。

4. **把演算法變成 MATLAB 宣告式。**

函式 random0 顯示如下：

```
function ran = random0(m,n)
%RANDOM0 Generate uniform random numbers in [0,1)
% Function RANDOM0 generates an array of uniform
% random numbers in the range [0,1). The usage
% is:
%
% random0(m)    -- Generate an m x m array
% random0(m,n)  -- Generate an m x n array

% Define variables:
%   ii        -- Index variable
%   iseed     -- Random number seed (global)
%   jj        -- Index variable
%   m         -- Number of columns
%   msg       -- Error message
%   n         -- Number of rows
%   ran       -- Output array
%
% Record of revisions:
%     Date          Programmer         Description of change
%     ====          ==========         =====================
%   02/04/07      S. J. Chapman       Original code

% Declare global values
global iseed      % Seed for random number generator

% Check for a legal number of input arguments.
msg = nargchk(1,2,nargin);
error(msg);

% If the n argument is missing, set it to m.
if nargin < 2
   n = m;
end

% Initialize the output array
ran = zeros(m,n);

% Now calculate random values
for ii = 1:m
   for jj = 1:n
      iseed = mod(8121*iseed + 28411, 134456 );
      ran(ii,jj) = iseed / 134456;
   end
end
```

5. 測試程式。

根據統計學分析的結果，假設由這些函式所產生的亂數序列，確實均勻分佈在 $0 \le$ ran < 1.0 範圍內的數字，則其平均值應該接近

0.5，而標準差也應該接近 $\frac{1}{\sqrt{12}}$。

此外，假設我們在介於 0 和 1 的範圍，劃分成幾個相同大小的區間，則落入這些區間的亂數個數應大致相同。直方圖（**histogram**）可用來畫出這些區間的亂數個數。MATLAB 的函式 `hist` 將從輸入資料中產生並畫出一張直方圖，而我們將使用這個直方圖，來驗證由函式 `random0` 所產生的亂數分佈。

爲了測試這些函式的結果，我們將執行以下的測試條件：

1. 呼叫函式 `seed`，並設定 `new_seed` = 1024。
2. 呼叫函式 `random0(4)` 以查看輸出是否爲隨機。
3. 呼叫函式 `random0(4)` 以確定每次的結果都不一樣。
4. 再次呼叫函式 `seed`，並設定 `new_seed` = 1024。
5. 呼叫 `random0(4)` 以查看結果是否與（2）的結果相同。這是爲了驗證函式 `seed` 是否被正確地重新設定。
6. 呼叫 `random0(2,3)` 以驗證兩個輸入引數是否被正確地使用。
7. 呼叫 `random0(1,100000)` 並使用 MATLAB 函數 `mean` 和 `std`，來計算產生資料的平均值及標準差。比較這些結果是否接近 0.5 及 $\frac{1}{\sqrt{12}}$。
8. 從（7）所得的資料，產生一張直方圖，以驗證在每個區間是否落下大約相等數目的亂數。

我們將執行這些測試，以確認我們之前所預期的結果。

```
» seed(1024)
» random0(4)
ans =
    0.0598    1.0000    0.0905    0.2060
    0.2620    0.6432    0.6325    0.8392
    0.6278    0.5463    0.7551    0.4554
    0.3177    0.9105    0.1289    0.6230
» random0(4)
ans =
    0.2266    0.3858    0.5876    0.7880
    0.8415    0.9287    0.9855    0.1314
    0.0982    0.6585    0.0543    0.4256
    0.2387    0.7153    0.2606    0.8922
```

```
» seed(1024)
» random0(4)
ans =
    0.0598      1.0000      0.0905      0.2060
    0.2620      0.6432      0.6325      0.8392
    0.6278      0.5463      0.7551      0.4554
    0.3177      0.9105      0.1289      0.6230
» random0(2,3)
ans =
    0.2266      0.3858      0.5876
    0.7880      0.8415      0.9287
» arr = random0(1,100000);
» mean(arr)
ans =
    0.5001
» std(arr)
ans =
    0.2887
» hist(arr,10)
» title('\bfHistogram of the Output of random0');
» xlabel('Bin');
» ylabel('Count');
```

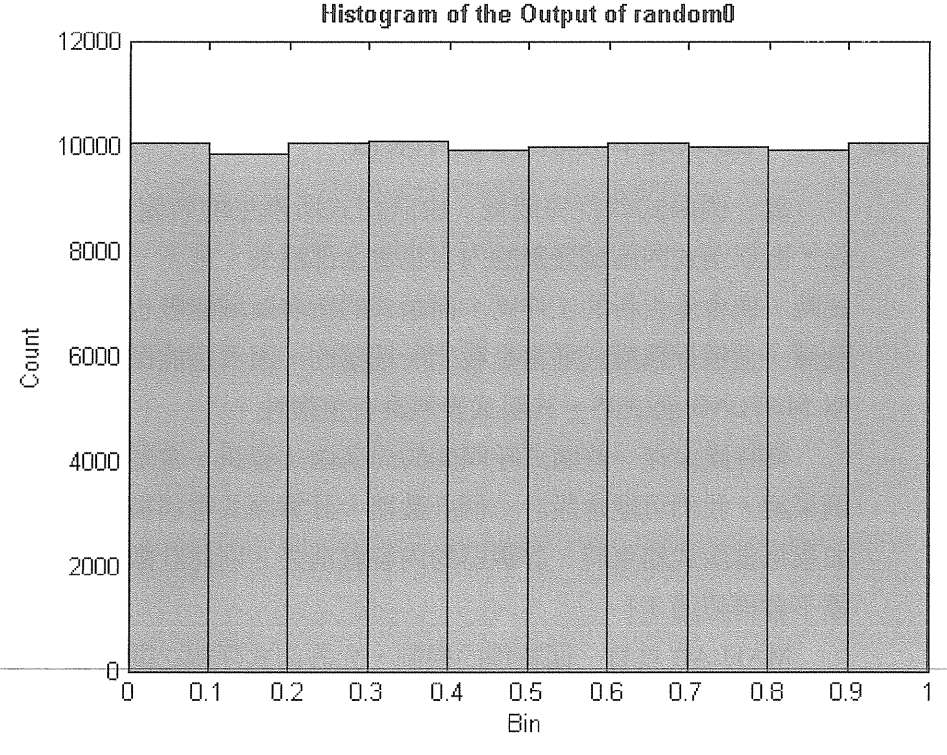

圖 5.5　函式 random0 **輸出資料的長方圖。**

這些測試結果看起來相當合理，所以函式似乎可以正常工作。這些亂數資料的平均值爲 0.5001，與理論值的 0.5000 非常接近，而亂數資料的標準差爲 0.2887，在顯示的位數上也與理論值一致。圖 5.5 的直方圖顯示了亂數的分佈也概略地平均分佈在各個區間內。

◀

MATLAB 包含兩個標準函式，可產生不同分佈的亂數值，它們是：

- `rand`——在 [0,1）範圍內產生均勻分佈的亂數。
- `randn`——產生常態分佈的亂數。
- 這兩個函式比起我們剛剛建立的簡單函式，不但更爲快速，而且更爲「隨機分佈」。假使您需要在程式中使用亂數，請使用其中一個函式。
- 函式 `rand` 和 `randn` 有下面幾個呼叫的方法：
- `rand()`——產生單一亂數值。
- `rand(n)`——產生一個 $n \times n$ 陣列的亂數值。
- `rand(m,n)`——產生一個 $m \times n$ 陣列的亂數值。

5.5 函式呼叫間的資料保存

當一個函式執行完成後，由該函式所產生的特定工作區會被消除，所以函式內區域變數的內容也會全部消失。當下一次呼叫這個函式時，便會產生新的工作區，而函式的所有區域變數也會回復變爲預設值。這種行爲模式通常是我們所期望的，因爲這能保證我們每次呼叫 MATLAB 函式時，其函式表現具有重複性。

有時候保存一些函式呼叫間的局部函式資訊，會是很有用的。舉例來說，我們可能會設計一個計數器，計算函式曾經被呼叫過幾次。如果每當函式結束時，計數器的內容就消失，則此計數器的計數將永遠不會再超過 1！

MATLAB 具有一個特別的機制，允許函式呼叫之間的區域變數可以保存下來。**續存記憶體**（**persistent memory**）是一種特別類型的記憶體，只允許在函式內存取，而在函數呼叫之間，其值保持不變。