目　　錄

第一章　緒論

第一節　研究動機與目的

近年來，隨著電腦的發展和普及，無論是在社會科學或自然科學的研究中，經常會需要利用電腦做為輔助的工具。而在統計的範疇中，當我們遇到困難的理論問題或是複雜的應用問題時，也會考慮電腦模擬的結果來做參考，此時亂數（random number）在其中扮演著極為重要的媒介角色。此外在抽樣（sampling）的問題上，亂數提供了抽樣的方法，使用精確的亂數值可以增加樣本的代表性。而在數值分析（numerical analysis）上，亂數可用來解決一些較複雜的數值分析問題，如將蒙地卡羅法（Monte Carlo method）應用在多維度的數值積分上。另外亂數亦廣泛應用在作業研究、品質管制及決策等學科上。

亂數的產生方式有著相當長的歷史，從早期利用丟銅板、擲骰子、或是抽籤、翻書等等各種人工的方式，都是頗為費時，同時這些方法的最大缺點就是無法重覆先前已產生過的亂數。另外，亂數表也是早期較為常用的方法之一，方法是在亂數表中任意決定一個起點，再利用橫向或縱向連續選取數字來做為隨機亂數，但亂數表需要逐一查詢，而且可提供的亂數有限，在實用上也是有其缺點。由於前面的方法都各有缺點，目前較為普遍的方式就是利用電腦來產生亂數。

一般而言，如果要利用電腦去得到亂數有兩種方法：方法一是由使用者自行設計撰寫程式來產生亂數，但是這對於某些不知道亂數結構的使用者，這些學習對他們來說就會屬於額外的負擔。方法二就是採用電腦套裝軟體所提供的亂數產生器（random number generator）來產生亂數。目前有越來越多的使用者利用電腦來做分析，而且在很多方面的應用上會依賴套裝軟體所提供的亂數值，所以在使用這些套裝軟體之前，先前的檢定是不可或缺的。而在一些統計分析上常使用的套裝軟體，包括 SAS

6.12、SPSS 8.0、EXCEL 97、S-PLUS 2000 及 MINITAB 12，在亂數產生器的部分都內設了多種機率分配的亂數產生器供使用者選擇（如附表 1 所示）。但基於實際上的考量，在此無法對各種機率分配的亂數做詳盡的介紹，同時在模擬的演算程序中，大部分的機率分配抽樣都可藉由轉換 U（0,1）亂數來產生，故將焦點放在均勻分配 U（0,1）的亂數產生器上。因此，本論文的主要目的在於：針對 SAS 6.12、SPSS 8.0、EXCEL 97、S-PLUS 2000 及 MINITAB 12 這五種統計分析上常使用的套裝軟體，分別介紹各個套裝軟體內設 U（0,1）亂數產生器的演算方法，以及實際上的操作使用，再利用亂數產生器所產生的亂數值來做統計分析比較。

第二節　研究方法

一開始先對亂數產生器做初步的說明，目前所有電腦的亂數產生器都是利用數學演算式來產生亂數數列，例如線性同餘法等等，因此若決定了某一個值之後，則接下來的亂數便可藉由數學演算式來推算，也就是由前一個亂數可以預測下一個亂數，並且會在一定的週期之後開始重覆，缺乏獨立的性質，所以這些亂數產生器所產生的亂數只能稱之為「假的隨機亂數」(pseudo-random number)。在本文中除非特別聲明，為方便故，文中亂數指的都是此種亂數。

至於什麼是理想的亂數產生器？Marsaglia 等人（1990）指出，一個好的亂數產生器在實用上通常需要具備以下幾項性質：

1. 長週期：週期長則不易發生重覆使用相同亂數數列的情形。
2. 良好的統計性質：希望亂數產生器所產生的亂數數列能夠滿足統計上獨立且均勻分配的性質。
3. 有效率的電腦執行：為了便於模擬，相同的亂數數列必須具有重覆執行（reproduce）的能力。演算的速度快，亂數的產生過程中佔用較少的時間和記憶體。

因此，若我們要知道一個亂數產生器是否合用，必須針對上述所需要的條件做一些評估，本文擬從週期長短、統計性質、電腦執行效率、三種不同觀點來評估這五種軟體在 U(0,1)亂數產生器的表現，觀察其是否能夠符合，而達到我們的要求。

第三節　論文架構

本文的架構共分為六章，各章的內容分別摘要如下：

第一章　緒論：說明本論文之研究動機與目的、研究方法以及論文架構。

第二章　介紹了三種亂數產生器常見的方法，包括了線性同餘法、移動暫存法以及組合法。

第三章　介紹各個套裝軟體內部的演算方法和各個軟體在實際上的操作使用。

第四章　針對亂數產生器在統計檢定方法的介紹。

第五章　從週期長短、統計、電腦執行效率三種不同觀點來評估這五種軟體在亂數產生器的表現。

第六章　結論與建議，將第五章的結果做歸納彙整，並提出未來的研究方向。

第二章　亂數產生器的方法

第一節　線性同餘產生器

線性同餘產生器（linear congruential generator；LCG）是目前使用最為廣泛的亂數產生器，這個方法是由 Lehmer（1951）首先提出，程序主要是連續重覆（2.1）的運算步驟，在給定了四個的參數值（m、a、c、x_0）之後，可得到整數數列$x_1.x_2,\ldots$。

$$x_i = ax_{i-1} + c \pmod{\mathrm{m}},\ i = 1,2,... \qquad (2.1)$$

上式的符號定義為： m是一個正整數的除數（modulus），a是一個非負整數的乘數（multiplier），c是非負整數的增量（increment），x_0為一個非負整數的起始值（seed or starting value），且$0 \le a, c, x_0 < m$。

然後再經由（2.2）的轉換：

$$u_i = \frac{x_i}{m} \qquad (2.2)$$

數列$u_1.u_2,\ldots$即是範圍介於 0 和 1 之間的亂數，在適當的選取參數值之下，就可以產生近似均勻分配 U（0,1）的亂數。

先前我們提到，由LCG產生的亂數並非真正地獨立，因為經過歸納[1]後可以發現，x_i數列具有以下特性：

$$x_i = \left(a^i x_0 + \frac{c(a^i - 1)}{a - 1} \right) \pmod{m}$$

因為每個x_i的值是完全的取決於 m、a、c、x_0，選用不同的組合，即可產生不同的亂數數列。這種藉由數學的演算式來產生亂數的方式，產生的數

---

[1] $x_1 = ax_0 + c - k_1 m$

$x_2 = ax_1 + c - k_2 m = a(ax_0 + c - k_1 m) + c - k_2 m = a^2 x_0 + c(a+1) - m(k_2 + ak_1)$

$x_3 = ax_2 + c - k_3 m = a^3 x_0 + c(a^2 + a) - m(ak_2 + a^2 k_1) + c - k_3 m = a^3 x_0 + c(a^2 + a + 1) - m(k_3 + ak_2 + a^2 k_1)$

$\vdots$

$x_i = a^i x_0 + c(a^{i-1} + a^{i-2} + \cdots + a + 1) - m(k_i + ak_{i-1} + \cdots + a^{i-1} k_1) = \left(a^i x_0 + \frac{c(a^i - 1)}{a - 1} \right) \pmod{\mathrm{m}}$

列 x_i 之間有相互的關係存在，但是若亂數能通過統計檢定的話，仍可視為真正的亂數來使用，因此我們要謹慎的選擇參數值，使其較符合隨機亂數的特性，顯示其為獨立且同分配的均勻分配。

同時，以此法所產生之亂數數列具有一重要性質，當 x_i 與之前曾出現的某一亂數值相同時，則 x_i 後之數列便會與開始反覆出現，形成一個循環現象（cycle）。循環的長度稱為週期（period），對任一 LCG 產生的亂數而言，假設週期為 $per(u_n)$，則 $per(u_n)$ 的範圍會小於等於 m，當 $per(u_n) = m$ 時稱為全週期（full period），為使 LCG 具有全週期的性質，m、a、c 之選定必須符合某些規則，Hull 和 Dobell（1962）提出下列定理，定理證明可參閱 Knuth（1981）：

定理 2.1 LCG 具有全週期的性質，若且唯若：

(1) m 和 c 互為質數。

(2) 假如質數 q 可整除 m，則 q 亦可整除 a-1。

(3) 假如 4 可整除 m，則 4 亦可整除 a-1。

以下舉例說明 LCG 的演算過程及全週期現象，在此我們選擇 m=16、a=5、c=1 和 x_0=3（符合定理 2.1），得到以下的亂數值（程式可參考附錄 1）：

表 2-1：$x_i = 5x_{i-1} + 1 \ (\text{mod}16), x_0 = 3$ 的亂數

i	x_i	u_i	i	x_i	u_i
0	3	---------	11	14	0.875
1	0	0	12	7	0.4375
2	1	0.0625	13	4	0.25
3	6	0.375	14	5	0.3125
4	15	0.9375	15	10	0.625
5	12	0.75	16	3	0.1875

6	13	0.8125	17	0	0
7	2	0.125	18	1	0.0625
8	11	0.6875	19	6	0.375
9	8	0.5	20	15	0.9375
10	9	0.5625	21	12	0.75

由上表可看出$u_{17}=u_1=0$、$u_{18}=u_2=0.0625\cdots$，所以這筆數列的週期$per(u_n)=m$=16，因此就具備了全週期的性質；實際上，為了要得到較長的週期，我們通常會使用較大的m值。

通常對於這個方法所產生的數列，會因為增量（c）設定值的不同而分為兩大類來討論，若$c>0$時稱為做混合型 LCG（mixed LCG），若$c=0$時則稱為乘法型 LCG（multiplicative LCG）。由於乘法型 LCG 的參數 c 為零，因此無法符合定理 2.1 (1)之要求，不具備全週期之性質，但只要慎選 m 和 a，週期的長度$per(u_n)$仍可達到$m-1$，定理的證明可參閱 Kunth（1981）。

定理 2.2 乘法型 LCG 週期的長度$per(u_n)=m-1$，若且唯若：

(1) m 是質數且 m 和X_0互為質數。

(2) a是除數m的質數元素（primitive element modulo m）[2]。

註解 [Cathy1]:
GF

符合定理 1.2 的乘法型 LCG 可稱為質除數乘法型 LCG（prime modulus multiplicative LCG；PMMLCG）。

到目前為止，關於 LCG 的發展已經有相當完整的研究，在參數值的選擇上也有許多文獻可供參考（L'Ecuyer 1988；Fishman 1990…），不過由於這個方法的運算關係式較為簡單，在經過了許多學者的驗證（Marsaglia 1968；Matteis 和 Pagnutti 1988；Niederreiter 1977 1991…）下，LCG 的確存在不少的缺失，例如長期相關性（long-range correlations）、以及其除數

[2] 假如a爲除數m之質數元素，則
$(a^i-1)/m\neq h$
$(a^{m-1}-1)/m=h \quad i=1,2,\ldots,m-2;h\in R$

及週期會受到電腦系統位元大小的限制等等的問題，所以其他型式的亂數產生器仍受到相當程度的重視，以下將介紹其他型式的亂數產生器。

第二節　移動暫存產生器

移動暫存產生器（shift-register generator）是由 Tausworthe 在 1965 年所發展而成的，因此也稱為 Tausworthe 產生器，移動暫存產生器的設計類似密碼傳遞方法，都是直接用位元（bit）值形成亂數，在給定了參數值 p、q 和最初幾個不全為 0 的 b_i 值為起始值之後，接下來重覆（2.3）的運算步驟，我們就可以得到二進位數位（binary digits）的 b_i 數列：

$$b_i = b_{i-p} + b_{i-q} \pmod 2 \text{；} 0 < p < q \;\; p, q \in R \qquad (2.3)$$

由於除數的值為 2，故（2.3）的運算也可以表示為「互斥或」（exclusive or；EOR）的運算[3]，如（2.4）所示：

$$b_i = b_{i-p} \text{ EOR } b_{i-q} \quad \text{；} 0 < p < q \;\; p, q \in R \qquad (2.4)$$

而在 b_i 數列產生之後，將 h 個 b_i 串連成一個二進位值，再把二進位值轉換為十進位值後，其值會介於 0 至 $2^h - 1$ 之間，因此除以 2^h 即可形成介於 0 和 1 之間的亂數且週期為 $(2^q - 1)/\gcd(h, 2^q - 1)$，所以如果選擇 $\gcd(h, 2^q - 1)$ 為 1 時，週期的長度就可達到 $(2^q - 1)$。移動暫存產生器的優點之一就是週期不會受到電腦位元大小的限制。

以下舉例說明移動暫存產生器的演算過程，在此我們選擇 p=1、q=4、$h = 4$ 和 $b_1 = b_2 = b_3 = b_4 = 1$，得到以下的亂數值（程式可參考附錄 2）：

表 2-2：$b_i = b_{i-1} + b_{i-4} \pmod 2, b_1 = b_2 = b_3 = b_4 = 1$ 的亂數

[3] EOR的運算定義爲：0 EOR 0 = 0, 0 EOR 1 = 1 , 1 EOR 0 = 1 , 1 EOR 1 = 0。

i	$b_{4i+1}b_{4i+2}b_{4i+3}b_{4i+4}$	x_i	u_i	i	$b_{4i+1}b_{4i+2}b_{4i+3}b_{4i+4}$	x_i	u_i
0	$1111_{(2)}$	$15_{(10)}$	---------	11	$0111_{(2)}$	$7_{(10)}$	0.435
1	$0101_{(2)}$	$5_{(10)}$	0.3125	12	$1010_{(2)}$	$10_{(10)}$	0.625
2	$1001_{(2)}$	$9_{(10)}$	0.5625	13	$1100_{(2)}$	$12_{(10)}$	0.75
3	$0001_{(2)}$	$1_{(10)}$	0.0625	14	$1000_{(2)}$	$8_{(10)}$	0.5
4	$1110_{(2)}$	$14_{(10)}$	0.875	15	$1111_{(2)}$	$15_{(10)}$	0.9375
5	$1011_{(2)}$	$11_{(10)}$	0.6875	16	$0101_{(2)}$	$5_{(10)}$	0.3125
6	$0010_{(2)}$	$2_{(10)}$	0.125	17	$1001_{(2)}$	$9_{(10)}$	0.5625
7	$0011_{(2)}$	$3_{(10)}$	0.1875	18	$0001_{(2)}$	$1_{(10)}$	0.0625
8	$1101_{(2)}$	$13_{(10)}$	0.8125	19	$1110_{(2)}$	$14_{(10)}$	0.875
9	$0110_{(2)}$	$6_{(10)}$	0.375	20	$1011_{(2)}$	$11_{(10)}$	0.6875
10	$0100_{(2)}$	$4_{(10)}$	0.25	21	$0010_{(2)}$	$2_{(10)}$	0.125

由上表可看出$u_{16}=u_1=0.3125$、$u_{17}=u_2=0.5625\cdots$，所以這筆數列的週期$per(u_n)$=15。

第三節　組合產生器

組合產生器是利用兩個或兩個以上的亂數產生器來組合成一組新的亂數產生器，Wichmann 和 Hill 是組合產生器的推廣者，例如 Wichmann 和 Hill（1982）提出組合

$$x_i = 171x_{i-1} \pmod{30269}$$
$$y_i = 172y_{i-1} \pmod{30307}$$
$$z_i = 170z_{i-1} \pmod{30323}$$
$$u_i = \frac{x_i}{30269} + \frac{y_i}{30307} + \frac{z_i}{30323} \pmod 1$$

此法主要是組合了三個線性同餘產生器，Zeisel（1986）指出此亂數產生器相當於線性同餘產生器中有著相當大的除數。近年來，L'Ecuyer

（1988）、L’Ecuyer 和 Tezuka（1991）對於合併數個線性同餘產生器以及合併數個移動暫存產生器也都做了相當深入的研究。組合產生器更成為了設計亂數產生器的主要趨勢，因為不僅使週期變長，且在實證上更可以改善一些統計性質。

第三章　常用統計套裝軟體內設U(0,1)亂數產生的方法

第一節　SAS 6.12

目前 SAS 6.12 內設 U(0,1)亂數產生器是參考 Fishman 和 Moore(1982)所提出的質除數乘法型線性同餘法，其中乘數設為 397,204,094，除數設為 $2^{31}-1$，先藉由下式產生 x_i：

$$x_i = 397{,}204{,}094\, x_{i-1} \pmod{2^{31}-1} \qquad i = 1,2,3,...$$

再將 x_i 除以 $2^{31}-1$ 而產生 U（0,1）的亂數，即

$$u_i = \frac{x_i}{2^{31}-1}$$

如果我們要藉由 SAS 6.12 來產生 U（0,1）的亂數值，可以利用『RANUNI（seed）』的函數來執行，當中的 seed 是此函數的隨機種子或起始值，其值可設為介於 $\pm 2^{31}-1$ 的任意整數值。依隨機種子的不同，將會有兩種不同的起始狀態：1、若隨機種子是 0 或負值時，此時的系統將會以當時執行的電腦時間作為起始值代入。2、若起始值隨機種子為正整數時，此時的系統將會以該隨機種子作為起始值代入。故若設定隨機種子為某固定的正整數時，即可得到重覆的亂數值。

實際上使用 SAS 6.12 的亂數函數來產生一筆 1000 個 U（0,1）的亂數值，可以執行以下的程式：

SAS 6.12程式：

```
data k1;
   do i=1 to 1000;
        obs=ranuni(0);
        output;
   end;
proc print;
run;
```

不過由於一般在實際模擬亂數值時，通常會需要大量的亂數值，因此以下提供執行一次就可以得到多個變數個數及多個亂數個數的程式，在此的變

數個數設為 100、亂數個數設為 1000：

SAS 6.12程式：

```
%let stream=100;
%let n=1000;

 data k;
   length vname $ 8;
   array obs{&n.} obs1-obs&n.;
   do i=1 to &stream.;
      vname="v"||compress(i);
      do j=1 to &n.;
         obs{j}=ranuni(0);
      end;
      output;
   end;
   drop i j;

proc transpose data=k out=k (drop=_name_);
   id vname;
proc print;
run;
```

第二節　SPSS 8.0

目前 SPSS 8.0 內設 U（0,1）亂數產生器也是參考 Fishman 和 Moore（1982）所提出的質除數乘法型線性同餘法，其中乘數設為 397,204,094，除數設為 $2^{31}-1$，先藉由下式產生 x_i

$$x_i = 397204094 x_{i-1} \pmod{2^{31}-1} \qquad i = 1,2,3,...$$

再將 x_i 除以 $2^{31}-1$ 而產生 U（0,1）的亂數，即

$$u_i = \frac{x_i}{2^{31}-1}$$

如果我們要藉由 SPSS 8.0 的視窗環境來產生 U（0,1）的亂數值，在點選轉換>計算之後，我們把對話盒內容設定如下圖，

就可以產生 1 組 U（0,1）的亂數值。此外，若想要得到重覆的亂數值，只要先在轉換>亂數種子之下設定亂數基值為某固定值，之後也可以再使用這個數值來產生相同的亂數值，不過若使用者沒有設亂數基值，每次啟動新的作業階段時，亂數種子就會自動重設成 2,000,000，即每次重新使用 SPSS 8.0 的亂數產生器都會得到我們就會得到以下的數列：0.1396, 0.4313, 0.6122, 0.2908, 0.1557,….。所以在使用亂數之前，一定要先把種子重設成特定值，否則就會得到完全一樣的亂數值。

另外，以下也提供利用 SPSS 8.0 語法編輯器執行一次就可以得到多個變數個數及多個亂數個數的程式，在此的變數個數設為 100、亂數個數設為 1000。

SPSS 8.0 語法編輯器：

```
data list free / a1 to a100.
begin data
end data.

save outfile=ttemp.

INPUT PROGRAM.
LOOP #I=1 TO 1000.
  COMPUTE SN=#I.
  END CASE.
END LOOP.
END FILE.
END INPUT PROGRAM.

add files file=*/ file=ttemp.
SET SEED RANDOM.
VECTOR A=a1 TO a100.
  loop #j=1 to 100.
    compute A(#j) = uniform(1).
  end loop.
execute.
```

第三節　EXCEL 97

目前 EXCEL 97 內設 U（0,1）亂數產生器是混合型線性同餘法，其中乘數設為 9821，除數設為10^6以及增量設為 211327。先藉由下式產生x_i

$$x_i = 9821x_{i-1} + 211327 \pmod{10^6} \quad i = 1,2,3,....$$

再將x_i除以10^6而產生 U（0,1）的亂數，即

$$u_i = \frac{x_i}{10^6}$$

因為亂數產生器是位於 EXCEL 97 的資料分析選單之下，通常使用者要先在工具>增益集中點選分析工具箱，這樣子才可以在工具之下有資料分析的功能，接著我們再點選工具>資料分析>亂數產生器，就可以得到以下的對話盒內容，我們把對話盒內容設定如下圖，就可以產生 100 筆 1000 個 U（0,1）的亂數值。此外，若想要得到重覆的亂數值，只要先在把亂數基值為某固定值，之後也可以再使用這個數值來產生相同的亂數值，不過若使用者沒有設亂數基值，則在 EXCEL 97，我們就會得到以下的數列：0.3820, 0.1007, 0.5965, 0.8991, 0.8846,…. ，所以在使用亂數之前，一定要先把種子重設成特定值，否則就會得到完全一樣的亂數值。

另外，以下也提供利用 Visual Basic 編輯器執行一次就可以得到多個變數個數及多個亂數個數的程式，在此的變數個數設為 100、亂數個數設為 1000：

Visual Basic 編輯器：

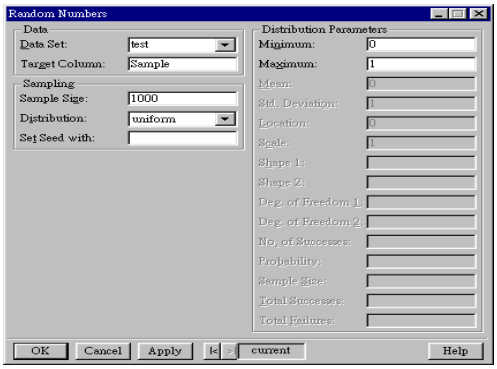
Public Function test()
  For j = 1 To 100
  For i = 1 To 1000
  Sheets("Sheet1").Cells(i, j) = Rnd
  Next i
  Next j
End Function

第四節　S-PLUS 2000

目前 S-PLUS 2000 內設 U（0,1）亂數產生器是參考 Marsaglia 在 1973 年所提出的 Super-Duper 亂數產生器，這個方法結合了線性同餘法：$X_n = 69{,}069 X_{n-1} \pmod{2^{32}}$，以及移動記錄法：$b_i = b_{i-32}$ EOR b_{i-17} ，主要是藉由 EOR 運算把這二組不同的產生法各以二進位形式表示的 32 位元數列合併，再轉換為十進位形式，接著除以 2^{31} 轉換為介於 0 和 1 之間的值。

如果要藉由 S-PLUS 2000 的程式來產生 U（0,1）的亂數值，可以利用『runif(n)』的函數來執行，當中的 n 是此函數所要產生的亂數個數。如果要得到重覆的亂數值，可以另外下 set.seed (i)的指令，就可以得到固定的起始值，i 的值可設為 0 至 1000。

另外，我們也可以在 S-PLUS 2000 的視窗環境下來操作，在點選 *Data* > *Random Numbers* 後，我們把對話盒內容設定如下，就可以在檔案 test 中產生一筆 1000 個 U（0,1）的亂數值，其中的亂數基值是用來產生亂數的選擇性數值，之後也可以再使用這個數值來產生相同的亂數值。



不過由於一般在實際模擬亂數值時，通常會需要大量的亂數值，因此以下提供執行一次就可以得到多個變數個數及多個亂數個數的程式，在此的變數個數設為 100、亂數個數設為 1000：

S-PLUS 2000 程式：

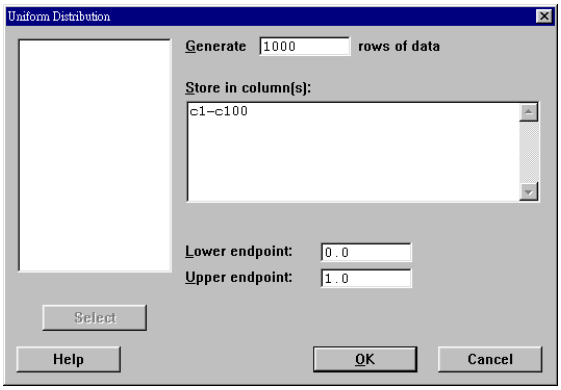
```
a_function(stream=100,n=1000){
   y_NULL
  for (i in 1:stream){
   x_runif(n)
   y_c(y,x)
   }
   matrix(y,ncol=stream)
   }
```

第五節　MINITAB 12

目前 MINITAB 12 設 U（0,1）亂數產生器是參考 L'Ecuyer（1988）所提出的組合產生器，主要是針對 16 位元的電腦所設計，組合了三組線性同餘法，形式如下：

$$x_i = 157x_{i-1} \pmod{32363}$$
$$y_i = 146y_{i-1} \pmod{31727}$$
$$z_i = 142z_{i-1} \pmod{31657}$$
$$u_i = \frac{x_i}{32363} + \frac{y_i}{31727} + \frac{z_i}{31657} \pmod 1 \quad i = 1,2,3,\ldots$$

如果我們要藉由 MINITAB 12 的視窗環境來產生 U（0,1）的亂數值，在點選 *Calc* > *Random data* > *Uniform Distribution* 之後，我們把對話盒內容設定如下圖，就可以產生 100 筆 1000 個 U（0,1）的亂數值。此外，若想要得到重覆的亂數值，只要先在 *Calc* > *Random data* > *Set Base* 設定亂數基值為某固定值，之後也可以再使用這個數值來產生相同的亂數值，不過若使用者沒有設亂數基值，起始值就會以當時電腦時間作為三個線性同餘法起始值代入。



相同的，如果我們要利用 MINITAB 12 的程式一次就得到 100 個變數個數及 1000 個亂數個數，就需要把以下的程式先存起來，

MINITAB 12 程式：
let k2=k1 RANDOM 1000 Ck2;   Uniform 0 1. add k1 1 k1

接著在 *Edit > Command Line Editor* 下，執行：

```
let k1=1.
execute 'c:\\ test.txt' 100.
```

第四章　亂數產生器的比較

在研究方法中曾經提到，一個好的亂數產生器在實用上通常需要具備長週期、良好的統計性質和有效率的電腦執行等三項性質。以下我們先介紹關於統計檢定的部分，至於週期和電腦執行方面，留到下一章的模擬結果再作進一步的討論。

統計檢定可分為理論檢定（theoretical test）和實證檢定（empirical test）兩部分，理論檢定主要是藉由分析亂數產生器的演算方法作先驗（priori）的分析，實際上並不需要產生亂數數列，就可以來評估此亂數數列的整體（global）行為。而實證檢定主要是根據亂數產生器實際模擬出來的亂數數列作後驗（posteriori）的統計分析，加以評估亂數數列的局部（local）行為。

第一節　理論檢定

Marsaglia（1968）指出，如果將線性同餘法所產生的連續 t 個亂數所構成點集合，例如（$u_1,u_2,\ldots,u_t$），（$u_{t+1},u_{t+2},\ldots,u_{2t}$），……，置於 t 維空間中，則這些點都會落在較少的平行超平面（Parallel Hyperplanes）之上，形成所謂的格子結構（Lattice Structure）。如圖 4-1、圖 4-2 分別是 $x_i = 37x_{i-1}+1 \pmod{64}$ 和 $x_i = 21x_{i-1}+1 \pmod{64}$ 兩個亂數產生器在平面上的格子結構，雖然皆為全週期（符合定理 2.1），但是可以很明顯看出圖 4-1 的格子點分佈較均勻，所以在二維空間而言，$x_i = 37x_{i-1}+1 \pmod{64}$ 所產生的亂數有較好的格子結構。

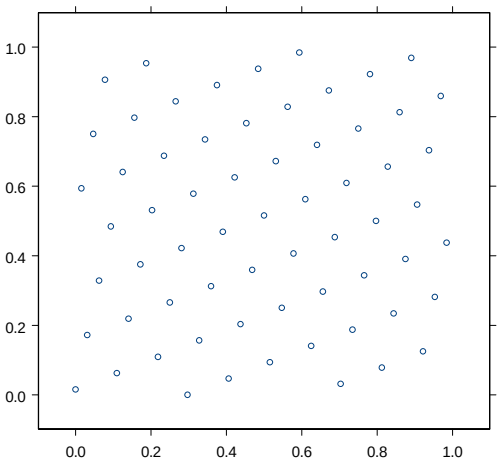


圖 4-1： $x_i = 37x_{i-1} + 1 \pmod{64}$ 在二維空間的格子結構

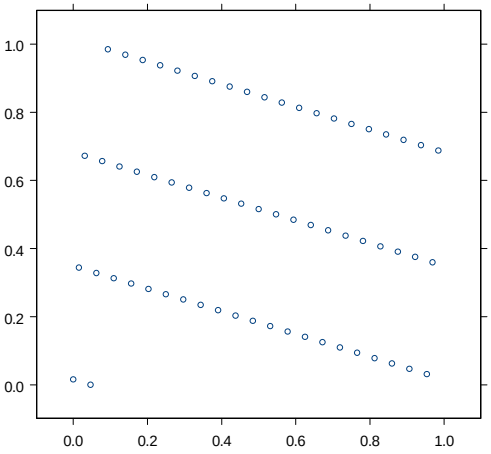


圖 4-2： $x_i = 21x_{i-1} + 1 \pmod{64}$ 在二維空間的格子結構

另外，若我們以 $x_i = 37x_{i-1} + 1 \pmod{64}$ 亂數產生器中所產生的連續三個亂數為空間中的一點，當從某一角度來看這些點時，如圖 4-3，會發現形成一組平行線的格子點結構。

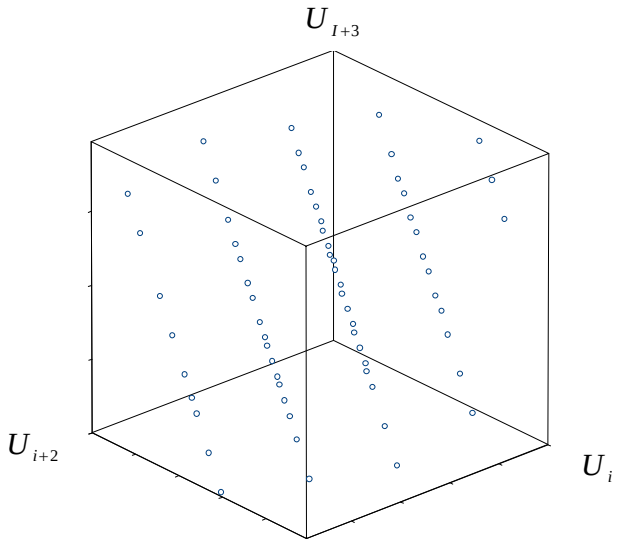


圖 4-3 $x_i = 37x_{i-1} + 1 \pmod{64}$ 在三維空間的格子點結構

而理論檢定就是針對格子點結構提出各種不同的評價方法，所採用的測量方法主要有下列五種：

1. 相鄰的平行超平面之間的最大距離，此法也稱為光譜檢定（Spectral test）（Coveyou 和 MacPherson 1967）。
2. 最長的與最短的基底向量的長度比例，此法也稱為格子檢定（Lattice test）（Beyer,，Roof，Williamson1971，Marsaglia 1972）。
3. 最小平行平面個數（Marsaglia 1968）。
4. 格子間最短距離（Smith 1971）。
5. 差異性（Discrepancy）（Niederreiter 1977）。

目前理論檢定最常見的方法是光譜檢定和格子檢定，但是這些檢定主要是針對線性同餘法來考量，同時要在全週期的條件下才可以對亂數產生器做比較，故我們無法就所有的統計套裝軟體的亂數產生器做理論檢定。

第二節　實證檢定

除了利用理論檢定來評估亂數數列的整體行為之外，我們也可以藉由亂數產生器模擬出來的亂數數列，以實證檢定的方法檢定其區域行為，一般亂數數列的實證檢定方法可分為二類：一類是均勻分配檢定（適合度檢定），另一類則是獨立性檢定（隨機性檢定）；在均勻分配檢定的部分有卡方檢定（chi-square test）、序列檢定（serial test）、K-S 檢定（Kolmogorov-Smirnov Test）等三種檢定；而在獨立性檢定的部分有升降趨勢檢定（runs up and runs down Test）、上升趨勢檢定（runs up test）、間隔檢定（gap test）、排列檢定（permutation test）及相關檢定（correlation Test）等五種檢定。

接著再利用 L'Ecuyer（1992）所提出的二階段檢定（Two-level Test）來測試亂數產生器的統計性質，方法是在第一階段是先產生 N 組的亂數數列，然後對每一組數列做統計檢定，故我們可以得到 N 個統計值所對應的 p 值$T_1,T_2,\ldots,T_N$，接下來的第二階段則是對$T_1,T_2,\ldots,T_N$此 N 個統計值所對應的 p 值之實證分配與統計檢定之理論分配作 K-S 適合度檢定，理論上 p 值的分配會近似 U(0,1)分配，所以如果 K-S 適合度檢定的結果為拒絕H_0的話，也就是這個亂數產生器沒有通過此項實證檢定，反之亦然。

以下將分別介紹各種實證檢定的方法，為了方便描述，在此以 Montgomery（1997）所附的亂數表中的亂數值（可參考附表 1）為例，做各種不同的檢定。有關實證檢定之程式部分，詳見附錄 3。

2.1 均勻分配檢定

2.1.1 卡方檢定

以卡方檢定法檢定亂數數列$u_1,u_2,\ldots,u_n$是否均勻分佈在（0,1）區間，主要是把（0,1）區間等分為 k 個小區間，然後比較各個小區間的觀察次數和理論次數的差異，卡方檢定統計量為：

$$X^2=\frac{\sum_{i=1}^{k}(o_i-e_i)^2}{e_i}$$

o_i：落在第i個區間亂數數列的觀察次數

e_i：落在第i個區間亂數數列的理論次數；$e_i=n/k$

當n很大時，檢定統計量X^2會近似自由度為k-1的卡方分配。由於X^2在 n 較大時近似的效果較好，為了避免其誤差，通常要求$e_i\geq 5$；若e_i小於5，通常需要與其他的區間合併，使合併後滿足$e_i\geq 5$的要求。

以附表 1 的 360 筆亂數值為例，我們想要以卡方檢定法檢定其是否為U(0,1)的均勻分配，小區間數（k）設為 10，因此各個小區間內的期望次數為 36，經表 4-1 的計算過程，檢定統計量X^2=9.333 < $\chi^2_{0.05,9}$=16.919，亦即，在顯著水準α=0.05 之下，此次樣本並無足夠的證據拒絕此亂數為均勻分配 U(0,1)的假設。

表 4-1 卡方檢定的計算過程

i	區間	觀測次數（o_i）	期望次數（e_i）	$(o_i-e_i)^2/e_i$
1	[0.0, 0.1)	42	36	1.00
2	[0.1, 0.2)	38	36	0.11
3	[0.2, 0.3)	45	36	2.25
4	[0.3, 0.4)	31	36	0.69
5	[0.4, 0.5)	34	36	0.11
6	[0.5, 0.6)	41	36	0.69
7	[0.6, 0.7)	32	36	0.44
8	[0.7, 0.8)	24	36	4.00
9	[0.8, 0.9)	37	36	0.03
10	[0.9, 1.0)	36	36	0.00
				X^2=9.333

2.1.2 序列檢定

序列檢定法為卡方檢定法將其推廣至d維空間之延伸應用，主要是檢定亂數數列$u_1,u_2,\ldots,u_n$所形成的d維數列$(u_1,\ldots,u_d),(u_{d+1},\ldots,u_{2d}),\ldots$是否均勻的分佈在$(0,1)^d$區間。檢定的方法是在每一個維度中，我們都將區間（0,1）等分成 k 個子區間（$\frac{n/d}{k^d}\geq 5$），之後檢視小數列內每一數值落在哪一個之小區間，令$f(j_1 j_2 \cdots j_d)$表示小數列內第一個數值落於小區間j_1，第二個數值落於小區間j_2，…，第d個數值落於小區間j_d的次數和，則檢定統計量為：

$$X^2=\frac{\sum_{j_1=1}^{k}\sum_{j_2=1}^{k}\cdots\sum_{j_d=1}^{k}[f(j_1 j_2\cdots j_d)-\frac{n/d}{k^d}]^2}{\frac{n/d}{k^d}}$$

當$\frac{n}{d}$很大時，檢定統計量X^2會近似自由度為k^d-1 的卡方分配。

利用附表 1 的 360 筆亂數值做序列檢定，我們想要知道此亂數是否均勻分佈在 2 維空間中，我們把（0,1）等分為（$0,\frac{1}{3}$）、（$\frac{1}{3},\frac{2}{3}$），（$\frac{2}{3}$,1）3 個小區間，經檢視後，得到表 4-2，其檢定統計量X^2=13.9<$\chi^2_{0.05,8}$=15.509，亦即，在顯著水準α=0.05 之下，並無足夠的證據拒絕此亂數均勻的分佈在$(0,1)^2$區間。

表 4-2　序列檢定的計算過程

集合	機率	期望次數	觀察次數
(1,1)	1/9	20	30
(1,2)	1/9	20	29
(1,3)	1/9	20	17
(2,1)	1/9	20	15
(2,2)	1/9	20	15
(2,3)	1/9	20	18
(3,1)	1/9	20	16
(3,2)	1/9	20	23
(3,3)	1/9	20	17

2.1.3 K-S 檢定

以 K-S 檢定法檢定亂數數列$u_1, u_2, \ldots, u_n$是否均勻分佈在（0,1）區間，主要是在比較亂數數列所構成的實證分配函數與 U(0,1)的理論分配函數是否一致，找出實證和理論累積分配函數之間的最大差異。K-S 檢定不必像卡方檢定需要對資料加以分組，因此較不會有資訊的損失。K-S 檢定的統計量為：

$$D = \max\{D^+, D^-\}$$

$$D^+ = \max_{1\le i\le n}\left|\frac{i}{n} - F(u_{(i)})\right| \quad , D^- = \max_{1\le i\le n}\left|F(u_{(i)}) - \frac{(i-1)}{n}\right|$$

其中的$F(u_{(i)})$為數列由小至大排序後，第i個亂數值的理論分配函數值。D的臨界值可以藉由查 K-S 檢定表得到。

我們以附表 1 的前 10 筆亂數值：0.10480, 0.22368, 0.24130, 0.42167, 0.37570, 0.77921, 0.99562, 0.96301, 0.89579, 0.85475 為例，說明 K-S 檢定法的演算過程。我們將亂數排序後，先計算出理論分配為 U(0,1) 的累積分配函數$F(u_{(i)})$，再計算實證分配的累積分配函數，求得彼此間差異的絕對值，經表 4-3 的計算過程，檢定統計量D =0.27921< $D_{(0.025,10)}$=0.40925，亦即，在顯著水準α=0.05 之下，此次樣本並無足夠的證據拒絕此亂數為均勻分配 U(0,1)的假設。

表 4-3　K-S 檢定的計算過程

i	$u_{(i)}$	$F(u_{(i)})$	i/n	$(i-1)/n$	$\lvert i/n - F(u_{(i)})\rvert$	$\lvert (i-1)/n - F(u_{(i)})\rvert$
1	0.10480	0.10480	0.1	0	0.00480	0.1048
2	0.22368	0.22368	0.2	0.1	0.02368	0.12368
3	0.24130	0.24130	0.3	0.2	0.05870	0.04130
4	0.37570	0.37570	0.4	0.3	0.02430	0.07570
5	0.42167	0.42167	0.5	0.4	0.07833	0.02167
6	0.77921	0.77921	0.6	0.5	0.17921	0.27921
7	0.85475	0.85475	0.7	0.6	0.15475	0.25475
8	0.89579	0.89579	0.8	0.7	0.09579	0.19579
9	0.96301	0.96301	0.9	0.8	0.06301	0.16301
10	0.99562	0.99562	1	0.9	0.00438	0.09562
					D^+= 0.17921	D^-= 0.27921

2.2 獨立性檢定

2.2.1 升降趨勢檢定

升降趨勢檢定的主要目的是在檢查亂數數列$u_1,u_2,\ldots,u_n$，其遞增或遞減長度個數的分佈情況，從而判斷此數列是否具有獨立性。如果數列不是獨立的話，則可能會有大幅度的連續上升或連續下降、或是具有某特定的循環規則。檢定的方法是把數列轉換為正負符號表示，若相鄰的兩數值為遞增，則以"+"表示，如遞減則以"—"表示。令 R 表示由遞增轉為遞減或由遞減轉為遞增的次數，在小樣本時，檢定統計量 R 的臨界值可以藉由查升降趨勢檢定表得到。Levene（1952）求出 R 的期望值和變異數分別為：

$$E(R)=\frac{2n-1}{3}\quad,\quad V(R)=\sqrt{\frac{16n-29}{90}}\quad,$$

當 n 很大時，R 會近似服從標準常態分配。

利用附表 1 的 360 筆亂數值做升降趨勢檢定，則數列可轉換為：

$$\underbrace{+\,+\,+}_{1}\;\underbrace{-}_{2}\;\underbrace{+\,+}_{3}\;\underbrace{-\,-\,-\,-}_{4}\;\underbrace{+}_{5}\;\underbrace{-}_{6}\;\cdots\cdots\;\underbrace{-}_{240}\;\underbrace{+}_{241}\;\underbrace{-}_{242}\;\underbrace{+}_{243}\;\underbrace{-}_{244}\;\underbrace{+\,+\,+}_{245}$$

也就是 R=245，故$|Z|=\left|\frac{R-E(R)}{\sqrt{V(R)}}\right|=0.752947<Z_{0.925}=1.96$。在顯著水準 α=0.05 之下，無足夠的證據拒絕此亂數具有獨立性的假設。

2.2.2 上升趨勢檢定

上升趨勢檢定的主要目的也是在判斷亂數數列是否具有獨立性，而在此種檢定法下，我們定義區間（run）為遞增數值所成之集合，區間長度（length of run）定義為該區間內之數值的個數。在此以附表1的前20筆亂數值為例，一共可分割成11個區間{0.10480, 0.22368, 0.24130, 0.42167}, {0.37570, 0.77921, 0.99562}, {0.96301}, {0.89579}, {0.85475}, {0.28918, 0.63553}, {0.94290, 0.10365}, {0.07119, 0.51085}, {0.23680}, {0.10110, 0.52162}, {0.07056}。

依據 Knuth（1981）之研究，數列在劃分成各區間後，區間之間並非

互為獨立，較長區間後經常伴隨著較短區間，而為了避免此種現象，Knuth 的作法是把區間後緊鄰之值捨棄，以使各區間長度互為獨立。例如上述數列應將 0.37570, 0.96301, 0.85475, 0.9429, 0.07119, 0.2368, 0.07056 值等捨棄後，劃分區間如下:{0.1048, 0.22368, 0.2413, 0.42167}, {0.77921, 0.99562}, {0.89579}, {0.28918, 0.63553}, {0.10365}, {0.51085}, {0.1011, 0.52162}，區間長度分別為 4, 2, 1, 2, 1, 1, 2，其中區間長度為 1、2、4 者分別為 3、3、1 個。

理論上區間長度為 k 的機率分配為 $\frac{k}{(k+1)!}$，而區間長度至少為 m 的機率則為 $1/m!$。然後再根據各區間長度類別下之實際個數和理論個數的比較，定義檢定統計量

$$X^2=\sum_{k=1}^{m-1}\frac{\left(f_i-\frac{vk}{(k+1)!}\right)^2}{\frac{vk}{(k+1)!}}+\frac{\left(f_m-\frac{v}{m!}\right)^2}{\frac{v}{m!}}$$

f_k：區間長度為 k 的個數，$k\le m-1$

f_m：區間長度至少為 m 的個數

其中 v 為區間的總個數，m 的取決在於能夠使個別的期望次數（$\frac{vk}{(k+1)!}$, $v/m!$）皆大於 5 以上。如果此亂數服從獨立性的話，檢定統計量 X^2 會近似自由度為 m-1 的卡方分配。

另外再利用附表 1 的 360 筆亂數值做升降趨勢檢定，在 m 為 4 的假設下，可得到表 4-4 的結果，其檢定統計量 X^2= 3.761538< $\chi^2_{0.05,3}$=7.815，亦即，在顯著水準 α=0.05 之下，無足夠的證據拒絕此亂數具有獨立性的假設。

表 4-4　上升趨勢檢定的計算過程

區間長度	機率	期望次數	觀察次數
1	0.5	65	67
2	0.333	43.29	35
3	0.125	16.25	22
4 以上	0.042	5.46	6

2.2.3 間隔檢定

間隔檢定的主要目的是在檢查亂數數列$u_1,u_2,\ldots,u_n$，觀察其是否位於選定的區間(α,β)內，從而判斷此數列是否具有獨立性。定義間隔(gap)為不落在區間內的集合（$U_i \notin (\alpha,\beta)$），所以間隔的長度（gap length）就是亂數$u_1,u_2,\ldots,u_n$中直到落到區間(α,β)內之前所需要的觀察次數。如果$u_1,u_2,\ldots,u_n$是獨立，則間隔長度的分配就會是參數為$(\beta-\alpha)$的幾何分配，即

$$P(\text{gap length} = i) = (\beta-\alpha)(1-\beta+\alpha)^i \qquad i = 0,1,2\ldots$$

理論上間隔長度為i的機率分配為$(\beta-\alpha)(1-\beta+\alpha)^i$，而間隔長度為$m$值時的機率則為$(\beta-\alpha)^m$。然後再根據各間隔長度類別下之實際個數和理論個數相互比較，則檢定統計量為：

$$X^2 = \sum_{i=0}^{k-2} \frac{\left(f_i - v(\beta-\alpha)(1-\beta+\alpha)^i\right)^2}{v(\beta-\alpha)(1-\beta+\alpha)} + \frac{\left(f_m - v(\beta-\alpha)^m\right)^2}{v(\beta-\alpha)^m}$$

f_k：間隔長度為i的個數，$i = 0,1,2\ldots$

f_m：間隔長度為m的個數

其中v為間隔的總個數，m的取決在於能夠使個別的期望次數$v(\beta-\alpha)(1-\beta+\alpha)^i$，$v(\beta-\alpha)^m$）皆大於 5 以上。如果此亂數服從獨立性的話，檢定統計量X^2會近似自由度為m的卡方分配。

利用附表 1 的 360 筆亂數值做間隔檢定，在區間(α,β)為(0.5,1)和 m 為 4 的假設下，可得到表 4-5 的結果，其檢定統計量X^2= 2.705882 $<\chi^2_{0.05,4}$=9.488，亦即，在顯著水準α=0.05 之下，無足夠的證據拒絕此亂數具有獨立性的假設。

表 4-5　間隔檢定的計算過程

間隔長度	機率	期望次數	觀察次數
0	0.5	85	76
1	0.25	42.5	46
2	0.125	21.25	24
3	0.0625	10.625	14
4 以上	0.0625	10.625	10

2.2.4 排列檢定

排列檢定的主要是將亂數數列$u_1,u_2,\ldots,u_n$，分割成不同的小組，觀察各組內數值大小排列的情況，藉以判斷此數列是否具有獨立性。我們將亂數數列$u_1,u_2,\ldots,u_n$依序分為 k 個一組：$(u_1,\ldots,u_k),(u_{k+1},\ldots,u_{2k}),\ldots$然後對每一個的集合中的觀察值排序（最小值為 1，最大值為 k），所以會有 k! 種可能的排列情形，如果亂數數列是獨立的，則各種排列情形發生的機率會是均等的（皆為$\frac{1}{k!}$）。先計算各種排列下的次數，再根據各排列組合下之實際個數和理論個數相互比較，即可判斷此數列是否具有獨立性。檢定統計量為：

$$X^2=\frac{\sum_{j_1\neq j_2\neq\cdots}\sum\cdots\sum_{\neq j_k}[f(j_1j_2\cdots j_k)-\frac{n/k}{k!}]^2}{\frac{n/k}{k!}}$$

如果此亂數服從獨立性的話，檢定統計量X^2會近似自由度為 k!-1 的卡方分配。

利用附表 1 的 360 筆亂數值做排列檢定，依序將亂數數列$u_1,u_2,\ldots,u_n$分為 3 個一組，經檢視後，得到表 4-6，其檢定統計量X^2= 3.4 $<\chi^2_{0.05,5}$=11.07，亦即，在顯著水準α=0.05 之下，無足夠的證據拒絕此亂數具有獨立性的假設。

表 4-6　排列檢定的計算過程

排列情形	機率	期望次數	觀察次數
(1,2,3)	1/6	20	25
(1,3,2)	1/6	20	19
(2,1,3)	1/6	20	14
(2,3,1)	1/6	20	21
(3,1,2)	1/6	20	22
(3,2,1)	1/6	20	19

2.2.5 相關檢定

相關檢定的主要目的是在檢查亂數數列中u_i與u_{i+j}的相關性，在亂數數列$u_1, u_2, \ldots, u_n$為均等分配的假設下，可以得到u_i與u_{i+j}的相關係數的估計值為

$$\hat{\rho}_j = \frac{12}{h_j + 1} \sum_{k=0}^{h_j} u_{1+kj} u_{1+(1+k)j} - 3$$

其中$h_j = \lfloor (n-1)/j \rfloor - 1$

$\lfloor \ \rfloor$：高斯符號

在虛無假設$\rho_j = 0$的情況及 n 很大時，檢定統計量$A_j = \dfrac{\hat{\rho}_j}{\sqrt{\dfrac{13h_j + 7}{(h_j + 1)^2}}}$會近似服從標準常態分配。

利用附表 1 的 360 筆亂數值做相關檢定，取$j = 1,2,3$，檢定統計量A_1, A_2, A_3分別為-1.07 、-0.86、 -0.52，經檢視後，個別的絕對值均小於$Z_{0.975}$=1.96，亦即，在顯著水準α=0.05 之下，無足夠的證據拒絕此亂數具有獨立性的假設。

第五章　分析結果

第一節　週期

由於一般電腦的亂數產生器都是藉由演算法來產生亂數，故亂數數列都有週期性，一個能夠產生較長週期的亂數產生器，將不會發生重覆使用相同亂數，而造成相關性。以下將分別探討五種軟體的週期長度。

SAS 6.12、SPSS 8.0 和 EXCEL 97 三者的亂數產生器皆是藉由線性同餘法所產生，其中 SAS 6.12 和 SPSS 8.0 兩者的乘數為 397,204,094，除數為 $2^{31}-1$，由定理 2.2 可以得到週期的長度為 $2^{31}-2$；而 EXCEL 97 的亂數產生器的乘數為 9,821，除數為 10^6 以及增量為 211,327，由定理 2.1 可以得到週期的長度為 10^6。

而 S-PLUS 2000 和 MINITAB 12 兩者的亂數產生器則是藉由組合法所產生的。S-PLUS 2000 亂數產生器的方法組合了線性同餘法和移動記錄法，此方法所得到的週期長度為 6.6×10^{14}（Ripley, 1994）；MINITAB 12 亂數產生器的方法是組合了三組乘法型線性同餘法，此方法所得到的週期長度為 8.1×10^{12}（L'Ecuyer, 1988）。

將上述的結果整理如下表所示，其中 S-PLUS 2000 亂數產生器的週期長度是五種軟體中最長的，而 EXCEL 97 亂數產生器的週期長度雖然是五種軟體中最短的，但是週期長度也有一百萬，應該足夠提供一般使用者在亂數上的需求。

表 5-1：五種軟體之週期長度

	SAS 6.12	SPSS 8.0	EXCEL 97	S-PLUS 2000	MINITAB 12
週期長度	2.1×10^{10}	2.1×10^{10}	10^6	6.6×10^{14}	8.1×10^{12}

第二節　統計檢定

在統計檢定部分，我們最主要是藉由實際從各個不同的套裝軟體所產生的亂數來做比較，我們採取的方法是利用電腦時間作為起始值代入各個軟體的亂數產生器，連續產生 1000 組，每組亂數的個數分別為 500 及 1000 筆，在顯著水準$\alpha = 0.05$之下，做二階段檢定。

2.1　亂數個數為 500 筆

在第一階段的統計檢定中，利用先前所提到的八種實證檢定法，分別為卡方檢定、K-S 檢定、序列檢定、升降趨勢檢定、上升趨勢檢定、間隔檢定、排列檢定及相關檢定等檢定。在卡方檢定部分，我們在此是利用 S-PLUS 2000 內設的函數（chisq.gof）來執行，而其內設的區間數 k 是參考 Moore (1986)所提供的方法，其公式為： $k = \lfloor 2n^{0.4} \rfloor + 1$，所以在亂數個數為 500 時，區間數設為 25。在 K-S 檢定部分，在此是利用 S-PLUS 2000 內設的函數(ks.gof）來執行。在序列檢定部分則考慮二維、三維的序列檢定，區間數分別設為5^2和3^3。在上升趨勢檢定部分，區間數 k 設為 4。在間隔檢定部分，區間設為（0, 0.5），間隔長度 m 設為 5。在排列檢定部分，我們將亂數數列依序分為 3 個一組。在相關檢定部分，考慮只檢定前三階，採用三階統計量的最小 p 值；以上的程式請參考附錄 4。

第二階段再利用第一階段所得的 1000 個 p 值做 K-S 檢定，檢定其 p 值是否為 U(0,1)分配，結果如下表所示。

表 5-2：

	SAS 6.12	SPSS 8.0	EXCEL 97	S-PLUS 2000	MINITAB 12
卡方檢定	0.506	0.279	0.197	0.750	0.584
K-S 檢定	0.179	0.505	0.455	0.279	0.639
2 維序列檢定	0.450	0.210	0.864	0.039	0.794
3 維序列檢定	0.093	0.251	0.038	0.566	0.005
排列檢定	0.556	0.402	0.063	0.459	0.177
間隔檢定	0.580	0.862	0.840	0.111	0.312

上升趨勢檢定	0.760	0.170	0.937	0.114	0.335
升降趨勢檢定	0.066	0.009	0.037	0.010	0.024
相關檢定	0.000	0.000	0.000	0.000	0.000

註：加網底表示 p 值≤0.05

2.2 亂數個數為 1000 筆

在第一階段的統計檢定中，利用先前所提到的八種實證檢定法，分別為卡方檢定、K-S 檢定、序列檢定、升降趨勢檢定、上升趨勢檢定、間隔檢定、排列檢定及相關檢定等檢定。在卡方檢定部分，區間數設為 32。在 K-S 檢定部分，在此是利用 S-PLUS 2000 內設的函數（ks.gof）來執行。在序列檢定部分則考慮二維、三維的序列檢定，區間數分別設為5^2和3^3。在上升趨勢檢定部分，區間數 k 設為 4。在間隔檢定部分，區間設為(0,0.5)，間隔長度 m 設為 5。在排列檢定部分，我們將亂數數列依序分為 3 個一組。在相關檢定部分，考慮只檢定前三階，採用三階統計量的最小值；以上的程式請參考附錄 5。

第二階段再利用第一階段所得的 1000 個 p 值做 K-S 檢定，檢定其 p 值是否為 U(0,1)分配，結果如下表所示。

表 5-3：

	SAS 6.12	SPSS 8.0	EXCEL 97	S-PLUS 2000	MINITAB 12
卡方檢定	0.959	0.988	0.541	0.917	0.426
K-S 檢定	0.156	0.853	0.230	0.255	0.337
2 維序列檢定	0.723	0.513	0.094	0.361	0.093
3 維序列檢定	0.187	0.674	0.265	0.284	0.570
排列檢定	0.732	0.904	0.498	0.636	0.545
間隔檢定	0.790	0.179	0.290	0.881	0.348
上升趨勢檢定	0.256	0.667	0.356	0.726	0.559
升降趨勢檢定	0.047	0.105	0.191	0.397	0.005
相關檢定	0.000	0.000	0.000	0.000	0.000

註：加網底表示 p 值≤0.05

2.3 蒙地卡羅積分法

由於先前的統計檢定方法中，無法判斷那一種軟體是較為理想的，理論上並無一致的解釋，於是在此採用蒙地卡羅法的求解積分值進行模擬，希望藉由模擬的結果比較出各個軟體在求解積分值的表現。在此先對蒙地卡羅法做一些介紹。蒙地卡羅法是在二次世界大戰時期，在美國 Los Alamos 發展核子武器所使用的一些分析方法。因為這個方法和賭博、亂數有關，所以 Neumann 和 Ulam 等人就用當時相當著名的賭場所在地蒙地卡羅（Monte Carlo）來命名。蒙地卡羅法的基本觀念，主要是在計算過程中利用了隨機亂數進行求解的一種技巧，只要知道機率分配函數，大部分的事件均可由蒙地卡羅法來模擬。一般而言，蒙地卡羅法的主要步驟可歸納如下：

1. 對每一種不同的問題，設計適合於該問題的隨機過程（stochastic process），而且此隨機過程中之各參數必須與問題之參數有相等的量（值）。
2. 觀察此隨機過程中各參數發生的量（值），再利用統計方法計算出該參數在問題中的量（值）。

而蒙地卡羅法的主要依據是建立在大數法則（law of large number）及中央極限定理（central limit theorem）之上，所以我們經常要藉由大量的重複試驗和亂數值來得到問題的結果。蒙地卡羅法的應用相當的廣泛，接下來將以求解積分值作為應用的例子， 考慮一維積分：

$$\theta = \int_a^b f(x)dx$$

其中 $f(x)$ 是 x 的任意函數，如果我們在 (a,b) 間隨機均勻選取 n 個 x_i 的值，然後把 n 個 $f(x_i)$ 所形成的 f_i 集合，看成是隨機變數的集合，則

$$\theta = (b-a)\frac{\sum_{i=1}^{n} f(x_i)}{n}$$

其中的 $\frac{\sum_{i=1}^{n} f(x_i)}{n}$ 即為 $f(x_i)$ 的平均值，這種方法類似隨機選取得許多不同 $f(x_i)$ 的值，然後取其平均值（總和除以樣本數），此平均值可看成原積分值除以變數的積分範圍，這樣的方法稱為直接取樣法（Straight Sampling），這種積分方法可推廣至任意 n 維空間之積分。

接下來以求 $\int_{-\infty}^{\infty} e^{-x^2} dx$ 的積分值為例，利用各個軟體亂數產生器實際所產生的亂數，應用蒙地卡羅法來做 $\int_{-\infty}^{\infty} e^{-x^2} dx$ 的估計。

一開始先令 $y = \frac{1}{1+x}$，將 $\int_{-\infty}^{\infty} e^{-x^2} dx$ 轉換為 $\int_0^1 \frac{2}{y^2} e^{-(\frac{1}{y}-1)^2} dy$，利用各個軟體所得到的U(0,1)亂數代入 $\frac{2}{y^2} e^{-(\frac{1}{y^2}-1)^2}$，再個別乘以 dx（$\frac{1}{n}$），然後把個別的機率值相加總，即可得出 $\int_{-\infty}^{\infty} e^{-x^2} dx$ 的近似值。 然後將各個近以值和 $\int_{-\infty}^{\infty} e^{-x^2} dx$ 的真實值[4]做比較，將結果整理如表，不同的軟體所得到的誤差率都算是蠻小的，其中以EXCEL 97 的表現最好，誤差率在 0.1%之下。

表 5-4：五種軟體在積分值之比較

	SAS 6.12	SPSS 8.0	EXCEL 97	S-PLUS 2000	MINITAB 12
n=500 重覆 1000 次	1.768813 $\lvert\hat{e}\rvert \approx 0.2054\%$	1.784288 $\lvert\hat{e}\rvert \approx 0.6676\%$	1.770828 $\lvert\hat{e}\rvert \approx 0.0917\%$	1.769235 $\lvert\hat{e}\rvert \approx 0.1816\%$	1.775522 $\lvert\hat{e}\rvert \approx 0.1731\%$
n=1000 重覆 1000 次	1.769274 $\lvert\hat{e}\rvert \approx 0.1794\%$	1.773801 $\lvert\hat{e}\rvert \approx 0.0760\%$	1.772364 $\lvert\hat{e}\rvert \approx 0.0051\%$	1.775115 $\lvert\hat{e}\rvert \approx 0.1501\%$	1.774833 $\lvert\hat{e}\rvert \approx 0.1342\%$

[4] $\int_{-\infty}^{\infty} e^{-x^2} dx = 2\int_0^{\infty} e^{-x^2} dx = 2 \times \frac{1}{2} \times \Gamma(\frac{1}{2}) = \sqrt{\pi} \cong 1.772453851$

第三節　電腦執行

在電腦執行方面，這五種軟體都滿足可重覆性，都提供起始值的功能，可參考第三章的操作說明，接下來就對亂數產生器的執行時間，以產生 100 組 500 筆、100 組 1000 筆和 250 組 1000 筆的亂數值為例，結果列於表 5-5，其中，SAS 6.12 的執行效率相當的高，EXCEL 97 在執行效率的表現就不是很好。

表 5-5：五種軟體在執行時間之比較　　單位：秒

	SAS 6.12	SPSS 8.0	EXCEL 97	S-PLUS 2000	MINITAB 12
100 組 500 筆	0.33	4	43	1	2
100 組 1000 筆	0.51	6	105	2	2
250 組 1000 筆	0.98	9	272	15	***

註：”***”表示無法執行，因為 MINITAB 12 內設的工作表取多只能存放 10,000 筆資料。

另外，在實際操作軟體上發現了一些問題：在 EXCEL 97 和 SPSS 8.0 的亂數產生器，如果在使用亂數之前，沒有先把種子重設成特定值，則在 EXCEL 97，就會得到以下的數列：0.3820, 0.1007, 0.5965, 0.8991, 0.8846,….，而在 SPSS 8.0，亂數種子就會自動重設成 2,000,000，就會得到以下的數列：0.1396, 0.4313, 0.6122, 0.2908, 0.1557,….。所以，一定要先把種子重設成特定值，否則就會得到完全一樣的亂數值。

第六章　結論與建議

亂數產生器的研究已經有很長的歷史，長久以來，大家利用各種軟體內設的亂數產生器也已成習慣，但是甚少對於所用的亂數產生器有過探討，因此本論文的主要目的在於：針對 SAS 6.12、SPSS 8.0、EXCEL 97、S-PLUS 2000 及 MINITAB 12 這五種統計分析上常使用的套裝軟體，分別介紹各個套裝軟體內設 U（0,1）亂數產生器的演算方法，以及實際上的操作使用，再利用亂數產生器所產生的亂數值來做統計分析比較。

由第五章的分析結果，可以歸納以下的結論，在週期部分，這五種軟體，週期長度最短的也有一百萬，應該可以滿足一般使用者的需求。在統計檢定部分，根據二階段檢定的結果，似乎沒有辦法很明確的評斷其優劣，另外，在此也採用蒙地卡羅法的求解積分值進行模擬，其中以 EXCEL 97 的表現最好，誤差率在 0.1%之下。最後，在電腦執行時間之比較上，SAS 6.12 的執行效率相當的高，EXCEL 97 在執行效率的表現就不是很好。

目前在亂數產生器的判斷準則上仍然相當的分歧，沒有一個單一、明確的判斷準則，而且在亂數產生器的理論發展和實際電腦的應用上還是有著相當大的差距，例如大多數的理論發展並沒有真正的落實在電腦的內設函數中，，所以如何將亂數產生器的理論發展和分析方法作一整合性的探討，仍待未來的研究者有新的突破。

附　錄

＜附表 1＞

隨機變數的分配函數	SAS 6.12	S-PLUS 2000	MINITAB 12	EXCEL 97	SPSS 8.0
均勻(uniform)分配	＊	＊	＊	＊	＊
常態(normal)分配	＊	＊	＊	＊	＊
指數常態(lognormal)分配		＊	＊		＊
指數(exponential)分配	＊	＊	＊		＊
貝他(beta)分配		＊	＊		＊
咖嗎(gamma)分配	＊	＊	＊		＊
科西(cauchy)分配	＊	＊	＊		＊
羅吉斯(logistic)分配		＊	＊		＊
韋伯(weibull)分配		＊	＊		＊
F 分配		＊	＊		＊
T 分配		＊	＊		＊
卡方(chi-square)分配		＊	＊		＊
Laplace 分配			＊		＊
Pareto 分配					＊
伯努利(bernoulli)分配			＊	＊	＊
二項(binomial)分配	＊	＊	＊	＊	＊
幾何(geometric)分配		＊	＊		＊
負二項(negbinomial)分配		＊			＊
卜瓦松(poisson)分配	＊	＊	＊	＊	＊
超幾何(hypergeometric)分配		＊	＊		＊

註：打＊者表示軟體有提供此種分配的亂數。

＜附錄 1＞

```
LCG_function(n,seed=3,m=16,a=5,cc=1){
   temp_seed
   z_NULL
   for (k in 1:n){
    temp_a*temp+cc
    temp_temp-floor(temp/m)*m
   z_c(z,temp)
   }
   z/m
   }
```

＜附錄 2＞

```
TG_function(){
      seq_NULL
      x2b_c(1, 1, 1, 1)
      for(i in 5:88)
            x2b_c(x2b, xor(x2b[i-4], x2b[i-1]))
      x2bleft_x2b
      while(length(x2bleft) >= 4) {
            seq_c(seq, sum(x2bleft[c(1:4)] * 2^(3:0)))
            x2bleft_x2bleft[-(1:4)]
      }
      list(x2b,seq/16)
}
```

<附表 2>

0.10480	0.15011	0.01536	0.02011	0.87647	0.91646	0.69179	0.14194	0.62590
0.22368	0.46573	0.25595	0.85393	0.30995	0.89198	0.27982	0.53402	0.93965
0.24130	0.48360	0.22527	0.97265	0.76393	0.64809	0.15179	0.24830	0.4934
0.42167	0.93093	0.06243	0.61680	0.07856	0.16376	0.39440	0.53537	0.71341
0.37570	0.39975	0.81837	0.16656	0.06121	0.91782	0.60468	0.81305	0.49684
0.77921	0.06907	0.11008	0.42751	0.27756	0.53498	0.18602	0.70659	0.90655
0.99562	0.72905	0.56420	0.69994	0.98872	0.31016	0.71194	0.18738	0.44013
0.96301	0.91977	0.05463	0.07972	0.18876	0.20922	0.94595	0.56869	0.69014
0.89579	0.14342	0.63661	0.10281	0.17453	0.18103	0.57740	0.84378	0.25331
0.85475	0.36857	0.53342	0.53988	0.53060	0.59533	0.38867	0.62300	0.08158
0.28918	0.69578	0.88231	0.33276	0.70997	0.79936	0.56865	0.05859	0.90106
0.63553	0.40961	0.48235	0.03427	0.49626	0.69445	0.18663	0.72695	0.52180
0.09429	0.93969	0.52636	0.92737	0.88974	0.33488	0.36320	0.17617	0.30015
0.10365	0.61129	0.87529	0.85689	0.48237	0.52267	0.67689	0.93394	0.01511
0.07119	0.97336	0.71048	0.08178	0.77233	0.13976	0.47564	0.81056	0.97735
0.51085	0.12765	0.51821	0.51259	0.77452	0.16308	0.60756	0.92144	0.49442
0.02368	0.21382	0.52404	0.60268	0.89368	0.19885	0.55322	0.44819	0.01188
0.01011	0.54092	0.33362	0.94904	0.31273	0.04146	0.18594	0.29852	0.71585
0.52162	0.53916	0.46369	0.58586	0.23216	0.14513	0.83149	0.98736	0.23495
0.07056	0.97628	0.33787	0.09998	0.42698	0.06691	0.76988	0.13602	0.51851
0.48663	0.91245	0.85828	0.14346	0.09172	0.30168	0.90229	0.04734	0.59193
0.54164	0.58492	0.22421	0.74103	0.47070	0.25306	0.76468	0.26384	0.58151
0.32639	0.32363	0.05597	0.24200	0.13363	0.38005	0.94342	0.28728	0.35806
0.29334	0.27001	0.87637	0.87308	0.58731	0.00256	0.45834	0.15398	0.46557
0.02488	0.33062	0.28834	0.07351	0.19731	0.92420	0.60952	0.61280	0.50001
0.81525	0.72295	0.04839	0.96423	0.24878	0.82651	0.66566	0.14778	0.76797
0.29676	0.20591	0.68086	0.26432	0.46901	0.20849	0.89768	0.81536	0.86645
0.00742	0.57392	0.39064	0.66432	0.84673	0.40027	0.32832	0.61362	0.98947
0.05366	0.04213	0.25669	0.26422	0.44407	0.44048	0.37937	0.63904	0.45766
0.91921	0.26418	0.64117	0.94305	0.26766	0.2594	0.39972	0.22209	0.71500
0.00582	0.04711	0.87917	0.77341	0.42206	0.35126	0.74087	0.99547	0.81817
0.00725	0.69884	0.62797	0.56170	0.86324	0.88072	0.76222	0.36086	0.84637
0.69011	0.65795	0.95876	0.55293	0.18988	0.27354	0.26575	0.08625	0.40801
0.25976	0.57948	0.29888	0.88604	0.67917	0.48708	0.18912	0.82271	0.65424
0.09763	0.83473	0.73577	0.12908	0.30883	0.18317	0.28290	0.35797	0.05998
0.91567	0.42595	0.27958	0.30134	0.04024	0.86385	0.29880	0.99730	0.55536
0.17955	0.56349	0.90999	0.49127	0.20044	0.59931	0.06115	0.20542	0.18059
0.46503	0.18584	0.18845	0.49618	0.02304	0.51038	0.20655	0.58727	0.28168
0.92157	0.89634	0.94824	0.78171	0.84610	0.82834	0.09922	0.25417	0.44137
0.14577	0.62765	0.35605	0.81263	0.39667	0.47358	0.56873	0.56307	0.67607

資料來源：Montgomery（1997, p682），已將資料轉換為介於 0 和 1 之間的數值

＜附錄 3＞

序列檢定程式

```
ss_function(serial,d=3,k=3){
   r_NULL
   w_rep(10,d)^c((d-1):0)
   p_rep(0,length(serial))
   for (i in 0:(k-1))    p_p+(serial>(i/k))
   while (length(p)>=d){
    y_p[1:d]
    p_p[-(1:d)]
    q_sum(y*w)
    r_c(r,q)
   }
    obs_table(r)
    exp_sum(obs)/(k^d)
    chisq_sum((obs-exp)^2/exp)
   pv_1-pchisq(chisq,length(obs)-1)
   pv
   }
```

排列檢定程式

```
permu_function(serial,k=3){
   permu_NULL
   y_rep(10,k)^c((k-1):0)
   while (length(serial)>=k){
      x_serial[1:k]
      serial_serial[-(1:k)]
      r_rank(x)
      p_sum(r*y)
      permu_c(permu,p)
                        }
      obs_table(permu)
      exp_sum(obs)/length(obs)
      chisq_sum((obs-exp)^2/exp)
   pv_1-pchisq(chisq,length(obs)-1)
   pv
   }
```

間隔檢定

```
gap_function(serial,L=0,U=0.5,k=6){
   gap_NULL
   obs_NULL
   p_NULL
   no_0
   TF_(L<serial & serial<U)
   for ( i in 1:length(TF) ){
    x_TF[i]
    if (x==1) {
       gap_c(gap,no)
       no_0
                  }
    else no_no+1
   }
   for (i in 0:(k-1)) {
      obs_c(obs,sum(gap==i))
      p_c(p,(1-(U-L))^i*(U-L))
   }
      obs_c(obs,sum(gap>=k))
      p_c(p,(1-U+L)^k)
      exp_sum(obs)*p
      chisq_sum((obs-exp)^2/exp)
   pv_1-pchisq(chisq,length(obs)-1)
   pv
   }
```

上升趨勢檢定

```
runup_function(serial,k=4){
 original_serial
 serial_c(serial,0)
 sign_NULL
 p_NULL
 obs_NULL
 no_1
 jump_0
 for ( i in 2:length(serial) ) {
  if (jump==0) {
```

```
    if (serial[i]>serial[i-1]) no_no+1
    else {
      sign_c(sign,no)
      no_1
      jump_1
    }
  }
  else jump_0
 }

for ( i in 1:(k-1)){
    obs_c(obs,sum(sign==i))
    p_c(p,i/gamma(i+2))
    }
  obs_c(obs,sum(sign>=k))
  p_c(p,1-sum(p))
    exp_sum(obs)*p
    chisq_sum((obs-exp)^2/exp)
  pv_1-pchisq(chisq,length(obs)-1)
  pv
}
```

升降趨勢檢定

```
run_function(serial){
  x_serial
  x1_((x[-c(1)]-x[-c(length(x))])>=0)*1
  x2_abs(x1[-c(1)]-x1[-c(length(x1))])
  x3_sum(x2)+1
  n_length(x1)
  x4_(x3-(2*n-1)/3)/sqrt((16*n-29)/90)
  pv_(1-pnorm(abs(x4)))*2
  pv
}
```

相關檢定

```
correlation_function(serial,lag=3){
      pvalue_NULL
      aa_NULL
    for (j in 1:lag){
       h_floor((length(serial)-1)/j)-1
       U_na.omit(serial[(1:length(serial))*j-j+1])[,1]
       U1_U[-length(U)]
       U2_U[-1]
      rho_(12/(h+1))*sum(U1*U2)-3
       a_rho/sqrt((13*h+7)/((h+1)^2))
       pv_(1-pnorm(abs(a)))*2
       pvalue_c(pvalue,pv)
       aa_c(aa,a)
       }
       min(pvalue)
       }
```

＜附錄 4＞

```
comb500_function(serial,group) {
   com_NULL
   for (i in 1:group) {
      ser_serial[,i]
      com_c(com,
            chisq.gof(ser,distribution="uniform",min=0,max=1)[[3]],
            ks.gof(ser,distribution="uniform",min=0,max=1)[[2]],
            ss(ser,d=2,k=5),ss(ser,d=3,k=3),
            permu(ser,k=3),gap(ser,L=0,U=0.5,k=5),
            runup(ser,k=4),run(ser), correlation (ser))
   }
   matrix(com, nrow=group,byrow=T)
}
```

＜附錄 5＞

```
comb1000_function(serial,group) {
   com_NULL
   for (i in 1:group) {
      ser_serial[,i]
      com_c(com,
            chisq.gof(ser,distribution="uniform",min=0,max=1)[[3]],
            ks.gof(ser,distribution="uniform",min=0,max=1)[[2]],
            ss(ser,d=2,k=5),ss(ser,d=3,k=3),
            permu(ser,k=3),gap500(ser,L=0,U=0.5,k=6) ,
            runup(ser,k=4),run(ser), correlation (ser))
   }
   matrix(com, nrow=group,byrow=T)
}
```