Computational Statistics and Data Analysis 71 (2014) 1011–1020

Contents lists available at ScienceDirect

Computational Statistics and Data Analysis

journal homepage: www.elsevier.com/locate/csda

Generating beta random numbers and Dirichlet random vectors in R: The package rBeta2009

Ching-Wei Cheng[a], Ying-Chao Hung[b,*], Narayanaswamy Balakrishnan[c]

[a] *Department of Statistics, Purdue University, 250 North University Street, West Lafayette, IN 47907-2066, USA*
[b] *Department of Statistics, National Chengchi University, Wenshan District, Taipei 116, Taiwan*
[c] *Department of Mathematics and Statistics, McMaster University, Hamilton, Ontario, Canada L8S 4K1*

ARTICLE INFO

Article history:
Received 30 August 2012
Received in revised form 29 January 2013
Accepted 14 February 2013
Available online 26 February 2013

Keywords:
R
Beta variates
Dirichlet random vectors
Computer time
Kolmogorov–Smirnov test
Ljung–Box test

ABSTRACT

A software package, **rBeta2009**, developed to generate beta random numbers and Dirichlet random vectors in R is presented. The package incorporates state-of-the-art algorithms so as to minimize the computer generation time. In addition, it is designed in a way that (i) the generation efficiency is robust to changes of computer architecture; (ii) memory allocation is flexible; and (iii) the exported objects can be easily integrated with other software. The usage of this package is then illustrated and evaluated in terms of various performance metrics.

1. Introduction

The beta variates and Dirichlet random vectors are extensively used in the areas of Bayesian statistics, stochastic modeling and simulation, program evaluation and review techniques (PERT), critical path method (CPM), and project management and control systems (Lange, 1995; Gupta and Nadarajah, 2004; Tian et al., 2010; da-Silva et al., 2011). Over the years, many algorithms have been introduced in the literature for the computer generation of beta variates and Dirichlet random vectors. For example, the beta generating function rbeta() in the R default package **stats** R Development Core Team (2008) is based on the algorithms by Cheng (1978); the beta generating function rand() in SAS utilizes the algorithms by Atkinson and Whittaker (1976), Cheng (1978) and Atkinson (1979); the beta generating function betarnd() in MATLAB Statistics Toolbox uses both the order statistics method and Jöhnk's method (Jöhnk, 1964; Rubinstein and Kroese, 1981); and the Dirichlet generating functions rdirichlet() and rDirichlet() in the R packages **MCMCpack** and **compositions** utilize the method based on the transformation of gamma variates (Hogg and Craig, 1978; Aitchison, 1986). The shortcoming for the existing software packages is that most algorithms used to develop the beta and Dirichlet generating functions are dated (the readers can refer to Hung et al. (2009, 2011) for a review of recent developments on beta and Dirichlet generating functions). Besides, their efficiency (i.e., computer generation time) is usually not robust to changes in hardware/software platform. From the viewpoint of implementation, it is better to use state-of-the-art algorithms so that maximum efficiency can be achieved on different computer platforms (e.g., 32 or 64-bit processors, Windows or Mac OS X operating systems, etc.).

In this study, we present a new R package **rBeta2009**, which contains functions rbeta() and rdirichlet() for generating beta variates and Dirichlet random vectors, respectively. The package mainly utilizes the recent guidelines provided by

* Corresponding author. Tel.: +886 2 29387115; fax: +886 2 29398024.
E-mail addresses: cheng138@purdue.edu (C.-W. Cheng), hungy@nccu.edu.tw (Y.-C. Hung), bala@mcmaster.ca (N. Balakrishnan).

0167-9473/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.csda.2013.02.019

Hung et al. (2009, 2011) for choosing the most efficient generating algorithms in accordance with the values of the shape parameters. In addition to providing high-speed generation, the package is also designed in a way that (i) it has good statistical properties; (ii) it is robust to changes in computer platform; and (iii) it is easily integrated with other software. These are criteria commonly used for evaluating a random number generator (Wichmann and Hill, 2006). The rest of this work is organized as follows. Section 2 introduces the guidelines based on which the generating functions are developed. Section 3 illustrates the usage of package **rBeta2009** and explains how the package is designed so as to achieve its portability and adaptability. Section 4 evaluates the proposed package in terms of efficiency, robustness, accuracy, and randomness. For efficiency and robustness, the proposed package is compared with the existing R package(s) in terms of computer generation time on various hardware and software platforms. For accuracy, both the univariate and multivariate goodness-of-fit tests are performed by means of the Kolmogorov–Smirnov statistic based on the generated random numbers/vectors. For testing randomness, the Ljung–Box (or Portmanteau) statistic is applied. Section 5 summarizes the results. The developed R package is available from Comprehensive R Archive Network (CRAN) at http://CRAN.R-Project.org/package=rBeta2009.

2. Guidelines for the generating functions

A standard beta variate $Beta(\alpha, \beta)$ has the probability density function

$$f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \quad 0 \le x \le 1, \tag{1}$$

with $\alpha, \beta > 0$ being shape parameters. As a multidimensional generalization of beta variate, a k-variate random vector $(Y_1, \dots, Y_k)$ is said to have a $Dirichlet(\alpha_1, \dots, \alpha_{k+1})$ distribution if the joint density function is given by

$$f(y_1, \dots, y_k) = \frac{\Gamma(\alpha_1 + \cdots + \alpha_{k+1})}{\Gamma(\alpha_1) \cdots \Gamma(\alpha_{k+1})} y_1^{\alpha_1 - 1} y_2^{\alpha_2 - 1} \cdots y_k^{\alpha_k - 1} \left(1 - \sum_{i=1}^{k} y_i\right)^{\alpha_{k+1}-1}, \tag{2}$$

where $y_i \ge 0$ for all $i = 1, \dots, k$, $\sum_{i=1}^{k} y_i \le 1$, and $\alpha_1, \dots, \alpha_{k+1} > 0$ are shape parameters. In this section, we introduce the guidelines utilized by package **rBeta2009** for generating beta random numbers and Dirichlet random vectors.

2.1. Guideline for generating beta random numbers

Hung et al. (2009) proposed a useful guideline for choosing a state-of-the-art algorithm for generating the $Beta(\alpha, \beta)$ random number and evaluated the guideline through various performance metrics. Our package mainly utilizes this guideline but removes one algorithm (called Kennedy's MK) that was used when the shape parameters α and β are both less than one and $\alpha + \beta > 1.2$. This was done since the Kennedy's MK algorithm is in fact a stochastic search method (i.e., an approximate method) that generates "asymptotically" the desired beta distribution, which may increase uncertainty in the generated random numbers. Based on the empirical study carried out by Hung et al. (2009), in our package, the Kennedy's MK algorithm is replaced by another algorithm called B00 (Sakasegawa, 1983). In summary, the beta generating function in **rBeta2009** utilizes the following guideline for choosing the most efficient algorithm:

- for $\alpha, \beta < 1$, choose the B00 algorithm (Sakasegawa, 1983);
- for $\alpha < 1 < \beta$ or $\alpha > 1 > \beta$, choose the B01 algorithm (Sakasegawa, 1983);
- for $\alpha, \beta > 1$, choose the B4PE algorithm (Schmeiser and Babu, 1980) if one parameter is less than 1.2 and the other is larger than 4; otherwise, choose the BPRS algorithm (Zechner and Stadlober, 1993).

2.2. Guideline for generating Dirichlet random vectors

It is known that the Dirichlet random vector can be generated by considering the transformation based on beta variates. To see how this works, suppose $X_1, \dots, X_k$ are independent beta random variables with $X_i \sim Beta(\alpha_i, \alpha_{i+1} + \cdots + \alpha_{k+1})$, $i = 1, \dots, k$, then by letting $Y_1 = X_1, Y_i = X_i \prod_{j=1}^{i-1}(1 - X_j)$ for $i = 2, \dots, k$, the random vector $(Y_1, \dots, Y_k)$ has a $Dirichlet(\alpha_1, \dots, \alpha_{k+1})$ distribution (Narayanan, 1990). In order to accelerate the generation speed based on this approach, Hung et al. (2011) introduced three useful guidelines. These three guidelines are stated below, where *Guideline* 1 for choosing the beta algorithm has been modified according to the description above in Section 2.1.

Guideline 1: choosing the fastest beta generation algorithm. Use the guideline stated in Section 2.1 for selecting the best algorithm for generating the beta variates.

Guideline 2: re-ordering the shape parameters. Denote the ordered values of $\alpha_1, \dots, \alpha_{k+1}$ by $\alpha_{(1)}, \dots, \alpha_{(k+1)}$, and let m ($\le k + 1$) be the number of shape parameters less than one.

- If $\alpha_{(k+1)} < 1$ and $(k + 1)$ is odd, generate the random vector from

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

- if $\alpha_{(k+1)} < 1$ and $(k+1)$ is even, generate the random vector from

  $$Dirichlet\,(\alpha_{(1)}, \alpha_{(2)}, \alpha_{(4)}, \ldots, \alpha_{(k-1)}, \alpha_{(k+1)}, \alpha_{(k-2)}, \alpha_{(k-4)}, \ldots, \alpha_{(5)}, \alpha_{(3)});$$
- if $1 \le \alpha_{(k+1)} \le 3$ and $\alpha_{(1)} \ge 1$, generate the random vector from

  $$Dirichlet\,(\alpha_{(k+1)}, \alpha_{(k-1)}, \alpha_{(k-2)}, \ldots, \alpha_{(1)}, \alpha_{(k)});$$
- if $\alpha_{(k+1)} > 3$ and $\alpha_{(1)} \ge 1$, generate the random vector from

  $$Dirichlet\,(\alpha_{(k+1)}, \alpha_{(k)}, \alpha_{(k-2)}, \alpha_{(k-3)}, \ldots, \alpha_{(1)}, \alpha_{(k-1)});$$
- if $\alpha_{(k+1)} \ge 1, \alpha_{(1)} < 1, m > \frac{k+2}{2}$, and $\alpha_{(m)} \le 0.5$, generate the random vector from

  $$Dirichlet\,(\alpha_{(1)}, \alpha_{(2)}, \ldots, \alpha_{(m-1)}, \alpha_{(m+1)}, \alpha_{(m+2)}, \ldots, \alpha_{(k+1)}, \alpha_{(m)});$$
- otherwise, generate the random vector from

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

Guideline 3: reducing the amount of arithmetic operations. Replace the formula $Y_i = X_i \prod_{j=1}^{i-1}(1 - X_j)$ in the beta transformation method by $Y_i = X_i \left(1 - \sum_{j=1}^{i-1} Y_j\right)$.

The package **rBeta2009** basically utilizes the above three guidelines to generate the Dirichlet random vectors. For implementation purpose, the beta transformation method along with these guidelines is summarized in the following algorithm:

Step 1: Re-order the shape parameters $\alpha_1, \ldots, \alpha_{k+1}$ based on *Guideline*2 and denote the new ordering by $\alpha_{s_1}, \ldots, \alpha_{s_{k+1}}$.

Step 2: For $i = 1, \ldots, k$, generate independent beta random variables $X_i \sim Beta\left(\alpha_{s_i}, \sum_{j=i+1}^{k+1} \alpha_{s_j}\right)$ by using *Guideline*1.

Step 3: Set $Y_1 = X_1$ and $Y_i = X_i\left(1 - \sum_{j=1}^{i-1} Y_j\right)$ for $i = 2, \ldots, k$ (*Guideline*3).

Step 4: Re-arrange the order of $Y_1, \ldots, Y_k$ suitably and return the desired vector.

3. Software design and implementation

In this section, we describe the design feature of package **rBeta2009** from the perspective of memory allocation, portability, and adaptability. We also illustrate the usage of this package. All numerical results presented in this section were executed on 3.0 GHz AMD® Athlon™ II X4 640 processors with 8 GB of cache under the operating system of Microsoft® Windows 7 64-bit Service Pack 1 (SP1).

3.1. Memory allocation and portability

Since the source code of package **rBeta2009** was primarily written in C, to pass the R objects to C, we can simply use the application program interfaces (APIs). There are two ways to call C functions using the R API: .C() and .Call(). The .C() interface returns the arguments of the function (i.e., void or primitive values/pointers such as double/double *), and so it can be simply wrapped with other software (e.g., Mathematica wraps value/pointer-type functions via the MathLink API; Python wraps value/pointer-type functions via the ctype module; MATLAB wraps void-type subroutines through creating C source MEX-files, etc.). The main shortcoming of using the .C() interface is that arguments such as as.double() or as.integer() need to be claimed so as to ensure that objects are passed with a correct type. This means that memory pre-allocation for the desired objects is necessary. To avoid the memory allocation problem induced by objects of large sizes, we utilize the .Call() interface in package **rBeta2009**.

The .Call() interface allows us to pass the R objects into the C environment and return the R data structure (such as matrix and list) without matching up the arguments. In addition, it uses functions such as allocMatrix to efficiently allocate/manage memory. It is noted that when .Call() is used, R objects are represented as a data structure in C, called an *S-expression* or SEXP (Symbolic EXPression). To convert the returned SEXP data structure into standard data types that can be incorporated with other software, we can simply remove the headers Rdefines.h, R.h, Rmath.h, the calls PROTECT(), UNPROTECT(), and replace SEXP by appropriate typedef (such as int or double). In addition, the following associated changes need to be made:

- Remove the calls GetRNGstate() and PutRNGstate(), substitute, for example, the SIMD-oriented Fast Mersenne Twister pseudo-random number generator (Saito and Matsumoto, 2008) for the uniform random number generator unif_rand().
- Include the header algorithm and substitute functions fmin() and fmax() for functions min() and max(), respectively.
- Substitute pow() for the power function R_pow().

Table 1
The computer generation time (in seconds) of 3×10^6 $Beta(\alpha, \beta)$ random numbers based on different choices of power functions under the 32 and 64-bit processors. Here, pow, R_pow, and exp–log represent the functions pow(a, b), R_pow(a, b), and exp($b * \log(a)$) for computing a^b in package **rBeta2009**, respectively.

$Beta(\alpha, \beta)$		32-bit processor			64-bit processor		
α	β	pow	R_pow	exp–log	pow	R_pow	exp–log
0.05	0.05	1.028	0.879	1.388	4.626	4.719	0.862
	0.5	0.660	0.521	0.919	3.205	2.846	0.542
	1.3	0.473	0.419	0.679	2.410	2.450	0.392
	7.3	0.516	0.462	0.731	2.544	2.617	0.435
0.5	0.5	0.675	0.345	0.934	3.159	0.345	0.552
	1.3	0.528	0.332	0.777	2.621	0.674	0.440
	7.3	0.684	0.522	0.976	3.445	1.525	0.572
1.3	1.3	0.450	0.410	0.505	1.334	1.355	0.330
	7.3	0.486	0.453	0.536	1.165	1.186	0.330
7.3	7.3	0.324	0.306	0.350	0.682	0.698	0.243

As for rendering the functions from returning double array pointer into returning void, we should make the claim for the object to be returned as an input argument, say, replace the command "return out" by "return", or simply remove the command "return out" in SEXP.

Remark. The readers can refer to the documents of Eddelbuettel and François (2011) and R Development Core Team (2012) for integrating other software packages with R.

3.2. Adaptability

It is worth noting that the algorithms used in package **rBeta2009** mainly involve the calculation of power functions, say, a^b. In the primary source code C, this computation can be done by using either pow(a, b), R_pow(a, b) (by including the header Rmath.h), or exp($b * \log(a)$). However, computational efficiency based on these three functions seems to be quite different under 32 and 64-bit architectures. To illustrate this, computer times for generating various beta random numbers based on these three different functions are shown in Table 1.

As can be seen from Table 1, R_pow(a, b) performs best (i.e., has the smallest execution time) under the 32-bit architecture, while exp($b * \log(a)$) performs best under the 64-bit architecture for generating the beta random numbers under consideration. Based on these numerical results, package **rBeta2009** can be designed in a way that the most efficient power function is chosen in accordance with a 32 or 64-bit platform. This is done simply by adding the following subroutine in the package:

```
double power(double a, double b) {
    #if __x86_64__ || __ppc64__
    return(exp(b * log(a)));
    #else
    return(R_pow(a, b));
    #endif
}.
```

3.3. Implementation of the beta generating function

The function rbeta(n, shape1, shape2) in package **rBeta2009** realizes the beta generating algorithm introduced in Section 2.1. It simulates n $Beta$(shape1, shape2) variates. Examples for illustrating the usage of the rbeta() function are shown below. First, the package needs to be loaded into R and a random number seed is arbitrarily set:

```
R > library("rBeta2009")
Attaching package: 'rBeta2009'.
The following object(s) are masked from 'package:stats':
    rbeta
R > set.seed(41352).
```

Now, we generate ten $Beta(4.3, 1.1)$ random variates as follows:

```
R > rbeta(10, 4.3, 1.1)
[1] 0.8551216 0.7952882 0.8843819 0.8751342 0.5245602 0.9142886 0.8428009
```

Table 2
The computer platforms considered for evaluating the efficiency of package **rBeta2009**.

Operating system	CPU	Cache
Windows 7 SP1	Intel® i7-3520M @ 2.90 GHz	8 GB DDR3-1600
Ubuntu Linux 12.10	Intel® i7-3520M @ 2.90 GHz	8 GB DDR3-1600
Mac OS X 10.8.2	Intel® Core™ 2 Duo @ 2.4 GHz	4 GB DDR3-1067

[8] 0.9901657 0.9044049 0.5308266.

Now, we generate ten $Beta(0.6, 7.2)$ random variates as follows:

$R > rbeta(10, 0.6, 7.2)$

[1] 5.476593e−02 2.516238e−01 4.813845e−02 2.643006e−02 7.112204e−02

[6] 3.909095e−01 1.727862e−05 2.018599e−01 6.598838e−05 2.456395e−02.

3.4. Implementation of the Dirichlet generating function

The function rdirichlet(n, shape) in package **rBeta2009** generates nk-variate Dirichlet random vectors with the shape parameters collected in the $(k+1)$-dimensional vector "shape". Instead of returning a k-component vector $(y_1, \dots, y_k)$, rdirichlet() returns a vector $(y_1, \dots, y_k, 1 - \sum_{i=1}^{k} y_i)$ with $(k+1)$ components so as to make the dimensionality of the output consistent with other existing packages such as **MCMCpack** and **gtools**. The usage of the function rdirichlet() is illustrated in the following example, which simulates seven 3-variate random vectors from the $Dirichlet(1.5, 0.7, 5.2, 3.4)$ distribution:

$R > set.seed(41352)$

$R > rdirichlet(7, c(1.5, 0.7, 5.2, 3.4))$.

```
        [, 1]         [, 2]         [, 3]       [, 4]
[1, ]   0.126182368   0.016922990   0.7430068   0.1138878
[2, ]   0.340030480   0.003867118   0.5466362   0.1094662
[3, ]   0.103226485   0.118618688   0.6603255   0.1178294
[4, ]   0.117983104   0.004902538   0.3256576   0.5514568
[5, ]   0.004467648   0.019712145   0.3093933   0.6664269
[6, ]   0.066052490   0.102936431   0.6683340   0.1626771
[7, ]   0.152099005   0.080641869   0.3782559   0.3890032
```

When the input parameters are not valid (e.g., the shape parameters are negative), an error message is issued:

$R > rdirichlet(7, c(-1.5, 0.7, 5.2, 3.4))$.

```
Error in rdirichlet(7, c(−1.5, 0.7, 5.2, 3.4)):
   Shape parameters should be all positive.
```

4. Performance evaluation

In this section, an empirical study is carried out to evaluate package **rBeta2009** in terms of efficiency, robustness, accuracy, and randomness.

4.1. Efficiency and robustness

We first demonstrate the efficiency and robustness of package **rBeta2009** by comparing it with the existing R packages in terms of computer generation time. Table 2 provides a list of computer platforms used to carry out these numerical comparisons, which mainly includes three types of operating systems (Windows 7, Ubuntu Linux, and Mac OS X). It should be mentioned here that both the 32 and 64-bit R binaries can be executed on Windows 7 and Mac OS X 10.8.2, while only the 64-bit binary can be executed on Ubuntu Linux 12.10.

Table 3 presents the computer times of the beta generating function in both package **rBeta2009** (in boldface) and the R default package **stats** executed on various computer platforms. Note that each computer time is recorded based on 3×10^6 generated beta variates with a wide range of selected shape parameters. As can be seen from Table 3, the package **rBeta2009** significantly outperforms the default package **stats** in generating all beta variates considered in this study throughout all platforms. In some cases such as $Beta(0.1, 1.1)$, the improvement of computer generation time is, surprisingly, up to 71%–81%.

Tables 4 and 5 present the computer times of the Dirichlet generating functions in package **rBeta2009** (in boldface) and two competing packages **MCMCpack** and **compositions** executed on various computer platforms. Here again, each

Table 3
The computer times (in seconds) of packages **rBeta2009** (in boldface) and **stats** on various computer platforms. Each computer time is recorded based on 3×10^6 generated $Beta(\alpha, \beta)$ variates.

α	β	Windows 7 SP1		Mac OS X 10.8.2		Ubuntu 12.10	
(32-bit binary)							
0.1	0.1	**0.360**	0.780	**0.921**	0.971		
	0.8	**0.225**	0.870	**0.472**	1.099		
	1.1	**0.169**	0.893	**0.328**	1.137		
	2.5	**0.211**	0.847	**0.384**	1.133		
	100	**0.208**	0.814	**0.397**	1.086		
0.8	0.8	**0.402**	0.664	**0.675**	0.741		
	1.1	**0.368**	0.665	**0.623**	0.790		
	2.5	**0.447**	0.721	**0.739**	0.840		
	100	**0.553**	0.800	**0.901**	0.916		
1.1	1.1	**0.304**	0.534	**0.449**	0.611		
	2.5	**0.241**	0.617	**0.357**	0.704		
	100	**0.267**	0.734	**0.586**	0.820		
2.5	2.5	**0.280**	0.579	**0.444**	0.656		
	100	**0.258**	0.629	**0.417**	0.695		
100	100	**0.200**	0.634	**0.330**	0.684		
(64-bit binary)							
0.1	0.1	**0.645**	0.704	**0.760**	0.863	**0.509**	0.658
	0.8	**0.386**	0.712	**0.489**	0.900	**0.288**	0.650
	1.1	**0.315**	0.700	**0.315**	0.858	**0.224**	0.628
	2.5	**0.338**	0.690	**0.377**	0.900	**0.261**	0.619
	100	**0.394**	0.691	**0.364**	0.832	**0.260**	0.589
0.8	0.8	**0.397**	0.604	**0.481**	0.589	**0.275**	0.539
	1.1	**0.373**	0.611	**0.350**	0.616	**0.254**	0.556
	2.5	**0.435**	0.657	**0.420**	0.673	**0.300**	0.610
	100	**0.550**	0.703	**0.509**	0.738	**0.382**	0.649
1.1	1.1	**0.258**	0.522	**0.268**	0.484	**0.180**	0.451
	2.5	**0.175**	0.578	**0.245**	0.558	**0.140**	0.493
	100	**0.209**	0.679	**0.291**	0.638	**0.203**	0.566
2.5	2.5	**0.225**	0.545	**0.288**	0.516	**0.170**	0.464
	100	**0.244**	0.617	**0.296**	0.544	**0.185**	0.477
100	100	**0.193**	0.612	**0.251**	0.535	**0.145**	0.486

computer time is recorded based on 3×10^6 generated Dirichlet random vectors with a wide range of selected shape parameters. As can be seen from Tables 4 and 5, the package **rBeta2009** significantly outperforms both packages **MCMCpack** and **compositions** in generating all Dirichlet random vectors considered in this study. In some cases such as $Dirichlet(1.1, 1.1, 1.1)$, $Dirichlet(1.1, 1.1, 1.1, 0.5, 0.5)$, and $Dirichlet(0.5, 0.5, 0.5, 1.1, 1.1)$, the improvement of computer generation time is up to 74% (compared to package **MCMCpack**) and 87% (compared to package **compositions**), respectively. In summary, all these numerical results demonstrate that package **rBeta2009** is very efficient when compared to the existing R packages for generating beta variates and Dirichlet random vectors. In addition, its performance is quite robust to changes in computer platform.

Remark. The R packages **MCMCpack**, **gtools**, **gregmisc**, **BGSIMD**, **phybase**, **MSBVAR**, **mc2d**, and **compositions** all utilize the method based on the transformation of gamma variates (Hogg and Craig, 1978; Aitchison, 1986) to generate Dirichlet random vectors.

4.2. Accuracy

Next, we evaluate the accuracy of package **rBeta2009** by performing goodness-of-fit test with the well-known Kolmogorov–Smirnov (KS) statistic for the generated random numbers/vectors. Briefly, it can be described as follows. Let $x_1, \ldots, x_n$ be n observations drawn from a univariate distribution F and define the corresponding empirical distribution by $\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} I_{\{x_i \le x\}}$, where I represents an indicator function. Let $x_{(1)} \le \cdots \le x_{(n)}$ be the order statistics of $x_1, \ldots, x_n$. The KS statistic is then defined as

$$\begin{aligned}\widetilde{D}_n &= \max_x \left|\hat{F}(x) - F(x)\right| \\ &= \max\left\{0, \max_{1\le j\le n}\left(\frac{j}{n} - D_j\right), \max_{1\le j\le n}\left(D_j - \frac{j-1}{n}\right)\right\},\end{aligned} \tag{3}$$

Table 4
The computer times (in seconds) of packages **rBeta2009** (in boldface), **MCMCpack** (in italic), and **compositions** on various computer platforms. Each computer time is recorded based on 3×10^6 generated *Dirichlet*$(\alpha_1, \alpha_2, \alpha_3)$ random vectors.

$\alpha_1 = \alpha_2$	α_3	Windows 7 SP1			Mac OS X 10.8.2			Ubuntu 12.10		
(32-bit binary)										
0.1	0.1	**1.082**	*1.862*	4.380	**2.014**	*2.480*	7.348			
	0.75	**0.689**	*1.934*	4.255	**1.360**	*2.550*	7.242			
	1.1	**0.910**	*1.842*	4.091	**1.616**	*2.466*	6.987			
	25	**0.993**	*1.622*	3.978	**1.740**	*2.249*	6.755			
0.75	0.1	**0.645**	*2.123*	4.402	**1.185**	*2.734*	7.331			
	0.75	**0.872**	*2.276*	4.629	**1.487**	*2.920*	7.406			
	1.1	**0.768**	*2.199*	4.645	**1.241**	*2.787*	7.362			
	25	**0.723**	*1.940*	4.635	**1.201**	*2.673*	7.134			
1.1	0.1	**0.545**	*1.849*	4.184	**0.971**	*2.479*	7.032			
	0.75	**0.704**	*1.903*	4.201	**1.195**	*2.748*	7.210			
	1.1	**0.559**	*1.842*	4.733	**0.966**	*2.536*	7.085			
	25	**0.648**	*1.758*	4.176	**1.047**	*2.376*	6.898			
25	0.1	**0.490**	*1.495*	3.851	**0.912**	*2.093*	6.631			
	0.75	**0.962**	*1.758*	4.384	**1.406**	*2.276*	6.886			
	1.1	**0.564**	*1.590*	4.199	**0.917**	*2.188*	6.684			
	25	**0.487**	*1.365*	3.992	**0.819**	*1.956*	6.632			
(64-bit binary)										
0.1	0.1	**1.386**	*1.671*	4.138	**1.613**	*1.959*	6.006	**1.059**	*1.339*	3.768
	0.75	**0.899**	*1.663*	4.038	**1.020**	*1.893*	5.429	**0.629**	*1.426*	3.569
	1.1	**1.041**	*1.513*	3.779	**1.059**	*1.860*	5.217	**0.780**	*1.322*	3.378
	25	**1.184**	*1.335*	3.613	**1.195**	*1.692*	5.093	**0.843**	*1.197*	3.339
0.75	0.1	**0.839**	*1.782*	4.155	**0.927**	*2.031*	5.555	**0.565**	*1.534*	3.610
	0.75	**0.848**	*1.935*	4.177	**0.853**	*2.214*	5.624	**0.616**	*1.632*	3.801
	1.1	**0.704**	*1.782*	3.981	**0.780**	*2.139*	5.589	**0.522**	*1.574*	3.549
	25	**0.788**	*1.602*	3.797	**0.784**	*1.995*	5.306	**0.549**	*1.425*	3.432
1.1	0.1	**0.696**	*1.416*	3.725	**0.842**	*1.870*	5.434	**0.459**	*1.327*	3.314
	0.75	**0.639**	*1.559*	3.812	**0.697**	*2.058*	5.296	**0.465**	*1.409*	3.484
	1.1	**0.485**	*1.347*	3.625	**0.605**	*2.097*	5.246	**0.371**	*1.244*	3.411
	25	**0.555**	*1.242*	3.498	**0.682**	*1.881*	5.249	**0.434**	*1.170*	3.239
25	0.1	**0.592**	*1.138*	3.401	**0.762**	*1.591*	5.014	**0.462**	*1.041*	3.157
	0.75	**0.778**	*1.274*	3.535	**0.991**	*1.705*	5.229	**0.581**	*1.165*	3.264
	1.1	**0.437**	*1.097*	3.356	**0.634**	*1.726*	5.001	**0.426**	*1.050*	3.123
	25	**0.406**	*0.930*	3.213	**0.590**	*1.636*	4.770	**0.348**	*0.895*	2.961

where $D_j = F(x_{(j)}), j = 1, \ldots, n$. Clearly, the null hypothesis that observations $x_1, \ldots, x_n$ are from the distribution F is rejected for large values of $\widetilde{D}_n$. To perform the test, the following asymptotic result given by Serfling (1980) is quite useful:

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

For multivariate distributions, the following procedure based on *statistically equivalent blocks* (SEBs) can be utilized to perform the goodness-of-fit test (Tukey, 1947; Foutz, 1980; Alam et al., 1993). Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be n observations of a random vector $\mathbf{X}$ obtained from a k-dimensional distribution $F^{[k]}$ having support $\mathcal{S}$. First, we choose n *cutting functions* $\phi_1, \ldots, \phi_n$ such that $\phi_i(\mathbf{X})$ is a continuous distribution for $i = 1, \ldots, n$. Let $\mathbf{x}_{(1)} = \arg\min_{\mathbf{x}\in\{\mathbf{x}_1,\ldots,\mathbf{x}_n\}} \phi_1(\mathbf{x})$ such that the support $\mathcal{S}$ is cut into two blocks

$$B_1 = \{\mathbf{x} \in \mathcal{S} : \phi_1(\mathbf{x}) \le \phi_1(\mathbf{x}_{(1)})\} \quad \text{and} \quad B_{2\cdots n} = \mathcal{S} \setminus B_1.$$

Next, let $\mathbf{x}_{(2)} = \arg\min_{\mathbf{x}\in\{\mathbf{x}_1,\ldots,\mathbf{x}_n\}\setminus\{\mathbf{x}_{(1)}\}} \phi_2(\mathbf{x})$ so that $B_{2\cdots n}$ is cut into two subblocks

$$B_2 = \{\mathbf{x} \in B_{2\cdots n} : \phi_2(\mathbf{x}) \le \phi_2(\mathbf{x}_{(2)})\} \quad \text{and} \quad B_{3\cdots n} = B_{2\cdots n} \setminus B_2.$$

Continuing in this manner, we see that there exist $\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(n)}$ and $(n+1)$ exclusive blocks $B_1, B_2, \ldots, B_{n+1}$ such that $\bigcup_{i=1}^{n+1} B_i = \mathcal{S}$. Let $D_j = P_F\big(X \in \bigcup_{i=1}^{j} B_i\big) = \sum_{i=1}^{j} P_F(\mathbf{X} \in B_i), j = 1, \ldots, n$. The corresponding KS statistic in (3) can then be obtained. Note that to perform the test for the generated Dirichlet random vectors, we choose the cutting functions suggested by Tukey (1947), namely, $\phi_{i+rk}(\mathbf{x}) = x_i, i = 1, \ldots, k, r = 0, 1, \ldots$, where x_i denotes the i-th coordinate of the vector $\mathbf{x}$. The SEBs constructed in this fashion are simply rectangular blocks.

Table 6 gives the results of the KS test for both beta and Dirichlet generating functions, wherein the empirical frequency (i.e., the probability of rejecting the null hypothesis) is computed based on 1000 Monte Carlo simulations with the sample

Table 5
The computer times (in seconds) of packages **rBeta2009** (in boldface), **MCMCpack** (in italic), and **compositions** on various computer platforms. Each computer time is recorded based on 3×10^6 generated *Dirichlet*$(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$ random vectors.

$\alpha_1 = \alpha_2 = \alpha_3$	$\alpha_4 = \alpha_5$	Windows 7 SP1			Mac OS X 10.8.2			Ubuntu 12.10		
(32-bit binary)										
0.05	0.05	**1.980**	*3.048*	7.449	**3.622**	*3.890*	12.166			
	0.9	**1.288**	*3.412*	7.513	**2.199**	*4.217*	11.813			
	1.1	**1.008**	*3.074*	7.303	**1.741**	*4.090*	11.508			
	8.5	**0.949**	*2.778*	6.908	**1.744**	*3.691*	11.223			
0.9	0.05	**1.540**	*3.582*	7.599	**2.536**	*4.484*	12.053			
	0.9	**1.892**	*3.802*	8.116	**2.965**	*4.850*	12.530			
	1.1	**1.767**	*3.661*	7.811	**2.865**	*4.642*	13.028			
	8.5	**1.839**	*3.272*	7.455	**2.887**	*4.297*	12.056			
1.1	0.05	**1.025**	*3.121*	7.198	**1.712**	*4.170*	11.810			
	0.9	**1.560**	*3.509*	7.460	**2.251**	*4.546*	12.125			
	1.1	**1.255**	*3.156*	7.184	**1.920**	*4.187*	11.916			
	8.5	**1.123**	*2.724*	6.853	**1.850**	*4.110*	11.563			
8.5	0.05	**1.020**	*2.578*	6.684	**1.861**	*3.766*	11.327			
	0.9	**1.518**	*2.785*	6.927	**2.529**	*4.002*	11.712			
	1.1	**1.234**	*2.441*	6.254	**1.793**	*3.861*	11.644			
	8.5	**1.135**	*2.129*	5.938	**1.782**	*3.263*	10.967			
(64-bit binary)										
0.05	0.05	**2.522**	*2.628*	6.474	**2.912**	*3.036*	9.064	**1.963**	*2.076*	5.777
	0.9	**1.500**	*2.873*	6.727	**1.577**	*3.256*	9.082	**1.073**	*2.342*	5.839
	1.1	**1.369**	*2.644*	6.284	**1.420**	*3.353*	9.016	**0.954**	*2.125*	5.548
	8.5	**1.298**	*3.437*	6.970	**1.464**	*2.822*	8.636	**0.947**	*1.902*	5.350
0.9	0.05	**1.706**	*3.259*	7.052	**1.621**	*3.380*	8.961	**1.115**	*2.475*	5.974
	0.9	**1.788**	*3.574*	7.451	**1.974**	*3.592*	9.684	**1.190**	*2.732*	6.211
	1.1	**1.642**	*2.916*	6.830	**1.692**	*3.637*	9.265	**1.172**	*2.508*	5.938
	8.5	**1.791**	*2.564*	6.416	**2.003**	*3.244*	9.350	**1.164**	*2.299*	5.750
1.1	0.05	**1.243**	*2.499*	6.680	**1.304**	*3.186*	8.853	**0.851**	*2.172*	5.523
	0.9	**1.414**	*2.656*	6.733	**1.562**	*3.424*	9.115	**1.023**	*2.400*	5.885
	1.1	**1.142**	*2.445*	6.463	**1.254**	*3.336*	9.039	**0.834**	*2.057*	5.562
	8.5	**0.937**	*2.228*	6.048	**1.238**	*3.273*	8.767	**0.707**	*1.918*	5.337
8.5	0.05	**1.166**	*1.992*	5.805	**1.385**	*2.807*	8.398	**0.853**	*1.872*	5.234
	0.9	**1.464**	*2.556*	6.036	**1.821**	*3.065*	9.194	**0.999**	*2.088*	5.506
	1.1	**0.895**	*1.907*	5.868	**1.219**	*3.049*	8.418	**0.791**	*1.855*	5.240
	8.5	**0.936**	*1.714*	5.714	**1.195**	*2.616*	8.283	**0.665**	*1.536*	5.031

Table 6
The empirical frequencies (probabilities of rejecting the null hypothesis) for the KS test at the 0.05 level of significance based on the generated beta variates and Dirichlet random vectors. Note that each empirical frequency is computed based on 1000 Monte Carlo simulations with the sample size 200.

Beta(α, β)		*Dirichlet*$(\alpha_1, \ldots, \alpha_{k+1})$	
Parameters	Empirical frequency	Parameters	Empirical frequency
(0.1, 0.1)	0.052	(0.1, 0.5, 0.9)	0.065
(0.1, 0.75)	0.052	(0.3, 0.8, 7.5)	0.047
(0.1, 1.2)	0.040	(0.5, 2.2, 3.4)	0.036
(0.1, 13.5)	0.052	(0.5, 1.5, 10)	0.050
(0.75, 0.75)	0.054	(1.5, 5, 10)	0.037
(0.75, 1.2)	0.049	(1.7, 5.0, 15.2)	0.051
(0.75, 13.5)	0.041	(2, 3, 5)	0.054
(1.2, 1.2)	0.042	(0.1, 0.3, 0.5, 0.7, 0.9)	0.042
(1.2, 13.5)	0.052	(0.7, 0.7, 2.7, 2.7, 5.3)	0.048
(13.5, 13.5)	0.054	(1.1, 3.3, 6.6, 11.1, 18.7)	0.059

size 200. As can be seen from Table 6, all the estimated empirical frequencies are quite close to 0.05, which support the accuracy of package **rBeta2009** in terms of goodness-of-fit.

Remark. In addition to the KS statistic introduced in this section, one can refer to Chiu and Liu (2009) for other statistics that can be used to test the goodness-of-fit for multivariate continuous distributions.

Table 7
The empirical frequencies (probabilities of rejecting the null hypothesis) for the Ljung–Box test with $H = 1, 2, 3$ based on the generated beta variates and Dirichlet random vectors. Note that each empirical frequency is computed based on 3000 Monte Carlo simulations with the sample size 3000, while the level of significance is chosen to be 0.05.

$Beta(\alpha, \beta)$				$Dirichlet(\alpha_1, \dots, \alpha_{k+1})$			
Parameter	$H = 1$	$H = 2$	$H = 3$	Parameter	$H = 1$	$H = 2$	$H = 3$
(0.1, 0.1)	0.050	0.050	0.052	(0.2, 0.2, 0.4)	0.046	0.042	0.046
(0.1, 0.7)	0.044	0.049	0.048	(0.2, 0.2, 0.9)	0.050	0.047	0.048
(0.1, 1.1)	0.048	0.050	0.059	(0.2, 0.2, 13)	0.050	0.049	0.052
(0.1, 12.5)	0.040	0.042	0.047	(1.2, 1.2, 0.4)	0.052	0.048	0.053
(0.1, 100)	0.047	0.050	0.052	(1.2, 1.2, 0.9)	0.054	0.047	0.049
(0.7, 0.7)	0.053	0.056	0.058	(1.2, 1.2, 13)	0.054	0.053	0.057
(0.7, 1.1)	0.053	0.056	0.047	(7.5, 7.5, 0.4)	0.051	0.049	0.056
(0.7, 12.5)	0.051	0.055	0.047	(7.5, 7.5, 0.9)	0.054	0.054	0.049
(0.7, 100)	0.053	0.053	0.050	(7.5, 7.5, 13)	0.058	0.054	0.056
(1.1, 1.1)	0.050	0.043	0.047	(0.3, 0.3, 0.5, 0.5, 0.5)	0.046	0.051	0.052
(1.1, 12.5)	0.053	0.051	0.048	(0.3, 0.3, 9.8, 9.8, 9.8)	0.055	0.054	0.058
(1.1, 100)	0.057	0.057	0.055	(0.6, 0.6, 0.5, 0.5, 0.5)	0.056	0.051	0.053
(12.5, 12.5)	0.047	0.048	0.054	(0.6, 0.6, 9.8, 9.8, 9.8)	0.058	0.053	0.050
(12.5, 100)	0.048	0.042	0.046	(1.1, 1.1, 0.5, 0.5, 0.5)	0.049	0.048	0.050
(100, 100)	0.044	0.050	0.047	(1.1, 1.1, 9.8, 9.8, 9.8)	0.050	0.043	0.048

4.3. Randomness

Next, we evaluate the randomness of the two generating functions in package **rBeta2009** by utilizing the Ljung–Box test (Ljung and Box, 1978). It can be briefly described as follows. Let $x_1, \dots, x_n$ be a sequence of observations and denote the autocorrelation of lag h by ρ_h. Then, testing the randomness of the sequence can be equivalently formulated as testing the null hypothesis $H_0 : \rho_1 = \cdots = \rho_H = 0$ for some selected value H. Ljung and Box (1978) introduced the following test statistic (called the Portmanteau statistic):

$$Q_H = n(n+2) \sum_{h=1}^{H} \frac{\hat{\rho}_h^2}{n-h}, \tag{5}$$

where $\hat{\rho}_h$ is the sample autocorrelation of lag h. Note that given the level of significance α, the null hypothesis is rejected if $Q_H > \chi^2_{H,1-\alpha}$.

The multivariate version of the Portmanteau statistic, which were discussed by Hosking (1980, 1981) and Li and McLeod (1981), is described as follows. Let $\mathbf{x}_1, \dots, \mathbf{x}_n$ be a sequence of k-variate observations and denote the cross-correlation matrix of lag h by $\boldsymbol{\rho}_h$. To test the null hypothesis $H_0 : \boldsymbol{\rho}_1 = \cdots = \boldsymbol{\rho}_H = \mathbf{0}$, where $\mathbf{0}$ is a zero matrix of order $k \times k$, we can consider the following statistic:

$$Q_H^{[k]} = n^2 \sum_{h=1}^{H} \frac{1}{n-h} trace\left(\boldsymbol{\Gamma}_h' \boldsymbol{\Gamma}_0^{-1} \boldsymbol{\Gamma}_h \boldsymbol{\Gamma}_0^{-1}\right), \tag{6}$$

where $\boldsymbol{\Gamma}_h$ is the estimated cross-covariance matrix of lag h. Note that given the level of significance α, the null hypothesis is rejected if $Q_H^{[k]} > \chi^2_{k^2H,1-\alpha}$.

To evaluate the randomness of the variates/vectors generated by package **rBeta2009**, we performed the Ljung–Box test by choosing $H = 1, 2, 3$. Table 7 summarizes the empirical frequencies of the test at the 0.05 level of significance based on 3000 Monte Carlo simulations with the sample size 3000. As can be seen, all the estimated empirical frequencies are very close to 0.05, which support the randomness of generating functions in package **rBeta2009**.

5. Summary

We have presented a new R package **rBeta2009** for generating beta variates and Dirichlet random vectors. The package incorporates state-of-the-art generating algorithms so as to minimize the computer generation time. From the design viewpoint, it also has many advantages such as flexible memory allocation, portability, and adaptability. Numerical evidence shows that it significantly outperforms the existing R packages on various hardware and software platforms in terms of computer generation time, while properties such as accuracy (or goodness-of-fit) and randomness of the generated quantities are very good. Finally, it is important to mention that this package can be efficiently used for generating random variates/vectors from some other distributions closely related to the beta distribution, such as the inverted Dirichlet distribution, the Liouville distribution, and the uniform distribution over convex polyhedrons.

References

Aitchison, J., 1986. The Statistical Analysis of Compositional Data. In: Monographs on Statistics and Applied Probability, Chapman & Hall, London, ISBN: 0412280604.

Alam, K., Abernathy, R., Williams, C.L., 1993. Multivariate goodness-of-fit tests based on statistically equivalent blocks. Communications in Statistics—Theory and Methods 22 (6), 1515–1533.

Atkinson, A.C., 1979. The computer generation of Poisson random variables. Journal of the Royal Statistical Society, Series C (Applied Statistics) 28 (1), 29–35.

Atkinson, A.C., Whittaker, J., 1976. A switching algorithm for the generation of beta random variables with at least one parameter less than 1. Journal of the Royal Statistical Society, Series A (General) 139 (4), 462–467.

Cheng, R.C.H., 1978. Generating beta variates with nonintegral shape parameters. Communications of the Association for Computing Machinery 21, 317–322.

Chiu, S.N., Liu, K.I., 2009. Generalized Cramér-von Mises goodness-of-fit tests for multivariate distributions. Computational Statistics & Data Analysis 53 (11), 3817–3834.

da-Silva, C.Q., Migon, H.S., Correia, L.T., 2011. Dynamic Bayesian beta models. Computational Statistics & Data Analysis 55 (6), 2074–2089.

Eddelbuettel, D., François, R., 2011. Rcpp: seamless R and C++ integration. Journal of Statistical Software 40 (8), 1–18. URL http://www.jstatsoft.org/v40/i08.

Foutz, R.V., 1980. A test for goodness of fit based on empirical probability measure. Annals of Statistics 8, 989–1001. URL http://www.jstor.org/stable/2240430.

Gupta, A.K., Nadarajah, S., 2004. Handbook of Beta Distribution and Its Applications. Marcel Dekker, New York, ISBN: 0824753968.

Hogg, R.V., Craig, V.A., 1978. Introduction to Mathematical Statistics, fourth ed. Macmillan, New York, ISBN: 0029789907.

Hosking, J.R.M., 1980. The multivariate Portmanteau statistic. Journal of the American Statistical Association 75 (371), 602–608. URL http://www.jstor.org/stable/2287656.

Hosking, J.R.M., 1981. Lagrange-multiplier tests of multivariate time-series models. Journal of the Royal Statistical Society, Series B (Methodological) 43 (2), 219–230. URL http://www.jstor.org/stable/2984852.

Hung, Y.C., Balakrishnan, N., Cheng, C.W., 2011. Evaluation of algorithms for generating Dirichlet random vectors. Journal of Statistical Computation and Simulation 81 (4), 445–459.

Hung, Y.C., Balakrishnan, N., Lin, Y.T., 2009. Evaluation of beta generation algorithms. Communications in Statistics—Simulation and Computation 38 (4), 750–770.

Jöhnk, M.D., 1964. Erzeugung von betaverteilten und gammaverteilten zufallszahlen. Metrika 8 (1), 5–15.

Lange, K., 1995. Applications of the Dirichlet distribution to forensic match probabilities. Genetica 96 (1–2), 107–117.

Li, W.K., McLeod, A.I., 1981. Distribution of the residual autocorrelations in multivariate ARMA time series models. Journal of the Royal Statistical Society, Series B (Methodological) 43 (2), 231–239. URL http://www.jstor.org/stable/2984852.

Ljung, G.M., Box, G.E.P., 1978. On a measure of lack of fit in time series models. Biometrika 65 (2), 297–303.

Narayanan, A., 1990. Computer generation of Dirichlet random vectors. Journal of Statistical Computation and Simulation 36 (1), 19–30.

R Development Core Team, 2008. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN: 3-900051-07-0, URL http://www.r-project.org/.

R Development Core Team, 2012. Writing R Extensions. ISBN: 3-900051-11-9, URL http://cran.r-project.org/doc/manuals/R-exts.pdf.

Rubinstein, R.Y., Kroese, D.P., 1981. Simulation and the Monte Carlo Method. John Wiley & Sons, New York, ISBN: 0-471-08917-6.

Saito, M., Matsumoto, M., 2008. Simd-oriented fast Mersenne twister: a 128-bit pseudorandom number generator. In: Keller, A., Heinrich, S., Niederreiter, H. (Eds.), Monte Carlo and Quasi-Monte Carlo Methods, 2006. Springer, Heidelberg, pp. 607–622.

Sakasegawa, H., 1983. Stratified rejection and squeeze method for generating beta random numbers. Annals of the Institute of Statistical Mathematics Part B 35, 291–302.

Schmeiser, B.W., Babu, A.J.G., 1980. Beta variate generation via exponential majorizing functions. Operations Research 28, 917–926.

Serfling, R.J., 1980. Approximation Theorems for Mathematical Statistics. John Wiley & Sons, New York.

Tian, G.L., Tang, M.L., Yuen, K.C., Ng, K.W., 2010. Further properties and new applications of the nested Dirichlet distribution. Computational Statistics & Data Analysis 54 (2), 394–405.

Tukey, J.W., 1947. Non-parametric estimation II. Statistically equivalent blocks and tolerance regions-the continuous case. The Annals of Mathematical Statistics 18 (4), 529–539.

Wichmann, B.A., Hill, I.D., 2006. Generating good pseudo-random numbers. Computational Statistics & Data Analysis 51 (3), 1614–1622.

Zechner, H., Stadlober, E., 1993. Generating beta variates via patchwork rejection. Computing 50, 1–18.