國 立 政 治 大 學 統 計 學 系

碩 士 學 位 論 文

指導教授： 洪英超博士

R 軟體套件"rBeta2009"之評估及應用

Evaluation and Applications of the R Package "rBeta2009"

研究生：劉世璿 撰

中華民國一百零一年六月

謝辭

時光過得飛快，才一眨眼的時間就到了畢業時分，將交出兩年以前連想都不敢想的論文本。在過程中，若不是得到許多人的幫助，自己是不可能完成的；因此，這本碩士學位論文謹獻給幫助我的這許多人。

首先誠摯的感謝指導教授　洪英超老師，老師悉心的指導使我得以一窺 R 軟體的深奧。還記得老師在我迷惘於論文題目時，以「絕非所有研究題目都是自己興趣所在，旨在研究過程所學之事不可磨滅」一句勉勵於我，並且在過程中不時的提點及細心的指導研究及寫作方向，使我在這兩年中獲益匪淺。本論文的完成另外得感謝口委　蔡紋琦老師及　曾能芳老師的大力協助。因為有你們的建議，使得本論文能夠更完整而嚴謹。此外，也感謝鄭經維學長在工作之餘還能指導 R 軟體的操作，對我的研究過程有極大的幫助。

這兩年的時光，撐過了許多難關，必須要感謝研討室諸位同學的陪伴。在研討室過夜趕報告，午餐時的打打鬧鬧，還有遇到問題先煩同學。我們就像個大家庭，有苦同苦，有樂共樂。

最後，謹以此文獻給我最親愛的爸媽。

劉世璿　謹致

中華民國一百零一年六月

摘要

本論文主要是介紹並評估一個 R 的軟體套件叫做"rBeta2009"。此套件是由 Cheng *et al.* (2012) [8] 所設計，其目的是用來產生貝他分配(Beta Distribution)及狄氏分配(Dirichlet Distribution)的亂數。本論文特別針對此套件之(i)有效性(effiniency)、(ii)精確性(accuracy)及(iii)隨機性(randomness)進行評估，並與現有的 R 套件作比較。此外，本論文也介紹如何應用此套件來產生(i)反貝他分配(Inverted Beta Distribution)、(ii)反狄氏分配(Inverted Dirichlet Distribution)、(iii)Liouville 分配及(iv)凸面區域上的均勻分配之亂數。

關鍵字： 狄氏分配、貝他分配、有效性、精確性、隨機性

Abstract

A package in R called "rBeta2009", originally designed by Cheng *et al*. (2012) [8], was introduced and evaluated in this thesis. The purpose of the package is generating beta random numbers and Dirichlet random vectors. In this paper, we not only evaluated (i) the efficiency, (ii) the accuracy and (iii) the randomness, but also compare it with other R packages currently in use. In addition, it was also scrutinized in this thesis how to generate (i) inverted beta random numbers, (ii) inverted Dirichlet random vectors, (iii) Liouville random vectors, and (iv) uniform random vectors over convex polyhedron by using the same package.

Key words: beta variates, Dirichlet random vectors, efficiency, accuracy, randomness.

目錄

表目錄

圖目錄

第一章　導論

貝他分配及狄式分配的應用層面很廣泛。貝他分配常被用在隨機模型及模擬、計畫評核術(PERT)、要逕法(CPM)及其他的計劃管理方法(詳細請見 Hung *et al.* 2009 [13])。狄氏分配最常用的是在貝氏分析(Bayesian inference)時可以做為多項分配及類別分配(categorical distribution)的共軛先驗分配(conjugate prior distribution) [21]，由於貝他分配是狄氏分配的特例，故也常用於貝氏分析當作二項分配及幾何分配的共軛先驗分配；且狄氏分配可當做類別變數或是多項分配的貝式混合模型的先驗分配[38]。狄式分配也被證明了在對文字文件中的單字分配建模時特別有用[25]。而在生物學方面，當模型序列屬於隱藏式馬可夫模型[31]或是對偶基因頻率時，狄氏分配常用於表示胺基酸的比例[22]。而貝他分配及狄式分配還常被用在秩序統計量。除了上述應用之外，還可以用在生成反貝他分配(Inverted Beta distribution)、反狄式分配(Inverted Dirichlet distribution) [35, 36]、Liouville 分配[23, 28]、凸面區域上的均勻分配[10]等等。標準貝他分配 $Beta(\alpha,\beta)$ 的機率密度函數為

$$f(x)=\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1} \tag{1}$$

其中 $0\le x\le 1$ 且參數 $\alpha,\beta>0$ 。如果一個 k 維度的隨機向量 $(Y_1,\dots,Y_k)$ 服從狄氏分配 $Dirichlet(\alpha_1,\dots,\alpha_k;\alpha_{k+1})$，那麼它的聯合機率密度函數就是

$$f(y_1,\dots,y_k)=\frac{\Gamma(\alpha_1+\cdots+\alpha_k)}{\Gamma(\alpha_1)\cdots\Gamma(\alpha_k)}y_1^{\alpha_1-1}\cdots y_k^{\alpha_k-1}\left(1-\sum_{i=1}^{k}y_i\right)^{\alpha_{k+1}-1} \tag{2}$$

其中 $y_i\ge 0$ 對 $i=1,\dots,k,\sum_{i=1}^{k}y_i\le 1$ 以及參數 $\alpha_1,\dots,\alpha_{k+1}>0$ 。

近年來，由於在電腦軟體中各種產生貝他及狄氏分配亂數的演算法皆已老舊，但舊的演算法目前還是廣為大眾使用。以 R 軟體為例，產生貝他分配亂數的是預設套件"stats"中的 rbeta()程式，所用的演算法是 Cheng (1978) [7]所提出；而 SAS 軟體則是用 rand()來生成貝他亂數，所用的演算法是使用由 Atkinson 和 Whittaker (1976) [1]及 Cheng (1978)所提出；Matlab 軟體中的 Matlab Statistics

Toolbox 則是使用 betarnd()來生成貝他亂數，所用的演算法是使用秩序統計量法及 Jöhnk's 法[19]。但以上演算法之速度已經比不上之後提出的其他演算法(見 Hung *et al.* 2009 [13])。而產生狄氏分配亂數的是套件"MCMCpack"及"gtools"中的 rdirichlet()程式，兩者使用的程式碼皆從 Greg's Miscellaneous Functions [37]而來，是使用 Gamma 分配的性質轉換而來。因此，Hung *et al.* (2009,2011) [13, 14]提出了新的方針，其想法主要是根據參數的範圍來選擇最有效率的演算法。之後，Cheng *et al.* (2012)根據這些方針發展出用來產生貝他分配及狄氏分配亂數的 R 套件—"rBeta2009" [8]。

本論文的目的主要是評估由 Cheng *et al.* (2012)設計用來產生貝他及狄式分配亂數的 R 套件—"rBeta2009"。我們會針對其有效性、精確性及隨機性做評估。評估有效性的作法為計算亂數生成的 CPU 時間，我們將會對"rBeta2009"套件與"stats"及"MCMCpack"的 CPU 時間在 32 位元及 64 位元處理器的環境下做比較。評估精確性的做法是根據統計對等區集(Statistically Equivalent Blocks，簡稱 SEB) [2, 4, 9, 11, 34]的概念執行所謂的適合度檢定(Goodness-of-fit test)。評估隨機性的做法是對其生成亂數做隨機性檢定。本文中主要使用 Ljung-Box 檢定[15, 16, 24]的方法來檢定其隨機性。

本文章節架構如下：第二章介紹"rBeta2009"套件所使用的演算法。第三章將測試不同處理器下"rBeta2009"套件的執行速度並與現有 R 套件做比較。此外，我們也對其產生亂數做適合度檢定與隨機性檢定。第四章將介紹"rBeta2009"套件之其他應用，內容主要是如何用貝他分配或狄式分配去產生反貝他分配、反狄式分配、Liouville 分配及凸面區域上的均勻分配。第五章為論文的總結。

第二章　"rBeta2009"套件之介紹

本章將介紹"rBeta2009"套件中，用來生成貝他分配所使用的方針，以及生成狄式分配亂數所使用的方針與演算法。

第一節　生成貝他分配之演算法

對於貝他分配 $Beta(\alpha,\beta)$，Hung *et al.* (2009)提出三點方針，根據貝他分配的參數大小來選擇生成貝他分配亂數最快速的演算法。以下是其方針：

- 方針一：當 $\alpha,\beta<1$ ，若 $\alpha+\beta>1.2$ ，則使用演算法 Kennedy's MK [20]；其餘則使用演算法 B00 [30]；
- 方針二：當 $\alpha<1<\beta$ 或 $\alpha>1>\beta$ ，則使用演算法 B01 [30]；
- 方針三：當 $\alpha,\beta>1$ ，若其中一個參數很接近 1 ，且另一個參數很大(例如 >4)，則使用演算法 B4PE [32]；其餘則使用演算法 BPRS [39]。

"rBeta2009"套件使用以上三個方針來產生貝他分配亂數。根據以上方針，就能利用演算法 B00、B01、B4PE 及 BPRS 的特性，以最快的時間生成各種參數的貝他分配亂數。值得一提的是，Kennedy's MK 演算法為一"近似"貝他分配的隨機搜尋方法，其產生的亂數有較大的不確定性(或誤差)，所以在"rBeta2009"套件中，此演算法被 B00 所取代。

第二節　生成狄氏分配之演算法

狄式分配可以由貝他分配經過轉換而得到。若 $X_1,\dots,X_k$ 為互相獨立的隨機變數服從貝他分配 $X_i \sim Beta\left(\alpha_i, \sum_{j=i+1}^{k} a_j\right),\ i=1,\dots,k$，令 $Y_1 = X_1,\ Y_i = X_i \prod_{j=1}^{i-1}\left(1-X_j\right),\ i=2,\dots,k$，則 $(Y_1,\dots,Y_k)$ 服從所謂的狄式分配，並記為 $Dirichlet(\alpha_1,\dots,\alpha_k;\alpha_{k+1})$。為了使此轉換過程速度更快，Hung *et al.* (2011) 提出了以下三個方針：

- 方針一：根據參數 α_i 選擇最快的演算法來生成貝他分配亂數。(例如，使用第一節介紹的方針來選擇演算法。)
- 方針二：將狄氏分配的參數重新排序。其想法是將原本的參數 $\alpha_1,\dots,\alpha_{k+1}$ 由小到大排序，稱為 $\alpha_{(1)},\dots,\alpha_{(k+1)}$，並令 $m\ (\leq k+1)$ 為參數中小於 1 的個數。
  - 若 $\alpha_{(k+1)} < 1$ 且 $k+1$ 是奇數，則生成

    $$Dirichlet(\alpha_{(1)},\alpha_{(3)},\dots,\alpha_{(k-1)},\alpha_{(k+1)},\alpha_{(k)},\alpha_{(k-2)},\alpha_{(k-4)},\dots,\alpha_{(4)},\alpha_{(2)})$$

    隨機向量；
  - 若 $\alpha_{(k+1)} < 1$ 且 $k+1$ 是偶數，則生成

    $$Dirichlet(\alpha_{(1)},\alpha_{(2)},\alpha_{(4)},\dots,\alpha_{(k-1)},\alpha_{(k+1)},\alpha_{(k-2)},\alpha_{(k-4)},\dots,\alpha_{(5)},\alpha_{(3)})$$

    隨機向量；
  - 若 $1 \leq \alpha_{(k+1)} \leq 3$ 且 $\alpha_{(1)} > 1$，則生成

    $$Dirichlet(\alpha_{(k+1)},\alpha_{(k-1)},\alpha_{(k-2)},\dots,\alpha_{(1)},\alpha_{(k)})$$

    隨機向量；
  - 若 $\alpha_{(k+1)} > 3$ 且 $\alpha_{(1)} \geq 1$，則生成

    $$Dirichlet(\alpha_{(k+1)},\alpha_{(k)},\alpha_{(k-2)},\alpha_{(k-3)}\dots,\alpha_{(1)},\alpha_{(k-1)})$$

    隨機向量；
  - 若 $\alpha_{(k+1)} \geq 1, \alpha_{(1)} < 1, m > \frac{k+2}{2}$ 且 $\alpha_{(m)} \leq 0.5$，則生成

    $$Dirichlet(\alpha_{(1)},\alpha_{(2)},\dots,\alpha_{(m-1)},\alpha_{(m+1)},\alpha_{(m+2)},\dots,\alpha_{(k+1)},\alpha_{(m)})$$

隨機向量；

- 其餘則生成

$$Dirichlet(\alpha_{(k+1)}, \alpha_{(k-1)}, \alpha_{(k-2)}, \dots, \alpha_{(1)}, \alpha_{(k)})$$

隨機向量。

➢ 方針三：減少乘法的運算次數。以 $Y_i = X_i(1 - \sum_{j=1}^{i-1} Y_j)$ 代替原本的 $Y_i = X_i \prod_{j=1}^{i-1}(1 - X_j)$，減少乘法的運算次數，加快運算速度。

套件"rBeta2009"使用以上三個方針來生成狄式分配亂數；並且由 Hung *et al.* (2011) 將以上整理成演算法，稱為 BETA-M。

BETA-M 演算法

步驟一：根據上述的方針二重新排序狄式分配的參數 $\alpha_1, \dots, \alpha_{k+1}$，定為 $\alpha_{s_1}, \dots, \alpha_{s_{k+1}}$ ；

步驟二：以方針一產生獨立的貝他分配 $X_{s_i} \sim Beta\left(\alpha_{s_i}, \sum_{j=i+1}^{k+1} \alpha_{s_j}\right)$, $i = 1, \dots, k$ ；

步驟三：令 $Y_{s_1} = X_{s_1}$ 及 $Y_{s_i} = X_{s_i}\left(1 - \sum_{j=1}^{i-1} Y_{s_j}\right), i = 2, \dots, k$ ；

步驟四：根據步驟一的排序將 $Y_{s_1}, \dots, Y_{s_k}$ 排回 $Y_1, \dots, Y_k$ 並輸出。

第三節　"rBeta2009"之指令執行

本節將介紹如何使用 R 軟體套件"rBeta2009"中的指令 rbeta()及 rdirichlet()。程式 rbeta(n, shape1, shape2)是以本章第一節所述之演算法，生成 n 個貝他分配 $Beta(\text{shape1}, \text{shape2})$ 亂數。其中 n 為生成的亂數個數，若長度大於 1 (length(n)>1)，則會以 n 的維度個數當作生成的亂數個數；而 shape1 及 shape2 為正形狀參數(shape parameter)。要注意的是，R 軟體的預設套件"stats"中有同名的程式，但只需讀取"rBeta2009"套件之後，在套件"stats"中的同名程式就能被取代。以下會舉例介紹如何使用"rBeta2009"套件中的 rbeta()。首先，將"rBeta2009"套件讀入 R 軟體並隨機設一個種子(set a random seed)：

```
R> library("rBeta2009")
R> set.seed(63522)
```

接著我們生成 10 個貝塔分配 $Beta(1.5, 0.3)$ 亂數：

```
R>rbeta(10, 1.5, 0.3)

[1] 0.7461113 0.8843041 0.9960109 0.9968034 0.2236435 0.5429963 0.5130998
[8] 0.9999988 0.9817266 0.7213977
```

生成 10 個貝塔分配 $Beta(4.2, 13)$ 亂數：

```
R > rbeta(10, 4.3, 13)

[1] 0.1456586 0.2540797 0.4101915 0.2163145 0.1844294 0.4010572 0.1043971
[8] 0.2315380 0.2456694 0.2543450
```

而程式 rdirichlet(n, shape)是以本章第二節之演算法生成 n 個 k 維度的狄氏分配亂數，其中 shape 為一個長度為 $k+1$ 正的形狀參數向量($k \geq 2$)，而 n 為生成的亂數個數，若長度大於 1 (length(n)>1)，則會以 n 的長度當作生成的亂

數個數。以下舉例介紹如何使用"rBeta2009"套件中的 rdirichlet()。生成 10 個狄氏分配 $Dirichlet(0.1, 0.5, 3, 1.7)$ 亂數($k = 3$)：

```
R> library("rBeta2009")
R> set.seed(39208)
R>rdirichlet(10, c(1.5, 0.5, 3, 1.7))
```

	[,1]	[,2]	[,3]	[,4]
[1,]	0.08085422	0.1004126229	0.4687021	0.3500311
[2,]	0.49012024	0.0667342799	0.1383359	0.3048096
[3,]	0.12066829	0.0073373939	0.6511637	0.2208306
[4,]	0.03876349	0.1528865212	0.4594006	0.3489494
[5,]	0.47923274	0.0946464104	0.1024129	0.3237080
[6,]	0.45851938	0.0432629126	0.2855953	0.2126224
[7,]	0.16739973	0.0140936809	0.4813785	0.3371281
[8,]	0.15519533	0.3961587541	0.2815362	0.1671098
[9,]	0.31466787	0.0004720956	0.3786146	0.3062454
[10,]	0.17986964	0.1555505958	0.3942865	0.2702932

若有非正參數在 shpae 向量中出現，則會出現警告：

```
R>rdirichlet(10, c(-1.5, 0.5, 3, 1.7))
```

錯誤在 rdirichlet(10, c(-1.5, 0.5, 3, 1.7)) :
  Shape parameters should be all positive

而若是 shape 向量長度為 2 時，會出現警告，但程式仍可以執行：

```
R> rdirichlet(10, c(3, 0.5))
```

[1] 0.9998994 0.9994591 0.8965159 0.9985557 0.8156766 0.9676691 0.9319676
[8] 0.9593898 0.7713237 0.7098055
警告訊息：
In rdirichlet(10, c(3, 0.5)) :
  'shape' is of length 2: reduced to beta variates

第三章　"rBeta2009"套件評估

本章將會對 R 套件"rBeta2009"中的 rbeta()及 rdirichlet()程式做測試評估。測試項目為：(一) 有效性(efficiency)；(二) 準確性(accuracy)；(三) 隨機性(randomness)。本文測試程式效率用的電腦設備為 Windows 7(家用進階型) 64 位元，2.10 GHz AMD Phenom(tm)II N950 Quad-Core 處理器，6.00GB 記憶體。

第一節　有效性 (Efficiency)

本節將實際模擬證明 R 套件"rBeta2009"所花的 CPU 時間較現有 R 套件("stats"及"MCMCpack")來的少。由於"rBeta2009"套件的設計在 32 位元及 64 位元的處理器環境內皆能使用，故本節將會在此兩種環境下做測試。

(表 1)是計算R程式rbeta()產生 3×10^6 個貝他分配亂數的CPU時間 (單位：秒)及增進比例 (Proportion of Improvement，新套件所減少的 CPU 時間與舊套件所花 CPU 時間的比值)，分別在我們的"rBeta2009"套件及 R 軟體內的預設套件"stats"下執行。從(表 1)可以看到，所有的參數的 CPU 時間都有頗高的增進比例。在 64 位元處理器環境的部分，兩個參數都較大的時候增進比例較高，參數組合較大的增進比例幾乎都超過了 70%；而在 32 位元處理器環境的部分，除了參數組合為 $(0.1, 0.1)$ 時增進比例為 50%，其餘參數組合的增進比例都超過了 60%，且參數組合 $(0.5, 0.5)$ 及 $(5, 5)$ 的增進比例甚至到了 77.59%及 77.51%。另外在處理器部分，根據(表 1)我們發現"stats"套件的 CPU 時間在 64 位元處理器環境時表現較好，反之"rBeta2009"套件則在 32 位元處理器環境時表現略好，尤其在參數小時較為明顯。

(表 2)及(表 3)為"rBeta2009"套件及"MCMCpack"套件分別在 64 位元及 32 位元處理器環境下計算 R 程式 rdirichlet()產生 3×10^6 個狄式分配亂數的 CPU 時間(單位：秒)及增進比例(同上段所述)。由(表 2)可以看到，在 64 位元處理器環境的部分，除了參數組合 $(0.1, 0.1)$ 的增進比例較低為 20.06%，其餘皆在 34%以上。表內有打"*"號的為增進比例超過 60%的，其中以參數組合

(表 1) R 套件"rBeta2009"及"stats"生成 3×10^6 個貝他分配亂數 $Beta(\alpha,\beta)$ 的 CPU 時間(秒)及增進比例(Proportion of Improvement)。

		rbeta()		Proportion of
α	β	rBeta2009	stats	Improvement
64 位元處理器				
0.1	0.1	1.46	1.99	26.63%
	0.5	1.08	2.00	46.00%
	5	0.88	1.90	53.68%
	10	0.83	1.84	54.89%
	100	0.91	1.88	51.60%
0.5	0.5	1.06	1.91	44.50%
	5	1.11	2.00	44.50%
	10	1.12	1.95	42.56%
	100	1.25	2.01	37.81%
5	5	0.39	1.59	75.47%
	10	0.43	1.54	72.08%
	100	0.45	1.47	69.39%
10	10	0.44	1.60	72.50%
	100	0.42	1.58	73.42%
100	100	0.45	1.64	72.56%
32 位元處理器				
0.1	0.1	1.10	2.20	50.00%
	0.5	0.73	2.30	68.26%
	5	0.64	2.21	71.04%
	10	0.66	2.17	69.59%
	100	0.59	2.20	73.18%
0.5	0.5	0.47	2.09	77.51%
	5	0.68	2.20	69.09%
	10	0.75	2.25	66.67%
	100	0.85	2.28	62.72%
5	5	0.39	1.74	77.59%
	10	0.43	1.75	75.43%
	100	0.50	1.73	71.10%
10	10	0.44	1.73	74.57%
	100	0.42	1.68	75.00%
100	100	0.44	1.81	75.69%

$(10, 10, 10, 100, 100)$ 為最高 62.58%。而由(表 3)可以看到，在 32 位元處理器環境的部分，除了參數組合 $(0.1, 0.1, 0.1, 0.1, 0.1)$ 的增進比例較低為 38.71%，其餘大部分都大於 60%，而其中增進最多為參數組合 $(0.5, 0.5, 0.5, 0.5, 0.5)$，竟達到 78.34%。綜合(表 2)及(表 3)，我們發現"MCMCpack"套件的 rdirichlet()與"stats"套件的 rbeta()一樣，在 64 位元處理器環境的表現較好，而"rBeta2009"套件則在 32 位元處理器環境表現較好，因此其增進比例也高上許多。

(表 2) 在 64 位元處理器下以 R 套件"rBeta2009"及"MCMCpack"生成 3×10^6 個狄氏分配亂數 $Dirichlet(\alpha_1, \alpha_2, \alpha_3, \alpha_4; \alpha_5)$ 的 CPU 時間(秒) 及增進比例 (Proportion of Improvement)。

$\alpha_1 = \alpha_2 = \alpha_3$	$\alpha_4 = \alpha_5$	rdririchlet()		Proportion of Improvement
		rBeta2009	MCMCpack	
0.1	0.1	5.38	6.73	20.06%
	0.5	3.84	7.11	45.99%
	5	3.21	5.98	46.32%
	10	3.21	6.02	46.68%
	100	3.15	5.85	46.15%
0.5	0.1	3.99	7.22	44.74%
	0.5	3.81	7.82	51.28%
	5	3.96	6.88	42.44%
	10	4.04	6.63	39.06%
	100	4.32	6.58	34.35%
5	0.1	2.66	5.66	53.00%
	0.5	3.13	6.09	48.60%
	5	1.98	4.99	*60.32%
	10	1.90	4.94	*61.54%
	100	1.87	4.87	*61.60%
10	0.1	2.71	5.62	51.78%
	0.5	3.31	5.93	44.18%
	5	1.90	4.92	*61.38%
	10	1.90	4.69	59.49%
	100	1.80	4.81	*62.58%
100	0.1	2.73	5.42	49.63%
	0.5	3.48	5.88	40.82%
	5	2.01	4.75	57.68%
	10	1.90	4.71	59.66%
	100	1.87	4.68	*60.04%

(表 3) 在 32 位元處理器下以 R 套件"rBeta2009"及"MCMCpack"生成 3×10^6 個狄氏分配亂數 $Dirichlet(\alpha_1, \alpha_2, \alpha_3, \alpha_4; \alpha_5)$ 的 CPU 時間(秒) 及增進比例 (Proportion of Improvement)。

$\alpha_1 = \alpha_2 = \alpha_3$	$\alpha_4 = \alpha_5$	rdririchlet()		Proportion of Improvement
		rBeta2009	MCMCpack	
0.1	0.1	4.67	7.62	38.71%
	0.5	3.11	8.06	61.41%
	5	2.50	6.99	64.23%
	10	2.53	6.85	63.07%
	100	2.59	6.70	61.34%
0.5	0.1	2.95	8.33	64.59%
	0.5	1.93	8.91	*78.34%
	5	2.79	7.56	63.10%
	10	2.90	7.63	61.99%
	100	3.20	7.50	57.33%
5	0.1	2.39	6.62	63.90%
	0.5	2.47	7.16	65.50%
	5	1.92	5.90	67.46%
	10	1.86	6.03	69.15%
	100	1.94	5.91	67.17%
10	0.1	2.35	6.50	63.85%
	0.5	2.59	7.04	63.21%
	5	1.86	5.99	68.95%
	10	1.92	5.82	67.01%
	100	1.87	5.77	67.59%
100	0.1	2.40	6.38	62.38%
	0.5	2.71	7.00	61.29%
	5	1.82	5.87	68.99%
	10	1.93	5.82	66.84%
	100	1.90	5.54	65.70%

第二節　準確性 (Accuracy)

本小節將會對"rBeta2009"套件的 rbeta()及 rdirichlet()所產生的亂數，分別以 Kolmogorov-Smirnov(K-S)檢定及多變量的 Kolmogorov-Smirnov 檢定做適合度檢定，以確定此亂數產生器所產生的亂數是否符合貝他分配及狄氏分配。多變量的 Kolmogorov-Smirnov 檢定所用到的檢定程序是依照統計對等區集(Statistically Equivalent Blocks，簡稱 SEBs)的概念來執行[2, 4, 9, 11, 34]。以下會解釋此檢定如何運作。

假設 $x_1, \dots, x_n$ 為一組 n 個 k 維度的狄式分配亂數觀測值， F 及 S 分別為其累積機率密度函數及其定義域。此方法首先選擇 n 個分配為連續的分割函數 $\phi_1, \dots, \phi_n$。接著定義$x_{(1)} = arg\,min_{x\in\{x_1,\dots,x_n\}} \phi_1(x)$，如此可以將定義域$S$切成

$B_1 = \{x \in S: \phi_1(x) \le \phi_1(x_{(1)})\}$ 及 $B_{2\dots n} = S \backslash B_1$ 兩塊；

接著定義 $x_{(2)} = arg\,min_{x\in\{x_1,\dots,x_n\}\backslash x_{(1)}} \phi_2(x)$，如此可將 $B_{2\dots n}$ 再切成

$B_2 = \{x \in B_{2\dots n}: \phi_2(x) \le \phi_2(x_{(2)})\}$ 以及 $B_{3\dots n} = B_{2\dots n} \backslash B_2$ 兩塊。

依此類推，持續做到將定義域 S 切成不重疊的 $n+1$ 塊 $B_1, \dots, B_n, B_{n+1}$，且 $\bigcup_{i=1}^{n+1} B_i = S$。令 $d_i = P_F(X \in B_i), i = 1, \dots, n$，Anderson (1966) [4]證明了 $(d_1, \dots, d_n)$ 服從定義在 n 維度的 simplex 內之均勻分配。將

$$\widetilde{D}_n = max\left\{0, \underset{1\le j\le n}{max}\{\frac{j}{n} - D_j\}, \underset{1\le j\le n}{max}\{D_j - \frac{j-1}{n}\}\right\} \tag{3}$$

定義為多變量的 Kolmogorov-Smirnov 統計量，其中 $D_j = P_F\left(X \in \bigcup_{i=1}^{j} B_i\right) = \sum_{i=1}^{j} d_i, j = 1, \dots, n$。令虛無假設為 $H_0: x_1, \dots, x_n$ 服從 F 的分配。根據這些定義，當 $\widetilde{D}_n$ 值太大時我們將拒絕虛無假設。以下由 Serfling (1980) [29]導出近似的結果可以幫助我們做所謂的適合度檢定：

$$\lim_{n\to\infty} P(\sqrt{n}\widetilde{D}_n \le d) = 1 - 2\sum_{j=1}^{\infty} (-1)^{j+1} e^{-2j^2d^2}, d > 0 \tag{4}$$

根據式(4)，若我們重複產生 n 個觀察值 N 次，並把 d 定為 N 個 $\sqrt{n}\widetilde{D}_n$ 觀

測值的第 95 個百分位數，則式(4)在虛無假設下就會非常接近 0.95。

值得一提的是，在計算 $D_j = P_F\left(X \in \bigcup_{i=1}^{j} B_i\right) = \sum_{i=1}^{j} d_i, j = 1, \ldots, n$ 時，本文並非使用單個 d_i 加總，而是利用 $1 - P_F\left(X \notin \bigcup_{i=1}^{j} B_i\right)$的公式。其詳細作法如下：

$$P_F\left(X \notin \bigcup_{i=1}^{j} B_i\right) = \bar{F}(x_1, \ldots, x_k)$$

$$= \begin{cases} 1 - \sum_{j=1}^{k} (-1)^j \sum_{1 \le i_1 \le \cdots \le i_j \le k} F_{X_{i_1}, \ldots, X_{i_j}}\left(x_{i_1}, \ldots, x_{i_j}\right), & if\ x_1 + \cdots + x_k \le 1 \\ 0, & if\ x_1 + \cdots + x_k > 1 \end{cases} \quad (5)$$

而 $F(x_1, \ldots, x_k)$ 在 $X \sim Dirichlet(a_1, \ldots, a_k; a_{k+1})$ 分配時的計算方法有兩種。第一種是用狄式分配的機率密度函數做多重積分，範圍是 $X \in S \setminus (\bigcup_{i=1}^{j} B_i)$。此法可用 R 軟體的"R2Cuba"套件中的 cuhre()程式來執行[11]，是屬於所謂決定型演算法(deterministic algorithm) [17, 18]。第二種是使用 Ashraf 和 Tamás 在 2009 年[3]提出的狄式分配累積機率密度函數。原則上第二種方法速度較快，但有一些特例無法執行，其定理如下：

(定理 1) 若 $(X_1, \ldots, X_k) \sim Dirichlet(a_1, \ldots, a_k; a_{k+1})$ ，則其累積機率密度函數為

$$F(x_1, \ldots, x_k; a_1, \ldots, a_k; a_{k+1}) = \frac{\Gamma(a_1 + \cdots + a_k + a_{k+1})}{\Gamma(a_1) \cdots \Gamma(a_k)\Gamma(a_{k+1})} \frac{x_1^{a_1}}{a_1} \cdots \frac{x_k^{a_k}}{a_k} \cdot F_A^{(n)}(1 - a_{k+1}, a_1, \ldots, a_k; a_1 + 1, \ldots, a_k + 1; x_1, \ldots, x_k), \quad (6)$$

其中 $F_A^{(n)}$ 為 A 型(第一型)的 Lauricella 函數，若 $x_1 + \cdots + x_k \le 1, x_1, \ldots, x_k \ge 0$，則

$$F_A^{(n)}(\mathrm{a}, b_1, \ldots, b_k; c_1, \ldots, c_k; x_1, \ldots, x_k) = \sum_{m_1, \ldots, m_k = 0}^{\infty} \frac{(\mathrm{a}, m_1 + \cdots + m_k)(b_1, m_1)(b_k, m_k)}{(c_1, m_1) \cdots (c_k, m_k)} \frac{x_1^{m_1}}{m_1!} \cdots \frac{x_k^{m}}{m_k!}, \quad (7)$$

式 (7) 中符號 $(\cdot,\cdot)$ 定義為

$$(\lambda,\mu)=\begin{cases}\dfrac{\Gamma(\lambda+\mu)}{\Gamma(\lambda)}=\lambda(\lambda+1)\cdots(\lambda+\mu-1), & if\ \mu>0,\\ \dfrac{(-1)^{-\mu}}{(1-\lambda,-\mu)}, & if\ \mu<0,\\ 1, & if\ \mu=0.\end{cases} \tag{8}$$

(注意 1) 由於 $\Gamma(\cdot)$ 的定義域不包含 0 及負整數，故在 Lauricella 函數中計算 $\Gamma(1-a_{k+1})$ 時，會有以下情況無法使用 Lauricella 函數配合 $1-\bar{F}(x_1,\ldots,x_k)$ 的方法來計算狄氏分配的累積機率：

情況一： $k\leq 3$ 時，若 $\alpha_{k+1}\in\mathbb{N}$。

情況二： $k>3$ 時，若 $\alpha_i\in\mathbb{N}, i=3,\ldots,k+1$。

以 $k=3$ 為例，若要計算 $Dirichlet(2,3,5)$ 的累積機率分配，則會出現發散的情況(如 $\Gamma(1-5)$)，故無法使用 Lauricella 函數；但若為 $Dirichlet(2,3,0.7)$ 的累積機率分配，則可以使用。而以 $k=5$ 為例，若要計算 $Dirichlet(2,3,5,0.3,0.9)$ 的累積機率分配，則也會出現發散情形(如 $\Gamma(1-5)$)，無法使用 Lauricella 函數；反之若為 $Dirichlet(0.1,0.3,0.5,0.7,0.9)$ 則可。

實際操作 Kolmogorov-Smirnov 檢定時，我們的分割函數是使用 Tukey (1947) [34]建議的 $\phi_{i+rk}(x)=x_i', i=1,\ldots,k, r=0,1,\ldots$，其中 x_i' 是 x 向量的第 i 個值；如此我們就能將上述統計對等區塊切成簡易的矩形(或長方體)，如此計算較為容易。接著我們產生 200 組貝他分配及狄氏分配的亂數並重複 1000 次，結果如(表 4)及(表 5)所示。由(表 4)及(表 5)可以發現所有的估計機率值都與 0.95 很接近，這表示"rBeta2009"所產生之亂數皆服從貝他分配及狄氏分配。

(表 4) 生成 200 個貝他分配亂數重複 1000 次，計算其式(4)的估計值。

rbeta()	
Parameters	$\hat{P}\left(\sqrt{200}\widetilde{D}_{200} \le d\right)$
$(0.1, 0.1)$	0.965
(0.1,0.7)	0.952
$(0.9, 0.9)$	0.956
(0.3,10)	0.942
(3,5)	0.948
$(1, 10)$	0.949
$(10, 10)$	0.950
$(100, 100)$	0.949

(表 5) 產生 200 個狄氏分配亂數重複 1000 次，計算其式(4)的估計值。其中標有 $*$ 之參數組合，計算其累積機率是使用 Lauricella 函數方法。

rdirichlet()	
Parameters	$\hat{P}\left(\sqrt{200}\widetilde{D}_{200} \le d\right)$
$(0.1, 0.5, 0.9)*$	0.940
$(0.1, 0.5, 7)$	0.939
$(2, 3, 5)$	0.945
$(2, 3, 0.7)*$	0.956
$(0.1, 0.3, 0.5, 0.7, 0.9)*$	0.957
$(2, 3, 5, 0.3, 0.9)$	0.949
$(1, 1, 2, 2, 3)$	0.961

第三節　隨機性 (Randomness)

對於一個亂數產生器來說，具備所謂的隨機性是很重要的一件事。本節將對"rBeta2009"套件中 rbeta()及 rdirichlet()程式產生的亂數做隨機性檢定。本文中使用 Ljung-Box 統計量來做亂數的隨機性檢定。由於檢定隨機性與檢定自我相關(Auto-correlation)值是否為 0 一樣，故 Ljung-Box 統計量是由自我相關係數平方加總的概念組成，以下介紹此檢定的過程。

在對單維的亂數做檢定時，我們令 H_0：x_t 與 $x_{t-1},\dots,x_{t-s}$ 之間的相關係數為 0 $(t=s+1,\dots,n)$。本文使用由 Ljung and Box (1978) [24]提出的統計量(稱為 Portmanteau 統計量)：

$$LB(s)=n(n+2)\sum_{j=1}^{s}\frac{r_j^2}{n-j}\sim\chi_s^2 \tag{9}$$

其中

$$r_j=\sum_{t=j+1}^{n}x_t x_{t-j}\Big/\sum_{t=1}^{n}x_t^2 \tag{10}$$

為隨機亂數 $x_1,\dots,x_n$ 的期差 j 期自我相關係數 (the autocorrelation at lag j)，而 s 為想檢定的落後期數(Lag)。則 Portmanteau 統計量 $LB(s)$ 在 n 夠大時就會近似自由度為 s 的卡方分配。在顯著水準 α 之下，若 $LB(s)>\chi_{1-\alpha,s}^2$ ，則拒絕 H_0。我們用"rBeta2009"套件的 rbeta()程式，生成 200 個亂數，重複 1000 次，計算其在顯著水準 $\alpha=0.05$ 之下，拒絕虛無假設的比例。我們計算 $s=1,\dots,6$ 時的拒絕比例，結果如(表 6)，拒絕比例皆接近 0.05，表示此亂數生成器具有隨機性。

在對多維的亂數做隨機性檢定時，我們令虛無假設為觀察值之間互相獨立(也就是觀察值之間的相關係數為 0)。令 $x_t=\left(x_t^1,\dots,x_t^k\right)'$，為了檢定 x_t 與 $x_{t-1},\dots,x_{t-s}$ 之間的相關係數是否為 0 $(t=s+1,\dots,n)$，我們使用由 Hosking (1980) [16]提出的多維度 Portmeanteau 統計量：

$$LB_k(s) = n(n+2)\sum_{j=1}^{s}\frac{1}{n-j}trace\{C_{0j}C_{00}^{-1}C_{0j}'C_{00}^{-1}\} \sim \chi^2_{k^2 s}, \quad (11)$$

其中

$$C_{0j} = \frac{1}{n}\sum_{t=j+1}^{n} x_t x_{t-j}', \quad (12)$$

$x_t = \left(x_t^1, \ldots, x_t^k\right)', t = 1, \ldots, n, s$ 為我們想檢定的落後期數(Lag)，且 $C_{0j}C_{00}^{-1}$ 矩陣為一期差 j 期自我相關係數矩陣(autocorrelation matrix at lag j)。則多維度 Portmanteau 統計量 $LB_k(s)$ 在 n 夠大時就會近似自由度為 $k^2 s$ 的卡方分配。在顯著水準 α 之下，若 $LB_k(s) > \chi^2_{1-\alpha,k^2 s}$ ，則拒絕虛無假設。我們用 "rBeta2009"套件的 rdirichlet()程式，生成 200 個亂數重複做 1000 次，計算其在顯著水準 $\alpha = 0.05$ 之下，拒絕虛無假設的比例。我們計算 $s = 1, \ldots, 6$ 時的拒絕比例，發現所有的拒絕比例都小於或近似 0.05，結果如(表 7)所示，這表示此亂數產生器所生成的亂數具有隨機性。

(表 6) H_0：x_t 與 $x_{t-1},\dots,x_{t-s}$ 之間獨立。生成貝他分配 $Beta(\alpha,\beta)$ 亂數 200 個做 Ljung-Box 檢定重複 1000 次，檢定與前 6 期 $(s=1,\dots,6)$之間是否獨立。表內為在顯著水準 0.05 之下拒絕虛無假設之比例。

Beta Parameters	s					
	1	2	3	4	5	6
$(0.1,0.1)$	0.060	0.047	0.045	0.045	0.046	0.058
(0.1,0.7)	0.043	0.036	0.028	0.042	0.041	0.039
$(0.9,0.9)$	0.042	0.048	0.042	0.053	0.046	0.058
(0.3,10)	0.037	0.035	0.040	0.049	0.043	0.046
(3,5)	0.052	0.048	0.044	0.042	0.049	0.065
$(1,10)$	0.037	0.041	0.038	0.044	0.036	0.050
$(10,10)$	0.059	0.038	0.049	0.044	0.040	0.041
$(100,100)$	0.057	0.066	0.049	0.055	0.052	0.045

(表 7) H_0：x_t 與 $x_{t-1},\dots,x_{t-s}$ 之間獨立。生成狄氏分配 $Dirichlet(\alpha_1\alpha_2;\alpha_3)$ 及 $Dirichlet(\alpha_1,\alpha_2,\alpha_3,\alpha_4;\alpha_5)$ 亂數 200 個做 Ljung-Box 檢定重複 1000 次，檢定與前 6 期 $(s=1,\dots,6)$之間是否獨立。表內為在顯著水準 0.05 之下拒絕虛無假設之比例。

Dirichlet Parameters	s					
	1	2	3	4	5	6
$(0.1,0.5,0.9)$	0.057	0.064	0.053	0.050	0.049	0.048
$(0.7,3,10)$	0.047	0.045	0.041	0.043	0.036	0.041
$(2,3,5)$	0.041	0.052	0.042	0.049	0.054	0.048
$(0.1,0.3,0.5,0.7,0.9)$	0.055	0.051	0.054	0.052	0.051	0.052
$(2,3,5,0.2,0.7)$	0.047	0.055	0.054	0.052	0.052	0.058
$(1,1,2,2,3)$	0.047	0.054	0.049	0.062	0.055	0.062

第四章　"rBeta2009"套件之其他應用

本章將介紹一些有關狄式分配的應用，並利用本論文所介紹"rBeta2009"套件生成不同亂數。這些應用包括反貝他分配、反狄式分配、Liouville 分配以及在凸面區域上的均勻分配。

第一節　生成反貝他分配 (Inverted Beta Distribution) 及反狄氏分配 (Inverted Dirichlet Distribution)

反貝他分配又被稱為貝他主要分配(beta prime distribution)或是第二類貝他分配(the second kind of beta distribution)。此分配在貝氏分析中可被當作伯努利分配(Bernoulli distribution)參數的勝算(odds)的共軛分配，且常被應用在股價報酬的實證估計。而 McDonald 和 Butler (1987) [27]使用兩個反貝他分配的混合分配來分析失業的持續時間。反狄氏分配則被 Sahai 和 Anderson (1973) [33]應用在計算隨機模型的變異數比率(variance ratio)的信賴區域。而 Bdiri 和 Nizar (2011) [5]將反狄氏分配的混合模型拿來做正的資料分群，在同一年他們也證明了反狄氏分配的無限混合模型(infinite mixture model)較高斯分配的無限混合模型來的有效[6]。若 $(X_1,\dots,X_k)$ 服從反狄氏分配 $IDirichlet(a_1,\dots,a_k;a_{k+1})$，則其機率密度函數為

$$f(x_1,\dots,x_k)=\frac{\Gamma\left(\sum_{i=1}^{k+1}a_i\right)}{\prod_{i=1}^{k+1}\Gamma(a_i)}\cdot\frac{\prod_{i=1}^{k}x_i^{a_i-1}}{\left(1+\sum_{i=1}^{k}x_i\right)^{\sum_{i=1}^{k+1}a_i}},\tag{13}$$

其中 $x_1,\dots,x_k>0, a_1,\dots,a_{k+1}>0, k>1$。若令 $k=1$ ，則 $(X_1,\dots,X_k)$ 服從所謂的反貝他分配 $IBeta(\alpha,\beta)$，其機率密度函數為

$$f(x)=\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\cdot\frac{x^{\alpha-1}}{(1+x)^{\alpha+\beta}},\tag{14}$$

其中 $x>0,\alpha,\beta>0$。

(定理 2) 令 $Y=(Y_1,\dots,Y_k)$，當

(a) $k>1$ 時，若 Y 服從狄氏分配 $Dirichlet(a_1,\dots,a_k;a_{k+1})$，則

$$X=(X_1,\dots,X_k)=\frac{Y}{1-Y_1-\cdots-Y_k}\tag{15}$$

(表 8) "rBeta2009"套件及"stats"的 rbeta()程式生成 3×10^6 個反貝他分配 $IBeta(\alpha, \beta)$ 亂數的 CPU 時間(秒)及增進比例(Proportion of Improvement)。

		rbeta()		Proportion of
α	β	rBeta2009	stats	Improvement
0.1	0.1	1.56	1.95	20.00%
	0.5	1.07	2.00	46.50%
	5	0.92	1.92	52.08%
	10	0.93	1.93	51.81%
	100	0.91	1.92	52.60%
0.5	0.5	1.17	1.90	38.42%
	5	1.14	2.04	44.12%
	10	1.28	2.06	37.86%
	100	1.31	2.08	37.02%
5	5	0.52	1.65	68.48%
	10	0.53	1.64	67.68%
	100	0.54	1.62	66.67%
10	10	0.50	1.65	69.70%
	100	0.52	1.59	67.30%
100	100	0.50	1.70	70.59%

服從反狄氏分配 $IDirichlet(a_1, \ldots, a_k; a_{k+1})$。

(b) $k = 1$ 時，若 Y 服從貝他分配 $Beta(\alpha, \beta)$，則

$$X = \frac{Y}{1 - Y} \tag{16}$$

服從反貝他分配 $IBeta(\alpha, \beta)$。

我們以"rBeta2009"套件及 R 內舊套件("stats"與"MCMCpack")的 rbeta()及 rdirichlet()程式生成狄氏分配及貝他分配亂數，再經由(定理 2)將其轉換成反狄氏分配及貝他分配亂數，分別記錄其 CPU 時間(單位：秒)及增進比例(Proportion of Improvement)結果列於(表 8)及(表 9)。(表 8)中紀錄的生成反貝他分配亂數的表現與(表 1)非常相似，在不同參數下皆能有良好的增進比例。其中也是參數較小時增進比例較低，範圍從 20%至 71%。而(表 9)紀錄了的產反狄氏分配亂數的 CPU 時間，增進比例範圍從 18%到 57%。

(表 9) 以套件"rBeta2009"及"MCMCpack"的 rdirichlet()生成3×10^6 個反狄氏分配 $IDirichlet(a_1, \ldots, a_4; a_5)$ 亂數的 CPU 時間(秒)及增進比例(Proportion of Improvement)。

		rdririchlet()		Proportion of
$\alpha_1 = \alpha_2 = \alpha_3$	$\alpha_4 = \alpha_5$	rBeta2009	MCMCpack	Improvement
0.1	0.1	5.92	7.29	18.79%
	0.5	4.46	7.55	40.93%
	5	3.74	6.56	42.99%
	10	3.72	6.44	42.24%
	100	3.79	6.40	40.78%
0.5	0.1	4.51	7.86	42.62%
	0.5	4.33	8.33	48.02%
	5	4.48	7.25	38.21%
	10	4.65	7.20	35.42%
	100	4.96	6.99	29.04%
5	0.1	3.31	6.31	47.54%
	0.5	3.76	6.75	44.30%
	5	2.46	5.48	55.11%
	10	2.49	5.63	55.77%
	100	2.37	5.54	57.22%
10	0.1	3.27	6.23	47.51%
	0.5	3.85	6.71	42.62%
	5	2.47	5.50	55.09%
	10	2.34	5.33	56.10%
	100	2.39	5.32	55.08%
100	0.1	3.23	6.02	46.35%
	0.5	4.09	6.46	36.69%
	5	2.53	5.41	53.23%
	10	2.37	5.24	54.77%
	100	2.37	5.18	54.25%

第二節　生成 Liouville 分配

Liouville 分配常被用於貝氏分析、多變量資料建模、皮爾森曲線系統(Pearson systems of curves) 、非參數推論(nonparametric inference)、分配無關之容忍區間(distribution-free tolerance intervals) 、隨機過程(stochastic processes)及其他領域。並且由 McNeil 和 Neslehova(2010)[26]將 Liouville 分配應用在阿基米得關聯結構(Archimedean coplas)，其中有 gamma-Liouville，inverse-gamma-Liouville 和 Clayton-Liouville 等關聯結構。若 $(X_1,\dots,X_n)$ 服從 Lioville 分配 $L_n(f(\cdot);\, a_1,\dots,a_n)$，則其聯合機率密度函數為

$$Kf\left(\sum_{i=1}^{n} x_i\right)\cdot\prod_{i=1}^{n} x_i^{a_i-1}, \tag{17}$$

其中 $x_i>0, a_i>0\,, i=1,\dots,n,\ K$ 為一常數，$f(\cdot)$ 為正連續函數，並且滿足

$$\int_{R_+} t^{a-1}f(t)dt<\infty\,, \tag{18}$$

$a=a_1+\cdots+a_n$ 。以下定理為兩種使用狄式分配產生 Liouville 分配的方法[25]：

(定理 3)

(a) 若 $(Y_1,\dots,Y_k)\sim Dirichlet(a_1,\dots,a_k;a_{k+1})$ ，令

$$(X_1,\dots,X_{k+1})=\left(Y_1,..,Y_k,1-\sum_{i=1}^{k}Y_i\right)\cdot Y_{k+1}; \tag{19}$$

(b) 若 $Y_i\sim Beta\left(\sum_{j=1}^{i}a_j\,,a_{i+1}\right), i=1,\dots,k$ ，令

$$(X_1,\dots,X_{k+1})=\left(\prod_{i=1}^{k}Y_i\,,(1-Y_1)\prod_{i=2}^{k}Y_i\,,\dots,1-Y_k\right)\cdot Y_{k+1}, \tag{20}$$

則 $(X_1,\dots,X_{k+1})\sim L_{k+1}(f(\cdot);a_1,\dots,a_{k+1})$ ，其中 $a_i>0, i=1,\dots,k+1$ ，$Y_{k+1}=\sum_{i=1}^{k+1}X_i\sim L_1\left(f(\cdot);\sum_{i=1}^{k+1}a_i\right)$ 且 $(Y_1,\dots,Y_k)$ 與 Y_{k+1} 互相獨立。

函數 $f(\cdot)$ 又分為兩類：第一類函數(Type I)的定義域為非封閉型(noncompact)，第二類函數(Type II)的定義域為封閉型(compact)。非封閉型意指其定義域為 $(0,\infty)$；而封閉型的定義域則為 $(0,t), t<\infty$，且若 $x\in(0,t)$，則

$x/t \in (0,1)$。以下將介紹兩個 $f(\cdot)$ 的例子：

(例一) (Type I: The Correlated Gamma Distribution)

令函數 $f(t) = t^{\alpha-1}e^{-\frac{t}{\beta}}$ ，若 $t, \alpha, \beta > 0$ ，則

$$f_{Y_{k+1}}(y) \propto y^{(\alpha+\sum a_i-1)-1}e^{-\frac{y}{\beta}}, \tag{21}$$

因此 $Y_{k+1} \sim Gamma(\alpha + \sum_{i=1}^{k+1} a_i - 1, \beta)$。

(例二) (Type II: The Dirichlet Distribution)

令函數 $f(t) = (1-t)^{b-1}$ ，若 $0 < t < 1, b > 0$ ，則

$$f_{Y_{k+1}}(y) \propto (1-y)^{b-1}y^{\left(\sum_{i=1}^{k+1} a_i\right)-1}, \tag{22}$$

因此 $Y_{k+1} \sim Beta\left(\sum_{i=1}^{k+1} a_i, b\right) = Dirichlet\left(\sum_{i=1}^{k+1} a_i; b\right)$ 。

根據(定理 3)、(例一)及(例二)，若要生成第一類函數的 Liouville 分配亂數，可利用(i)狄式分配跟 Gamma 分配或(ii)貝他分配跟 Gamma 分配。若要生成第二類函數的 Liouville 分配亂數，則可利用(i)狄式分配跟貝他分配或(ii)只用貝他分配。(表 10)及(表 11)為使用"rBeta2009"套件與其他舊 R 套件所花的 CPU 時間(單位：秒)及增進比例(Proportion of Improvement)，表內分別使用(定理 3)的(a)及(b)的方法來生成亂數。(表 10)是利用(例一)的第一類函數參數 $(\alpha, \beta) = (2, 5)$，分別用 rdirichlet()及 rbeta()來產生 Liouville 亂數，其中使用的 rgamma()是屬於"stats"套件。結果顯示，(定理 3)的(a)方法明顯比(b)快上許多，而"rBeta2009"套件的增進比例除了參數為 (0.1,0.1,0.1,0.1,0.1) 是由(b)方法較高，其餘皆是以(a)方法較高，從 32%到 44%，而(b)方法是從 19%到 37%。而(表 11)是利用(例二)的第二類函數配上參數 $b = 2$ ，分別用 rdirichlet()及 rbeta()來產生 Liouville 亂數。結果與(表 10)相似，(定理 3)的(a)方法比(b)快上許多，而增進比例除了參數為 (0.1,0.1,0.1,0.1,0.1) 是由(b)方法較高，其餘皆是以(a)方法較高，從 35%到 51%，而(b)方法是從 25%到 44%。總而言之，"rBeta2009"套件在產生 Liouville 分配亂數時，較其他 R 套件快了許多，增進比例也有一定的良好表現。

(表 10) 產生 3×10^6 個第一類函數的 Liouville 分配 $L_5(f(\cdot); a_1, \ldots, a_5)$ 亂數所需 CPU 時間(秒)，其中函數 $f(t) = t^{\alpha-1}e^{-\frac{t}{\beta}}, \alpha = 2, \beta = 5, t > 0$。

Generate method：(定理 3)		(a)			(b)		
$a_1 = a_2 = a_3$	$a_4 = a_5$	rBeta2009	MCMCpack	Proportion of Improvement	rBeta2009	stats	Proportion of Improvement
0.1	0.1	8.16	9.30	12.26%	11.98	14.40	16.81%
	0.5	6.64	9.82	32.38%	11.57	14.45	19.93%
	5	5.49	8.50	35.41%	10.47	13.90	24.68%
	10	5.62	8.36	32.78%	10.46	13.55	22.80%
	100	5.61	8.13	31.00%	10.56	13.69	22.86%
0.5	0.1	6.63	10.05	34.03%	9.89	14.07	29.71%
	0.5	6.50	10.63	38.85%	10.22	14.31	28.58%
	5	6.35	9.05	29.83%	8.97	13.41	33.11%
	10	6.48	9.08	28.63%	8.80	13.39	34.28%
	100	6.76	9.08	25.55%	8.89	13.43	33.80%
5	0.1	5.07	8.10	37.41%	8.80	13.12	32.93%
	0.5	5.57	8.52	34.62%	9.53	13.32	28.45%
	5	4.31	7.33	41.20%	8.09	12.51	35.33%
	10	4.16	7.39	43.71%	7.85	12.47	37.05%
	100	4.15	7.33	43.38%	7.94	12.32	35.55%
10	0.1	5.26	8.10	35.06%	8.93	13.14	32.04%
	0.5	5.82	8.55	31.93%	9.61	13.41	28.34%
	5	4.26	7.32	41.80%	8.14	12.43	34.51%
	10	4.22	7.02	39.89%	7.93	12.34	35.74%
	100	4.11	7.28	43.54%	7.94	12.40	35.97%
100	0.1	5.20	7.99	34.92%	8.83	13.26	33.41%
	0.5	5.94	8.33	28.69%	9.74	13.44	27.53%
	5	4.30	7.25	40.69%	7.94	12.57	36.83%
	10	4.27	7.24	41.02%	7.77	12.37	37.19%
	100	4.21	7.00	39.86%	7.86	12.46	36.92%

(表 11) 產生 3×10^6 個第二類函數的 Liouville 分配 $L_5(f(\cdot);a_1,\ldots,a_5)$ 亂數所需 CPU 時間(秒)， 其中函數 $f(t)=(1-t)^{b-1}, b=2, 0<t<1$。

Generate method：(定理 3)		(a)			(b)		
$a_1=a_2=a_3$	$a_4=a_5$	rBeta2009	MCMCpack	Proportion of Improvement	rBeta2009	stats	Proportion of Improvement
0.1	0.1	8.26	10.25	19.41%	11.71	15.24	23.16%
	0.5	6.23	10.41	40.15%	11.00	14.71	25.22%
	5	5.44	9.30	41.51%	10.31	14.53	29.04%
	10	5.55	9.33	40.51%	10.51	14.72	28.60%
	100	5.56	9.44	41.10%	10.42	14.60	28.63%
0.5	0.1	6.42	10.58	39.32%	9.28	14.60	36.44%
	0.5	5.90	10.97	46.22%	9.72	14.70	33.88%
	5	6.21	10.04	38.15%	8.62	14.01	38.47%
	10	6.40	9.96	35.74%	8.63	14.12	38.88%
	100	6.77	10.03	32.50%	8.61	14.10	38.94%
5	0.1	5.18	9.10	43.08%	8.61	13.95	38.28%
	0.5	5.42	9.54	43.19%	9.37	14.29	34.43%
	5	4.27	8.41	49.23%	7.72	13.34	42.13%
	10	4.12	8.36	50.72%	7.73	13.30	41.88%
	100	4.18	8.24	49.27%	7.64	13.20	42.12%
10	0.1	5.07	8.94	43.29%	8.53	14.11	39.55%
	0.5	5.54	9.57	42.11%	9.33	14.15	34.06%
	5	4.37	8.40	47.98%	7.77	13.31	41.62%
	10	4.13	8.02	48.50%	7.59	13.45	43.57%
	100	4.26	8.03	46.95%	7.49	13.29	43.64%
100	0.1	5.04	8.95	43.69%	8.78	14.02	37.38%
	0.5	5.86	9.35	37.33%	9.55	14.25	32.98%
	5	4.30	8.11	46.98%	7.82	13.22	40.85%
	10	4.21	7.99	47.31%	7.88	13.16	40.12%
	100	4.24	7.97	46.80%	7.72	13.50	42.81%

第三節　生成凸面區域的均勻分配

凸面區域的均勻分配可被應用在實驗設計(experimental design)中的均勻設計(uniform design)及約束最佳化(constrained optimization problem)。本節將介紹如何用"rBeta2009"套件的 rdirichlet()程式及 Fang *et al.* (1997)提出的方法(稱為 Simplex method)來產生在凸面區域上的均勻分配[9]，並與傳統的拒絕法(rejection method)做比較。以下先介紹如何使用狄式分配來產生在凸面區域上的均勻分配。

令 $\mathbf{v}_j = (v_j^1, \dots, v_j^n)'$ 為 n 維度中凸面區域的頂點座標，其中 $j = 1, \dots, k+1$。令 $\mathbf{B}_{n\times k} = (\mathbf{v}_1 - \mathbf{v}_{k+1}, \mathbf{v}_2 - \mathbf{v}_{k+1}, \dots, \mathbf{v}_k - \mathbf{v}_{k+1})$ ， $\mathbf{y}_{(k)} = (y_1, \dots, y_k)'$ ，而 S 為 $k+1$ 個頂點所包圍的凸面區域範圍：

$$S = \left\{ \mathbf{x} \in R_n : \mathbf{x} = \mathbf{B}\mathbf{y}_{(k)} + \mathbf{v}_{k+1}, y_i > 0, 0 < i \le k, \sum_{i=1}^{k} y_i \le 1 \right\}. \tag{23}$$

Fang *et al.* (1997)提出以下定理：

(定理 4) 若 $\mathbf{x} = (x_1, \dots, x_n)' \sim Uniform(S)$ ，且 $rank(\mathbf{B}) = min(n, k)$ 時，

$$\mathbf{x} = \mathbf{B}\mathbf{y}_{(k)} + \mathbf{v}_{k+1}, \tag{24}$$

其中 $\mathbf{y}_{(k)} \sim Dirichlet(a_1, \dots, a_k; a_{k+1})$ ，且其參數符合

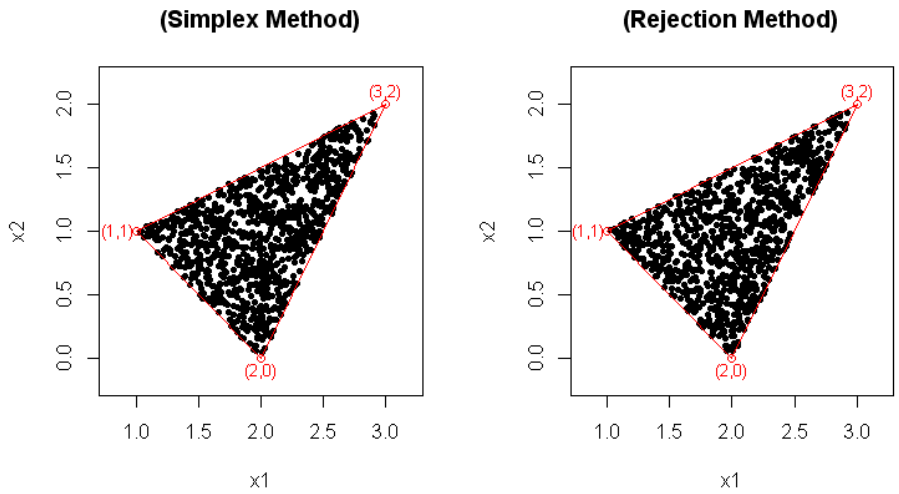
$$\begin{cases} a_1 = \cdots = a_{k+1} = 1, & for\ k \le n, \\ a_1 = \cdots = a_n = 1, a_{n+1} = \cdots = a_{k+1} = \dfrac{1}{k+1-n}, & for\ k > n. \end{cases} \tag{25}$$

我們利用(定理 4)將"rBeta2009"套件及"MCMCpack"套件中 rdirichlet()程式所生成的狄氏分配亂數轉換成為三角形區域內的亂數。以頂點為 $(0,0),(0,1),(1,0)$ 以及 $(2,0),(1,1),(3,2)$ 之兩組三角形區域為例，生成 3×10^6 個亂數，結果如(表 12)。這邊所使用的拒絕法(rejection method)以頂點 $(2,0),(1,1),(3,2)$ 之三角區域為例，我們生成均勻分配亂數在頂點為 $(1,0),(3,0),(3,2),(1,2)$ 矩形中，若亂數在頂點為 $(2,0),(1,1),(3,2)$ 的三角形區域內則留下，其餘則捨棄(接受率約為 37.5%)，而以頂點為 $(0,0),(0,1),(1,0)$ 之三角形區域的接受率約為 50%。可以看到"rBeta2009"套件所花的 CPU 時間(秒)較"MCMCpack"套件快上許多，兩例

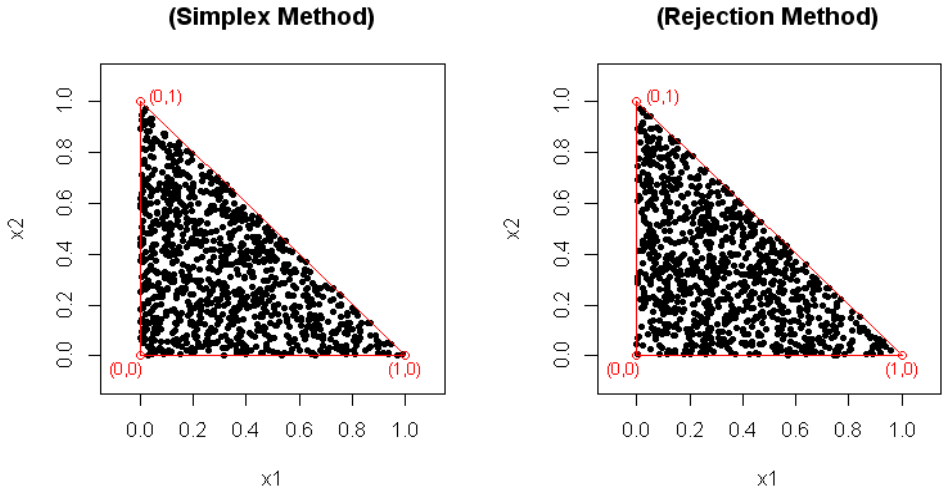
(表 12) 使用"rBeta2009"及"MCMCpack"來生成 3×10^6 個三角形區域內均勻分配亂數的 CPU 時間(秒)。()內為增進比例。

(v_j^1, v_j^2)	rdirichlet()		runif()
	rBeta2009	MCMCpack	stats
$(2,0)$ $(1,1)$ $(3,2)$	2.06	5.84(64.73%)	11.00(81.27%)
$(0,0)$ $(0,1)$ $(1,0)$	2.30	6.76(65.98%)	7.46(69.17%)

的增進比例皆在 65%左右，並且可以看到使用 simplex 方法及"rBeta2009"套件較使用拒絕法及"stats"套件快上許多。 (圖 1)為生成頂點 $(2,0),(1,1),(3,2)$ 之三角區域內的亂數 1000 個，左圖是使用(定理 4)的方法(simplex method)，用 rdirichlet()程式生成亂數所轉換而成；右圖是使用拒絕法，其中均勻分配之亂數由 runif()程式生成。 (圖 2)是將(圖 1)中三角區域範圍改成 $(0,0),(0,1),(1,0)$，我們可以看到，(圖 1)及(圖 2)中左右兩邊的圓點分布幾乎一樣均勻(散亂)，這表示兩種方法的準確性大致相同。但是 simplex 方法能讓生成凸面區域上的均勻分配更有效率(不需捨棄生成之亂數)，而"rBeta2009"套件更是增加了亂數生成的速度，其增進比例約為 65%。



(圖 1) 產生 1000 個以頂點 $(v_1^1, v_1^2) = (2,0), (v_2^1, v_2^2) = (1,1), (v_3^1, v_3^2) = (3,2)$ 圍成三角形區域的均勻分配。左圖使用套件"rBeta2009"中的 rdirichlet()程式及 simplex 方法，右圖使用"stats"套件中的 runif()程式及拒絕法(rejection method)。



(圖 2) 產生 1000 個以頂點 $(v_1^1, v_1^2) = (0,0), (v_2^1, v_2^2) = (1,0), (v_3^1, v_3^2) = (0,1)$ 圍成三角形區域的均勻分配。左圖使用套件"rBeta2009"中的 rdirichlet()程式及 simplex 方法，右圖使用"stats"套件中的 runif()程式及拒絕法(rejection method)。

第五章　結論

本篇論文介紹如何使用新的 R 套件"rBeta2009"來生成貝他分配及狄氏分配亂數並評估此套件之有效性、準確性及隨機性。新套件"rBeta2009"無論在 32 位元或 64 位元處理器的環境下其 CPU 時間比現有的 R 套件"stats"及"MCMCpack"有顯著的減少，其增進比例約為 10%到 80%。利用 Kolmogorov-Smirnov 檢定與 Ljung-Box 檢定，本文也驗證了此套件的準確性及隨機性。以實際應用的角度來說，套件"rBeta2009"可以完全取代其他的 R 套件來執行貝他分配及狄式分配的亂數產生。

此外，我們也介紹了四個可以利用"rBeta2009"套件來產生的分配，分別是反貝他分配、反狄式分配、Liouville 分配及在凸面區域上的均勻分配。這對於需要大量運算及電腦模擬相關分配的研究工作者(或企業界)有很大的幫助。

參考文獻

[1] A.C. Atkinson and Whittaker (1976). A Switching Algorithm for the Generation of Beta Random Variables With at Least One Parameter Less Than One. *Proceedings of the Royal Society of* London, Series A, 139, pp. 462-467.

[2] K. Alam, R. Abernathy and C.L.Williams (1993). Multivariate Goodness-of-Fit Tests Based on Statistically Equivalent Blocks. *Communication in Statistics*, Theory Methods 22, pp. 1515–1533.

[3] A.G. Ashraf and S. Tamás (2009). On Numerical Calculation of Probabilities According to Dirichlet Distribution. *Annals of Operations Research*, 177, pp. 185–200.

[4] T.W. Anderson (1966). Some Nonparametric Procedures Based on Statistically Equivalent Blocks. P*roceedings of International Symposium on Multivariate Analysis*. P.R. Krishnaiah ed., Academic Press Inc., New York, pp. 5–27.

[5] T. Bdiri and N. Bouguila (2011). Learning Inverted Dirichlet Mixtures for Positive Data Clustering. *Neural Information Processing*, ICONIP 2011, Part II, LNCS 7063, pp. 71–78. Springer, Heidelberg.

[6] T. Bdiri and N. Bouguila (2011). Learning Inverted Dirichlet Mixtures for Positive Data Clustering. *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, RSFDGrC 2011. LNCS, 6743, pp. 265–272. Springer, Heidelberg.

[7] R.C.H. Cheng (1978). Generating Beta Variates with Nonintegral Shape Parameters. *Communications of the Association for Computing Machinery*, 21, pp. 317-322.

[8] C.W. Cheng, Y.C. Hung and N. Balakrishnan (2012). Package "rBeta2009". URL http://cran.r-project.org/package=rBeta2009.

[9] R.V. Foutz (1980). A test for goodness of fit based on empirical probability

measure. *The Annals of Statistics*, 8, pp. 989–1001.

[10] K.T. Fang, G.L. Tian and M.Y. Xie. (1997). Uniform Distribution on Convex Polyhedron and Its Applications. Department of Mathematics, Hong Kong Batist University, No. 149.

[11] D.A.S. Fraser (1957). *Nonparametric Methods in Statistics*, JohnWiley & Sons, NewYork.

[12] T. Hahn (2005). CUBA—a library for multidimensional numerical integration. *Computer Physics Communications*, 168, pp. 78–95.

[13] Y.C. Hung, N. Balakrishnan and Y.T. Lin (2009). Evaluation of Beta Generation Algorithms. *Communications in Statistics - Simulation and Computation*, 38, pp. 750-770.

[14] Y.C. Hung, N. Balakrishnan and C.W. Cheng (2011). Evaluation of Algorithms for Generating Dirichlet Random Vectors. *Journal of Statistical Computation and Simulation*, 81, pp. 445-459.

[15] J.R.M. Hosking (1981). Fractional Differencing. *Biometrica*, Vol. 68, No. 1981, pp. 165-176.

[16] J.R.M. Hosking (1980). The Multivariate Portmanteau Statistic. *Journal of American Statistical Association*, 75, pp. 602-608.

[17] B. Jarle and O.E. Terje (1991). An adaptive algorithm for the approximate calculation of multiple integrals. *ACM Transactions on Mathematical Software*, Vol. 17, No. 4, pp. 437-451.

[18] B. Jarle and O.E. Terje (1991). Algorithm 698: DCUHRE: An adaptive algorithm for the approximate calculation of multiple integrals. *ACM Transactions on Mathematical Software*, Vol. 17, No. 4, pp. 452-456.

[19] M.D. Jöhnk (1964). Erzeugung von betaverteilten und gammaverteilten zuffallszahlen. *Metrika*, 8, pp. 5-15.

[20] D.P. Kennedy (1988). A note on stochastic search methods for global optimization. *Advances in Applied Probability*, 20, pp. 476-478.

[21] K. Lange (2005). Applications of the Dirichlet distribution to forensic match probabilities, *Genetica* , 96, pp. 107–117.

[22] G. Laval, M. SanCristobal and C. Chevalet (2003). Maximum-likelihood and Markov chain Monte Carlo approaches to estimate inbreeding and effective size from allele frequency changes. *Genetics*, 164 (3), pp. 1189-1204.

[23] J. Liouville (1839). Note sur quelquess integrals définies. *Journal de Mathématiques Pures et Appliquées*, 4, pp. 225-235.

[24] G.M. Ljung and G.E.P. Box (1978). On a Measure of Lack of Fit in Time Series Models. *Biometrika*, 65, pp. 297-303.

[25] R.E. Madsen, D. Kauchak and C. Elkan (2005). Modeling Word Burstiness Using the Dirichlet Distribution. *Proceeding of the 22nd International Conference on Machine Learning*, pp. 545-552.

[26] A.J. McNeila and J. Nešlehová (2010).From Archimedean to Liouville copulas. *Journal of Multivariate Analysis*, 101, 8, pp. 1772-1790.

[27] J.B. McDonald and R.J. Butler (1987). Some generalized mixture distributions with an application to unemployment duration. *The Review of Economics and Statistics*, 69, pp. 232–240.

[28] D.G. Rameshwar and St. P.R. Donald (1987). Multivariate Liouville distribution. *Journal of Multivariate Analysis*, 23, pp. 233-256.

[29] R.J. Serfling (1980). *Approximation Theorems for Mathematical Statistics*. JohnWiley & Sons, NewYork.

[30] B.W. Schmeiser and A.J.G. Babu (1980). Beta Variate Generation via Exponential Majorizing Functions. *Operations Research*, 28,pp. 917-926.

[31] K. Sjölander, K. Karplus, M. Brown, R. Hughey, A. Krogh, I. S. Mian and D.

Haussler (1996). Dirichlet mixtures: A method for Improving Detection of Weak but Significant Protein Sequence Homology. *The Computer Application in Bioscience*, 12, pp. 327-345.

[32] H. Sakasegawa (1983). Stratified rejection and squeeze method for generating beta random numbers. *Annals of the Institute Statistical Mathematics*, 35,pp. 291-302.

[33] H. Sahai and R.L. Anderson (1973). Confidence regions for variance ratios of the random models for balanced data. *Journal of the American Statistical Association*, 68, 344, pp. 951-952.

[34] J.W. Tukey (1947). Non-parametric estimation II. Statistically Equivalent Blocks and tolerance regions – the continuous case. *The Annals of Mathematical Statistics*, 18, pp. 529–539.

[35] G.G. Tiao and I. Guttman (1965a). The inverted Dirichlet distribution with applications. *Journal of the American Statistical Association*, 60, pp. 793–805.

[36] G.G. Tiao and I. Guttman (1965b). The inverted Dirichlet distribution with applications. *Journal of the American Statistical Association*, 60, pp. 1251-1252.

[37] G.R. Warnes (2010). Various R programming tools. URL http://cran.r-project.org/web/packages/gtools/gtools.pdf.

[38] E.P. Xing, M.I. Jordan, R.M. Karp and S. Russell (2002). A Hierarchical Bayesian Markovian Model for Motifs in Biopolymer Sequences. *Proceedings of Advances in National Information Processing Systems*, pp. 1489-1496.

[39] H. Zechner and E. Stadlober (1993). Generating beta variates via patchwork rejection. *Computing*, 50,pp. 1-18.