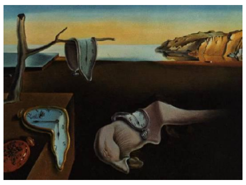

Journal of Statistical Software

March 2011, Volume 39, Issue 11. *http://www.jstatsoft.org/*

Computing the Two-Sided Kolmogorov-Smirnov Distribution

Richard Simard
Université de Montréal

Pierre L'Ecuyer
Université de Montréal

Abstract

We propose an algorithm to compute the cumulative distribution function of the two-sided Kolmogorov-Smirnov test statistic D_n and its complementary distribution in a fast and reliable way. Different approximations are used in different regions of (n, x). Java and C programs are available.

Keywords: KS test, Kolmogorov-Smirnov distribution, goodness of fit.

1. Introduction

The Kolmogorov-Smirnov (KS) *two-sided* test statistic D_n is widely used to measure the goodness-of-fit between the empirical distribution of a set of n observations and a given continuous probability distribution. It is defined by

$$D_n = \sup_x \left| G(x) - \hat{G}_n(x) \right| ,$$

where n is the number of (independent) observations, $\hat{G}_n$ is their empirical cumulative distribution function (cdf), and G is a completely specified continuous theoretical cdf. Let F_n denote the cdf of D_n under the null hypothesis $\mathcal{H}_0$ that the n observations are independent and have cdf G, that is,

$$F_n(x) = \mathsf{P}[D_n \leqslant x \mid \mathcal{H}_0] \qquad \text{for } x \in [0, 1].$$

This F_n is loosely called the KS distribution.

Computing $F_n(x)$ in a fast and reliable way, for arbitrary values of n and x, is not easy. With the help of a symbolic algebra package, Drew, Glen, and Leemis (2000) computed the exact KS distribution for $1 \leqslant n \leqslant 30$ as a set of piecewise polynomials of degree n resulting from a large number of integrals. They also gave a short literature review. Brown and Harvey (2007) consider seven different methods proposed in the past to compute the exact KS distribution. They programmed these seven methods in Mathematica, using only rational numbers to obtain exact results. Although their Mathematica programs are very useful for verification purpose, the calculations are extremely slow for large n. For example, computing $F_n(x)$ for $n = 10000$ and $x = 2/\sqrt{n}$, using the Durbin (1968) recursion formula, takes three hours on our (standard) desktop computer. The Durbin recursion formula gives the fastest exact method among all those programmed by Brown and Harvey (2007). Unfortunately, it contains expressions that suffer from subtractive cancellation between very large terms, and is thus unsuitable for floating-point computations with limited precision. Among the formulæ considered by Brown and Harvey (2007), only those of Noé, Pomeranz, and the Durbin matrix formula are not subject to catastrophic cancellation because they contain only non-negative terms.

The method of Pomeranz (1974) and the Durbin (1968) recursion method were programmed in single precision Fortran 77 by Pomeranz (1974). The Durbin recursion method is numerically unstable and gives sensible results only for very small n. The Pomeranz method makes use of floors and ceilings which must be computed with great care; as an illustration, the KS distribution computed by the program of Pomeranz (1974) sometimes gives a non-smooth distribution with very large errors, as we will see in Section 2.2. Kallman (1977) programmed a generalized version of Pomeranz method in Fortran 77; this program works well in single precision for small n but starts to break down for moderate n and when $F_n(x)$ is close to 1. Miller (1999) programmed the complementary KS distribution $\bar{F}_n(x) = 1 - F_n(x)$ in Fortran 90, using different approximations; his program is very fast, but gives only 0 to 3 decimal digits of precision for $n > 120$ and sometimes returns NaN (not a number), for example for $n = 100$ and $x = 0.54, 0.66, 0.77$, or negative values, for example for $n = 50$ and $0.35 < x < 0.50$.

Recently, Marsaglia, Tsang, and Wang (2003) wrote a C procedure implementing the matrix formula of Durbin (1973). Although it works very well for small n and gives 13 decimal digits of precision according to the authors, it is very slow for large n and sometimes returns NaN or infinite values; for example for $n = 11000$ and $0.000413 \leq x \leq 0.000414$; for $n = 21000$ or 21001 and $0.000434 \leqslant x \leqslant 0.000526$; for $n = 42001$ and $0.000206 \leq x \leq 0.000263$; and for $n = 62000$ and $0.001 \leqslant x \leqslant 0.007$. Furthermore, computing a p-value as $p = \mathsf{P}[D_n \geqslant x] = 1 - F_n(x)$ gives rise to loss of precision when $F_n(x)$ is close to 1, due to subtractive cancellation.

A close look at popular statistical software reveals that even some of the best products use poor approximations for the KS distribution (or the complementary distribution) in some regions (values of (n, x)). For example, in the popular statistical software environment R (2010), the method of Marsaglia *et al.* (2003) was recently implemented to compute the p-value $p = 1 - F_n(x)$ in KS tests (with the option "exact"). However, because this method is very slow for large n, the default algorithm in R for $n \geqslant 100$ is an asymptotic approximation that gives only 0 to 2 decimal digits of precision. For example, for $n = 120$ and $x = 0.0874483967333$, R default method returns $p = 0.3178$ while the correct value is $p = 0.30012$. For $n = 500$ and $x = 0.037527424$, R default method returns $p = 0.482$ while the correct value is $p = 0.47067$. R "exact" (but slow) method gives the correct values in both cases. However for p-values smaller than 10^{-15}, neither method gives any correct decimal.

MATLAB (2009) uses different methods to approximate the KS distribution depending on n and x. For example, for $n = 20$ and $x = 0.8008915818$, MATLAB returns $p = 2.2544 \times 10^{-12}$

while the correct value is $p = 2.5754 \times 10^{-14}$, and for $n = 20$ and $x = 0.9004583223$, it returns $p = 1.5763 \times 10^{-15}$ while the correct value is $p = 1.8250 \times 10^{-20}$. These returned values have zero decimal digits of precision in both cases.

Available programs for the KS distribution in standard programming languages such as C, Fortran, and Java are either fast but unreliable, or precise but slow. Moreover, they either compute only the KS distribution or only its complementary. So there is a need for a fast and reliable program that computes both the KS distribution and its complementary distribution for arbitrary n and x.

In this paper, we describe such an algorithm and its implementation for the KS distribution with floating-point numbers of 53 bits of precision. In Section 2, we briefly describe the exact methods that we use to compute the KS distribution. We implemented the Pomeranz recursion formula and we compare its speed with the Durbin matrix formula. In Section 3, we introduce faster but less precise approximations that we use for large n. In Section 4, we partition the space (n, x) in different regions and we explain why each method is more appropriate in a given region to compute the KS distribution. In Section 5, we do the same for the complementary KS distribution. Finally, we measure the speed of our C program at several values of n and x.

2. Exact methods

2.1. Some values in the tails

There are a few special cases for which the exact KS distribution is known and has a very simple form. In the tails (for x close to 0 or 1), the exact values for arbitrary n are (Ruben and Gambino 1982):

$$F_n(x) = \begin{cases} 0 & \text{for } x \leqslant \frac{1}{2n} \\ n!\left(2x - \frac{1}{n}\right)^n & \text{for } \frac{1}{2n} < x \leqslant \frac{1}{n} \\ 1 - 2(1-x)^n & \text{for } 1 - \frac{1}{n} \leqslant x < 1 \\ 1 & \text{for } 1 \leqslant x. \end{cases} \tag{1}$$

This also provides an exact formula for the complementary cdf $\bar{F}_n(x) \overset{\text{def}}{=} 1 - F_n(x)$ over the same range of values of (n, x). We will use these simple formulas whenever they apply.

2.2. The Pomeranz recursion formula

The Pomeranz formula permits one to compute a $(2n+2) \times (n+1)$ matrix whose entries are defined by a recurrence, and the last of those entries is $F_n(x)$. The method is described in Pomeranz (1974) and Brown and Harvey (2007), and explained below. It makes use of floors and ceilings which must be computed with care, because it is very sensitive to numerical imprecision, as we will see in a moment. Let $t = nx$ and define the $A_1, \ldots, A_{2n+2}$ (which depend on t) as follows:

$$\begin{aligned}
A_1 &= 0, \\
A_2 &= \min\{t - \lfloor t \rfloor, \lceil t \rceil - t\}, \\
A_3 &= 1 - A_2, \\
A_i &= A_{i-2} + 1, \qquad \text{for } i = 4, 5, \ldots, 2n+1, \text{ and} \\
A_{2n+2} &= n
\end{aligned}$$

We then define the entries $V_{i,j}$ of a $(2n+2) \times (n+1)$ matrix $\mathbf{V}$ (which depend on t) by $V_{1,1} = 1$, $V_{1,j} = 0$ for $j = 2, \ldots, n+1$, and

$$V_{i,j} = \sum_{k=k_1(i,j)}^{k_2(i,j)} \frac{((A_i - A_{i-1})/n)^{j-k}}{(j-k)!} V_{i-1,k} \tag{2}$$

for $i = 2, \ldots, 2n+2$ and $j = \lfloor A_i - t \rfloor + 2, \ldots, \lceil A_i + t \rceil$, where

$$k_1(i,j) = \max(1, \lfloor A_{i-1} - t \rfloor + 2) \quad \text{and} \quad k_2(i,j) = \min(j, \lceil A_{i-1} + t \rceil). \tag{3}$$

Then we have

$$F_n(x) = n!\, V_{2n+2,n+1}. \tag{4}$$

As observed by Pomeranz, the differences $A_i - A_{i-1}$ can take at most four different values. This can be exploited by precomputing the factors $((A_i - A_{i-1})/n)^j/j!$ in (2), and this increases the speed of computation significantly. It is also clear from (2) that we never need to store more than two rows of matrix $\mathbf{V}$ at a time, since row i can be computed from row $i-1$ only.

Pomeranz's algorithm uses the floor and ceiling of $A_i \pm t$ to define the lower and upper bounds of summation in (2), which are defined in (3). For some values of t (and x), unavoidable round-off errors in floating-point calculations will give the wrong value of $k_1(i,j)$ or $k_2(i,j)$, and this may give rise to large errors in the cdf. As an illustration, Figure 1 shows (in blue) the KS distribution $F_n(x)$ as computed by the program `487.f` of Pomeranz (1974) for $n = 20$ and $0.17 \leqslant x \leqslant 0.19$, compared with the correct distribution (in red). The zigzags in the blue line are due to floating-point errors in computing $k_1(i,j)$ and $k_2(i,j)$.

To avoid this type of problem, we precompute exactly all floors and ceilings of $A_i \pm t$ in (3) for the given value of $t = nx$, although nx itself is not computed exactly due to floating-point error in the multiplication. We decompose $t = \ell + f$, where ℓ is an non-negative integer and $0 \leqslant f < 1$. There are three cases to consider depending on where the fractional part f lies. We must make sure to identify correctly in which case we are, and use the corresponding formulas for the precomputations.

Case (i): $f = 0$. In this case, we have

$$\begin{array}{lll}
A_{2i} = i - 1 & A_{2i+1} = i, & \text{for } i = 1, 2, \ldots \\
\lfloor A_{2i} - t \rfloor = i - 1 - \ell & \lceil A_{2i} + t \rceil = i - 1 + \ell, & \text{for } i = 1, 2, \ldots \\
\lfloor A_{2i+1} - t \rfloor = i - \ell & \lceil A_{2i+1} + t \rceil = i + \ell, & \text{for } i = 0, 1, 2, \ldots
\end{array}$$

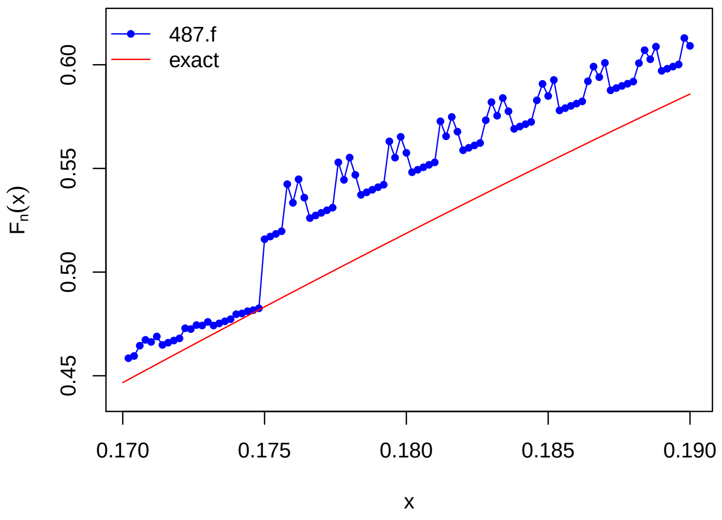

Figure 1: The KS distribution for $n = 20$ computed by program 487.f (in blue) compared with the correct cdf (in red).

Case (ii): $0 < f \leqslant 1/2$. Here we have

$$\begin{array}{lll}
A_{2i} = i - 1 + f & A_{2i+1} = i - f, & \text{for } i = 1, 2, \dots \\
\lfloor A_1 - t \rfloor = -\ell - 1 & \lceil A_1 + t \rceil = \ell + 1 & \\
\lfloor A_{2i} - t \rfloor = i - 1 - \ell & \lceil A_{2i} + t \rceil = i + \ell, & \text{for } i = 1, 2, \dots \\
\lfloor A_{2i+1} - t \rfloor = i - 1 - \ell & \lceil A_{2i+1} + t \rceil = i + \ell, & \text{for } i = 1, 2, \dots
\end{array}$$

Case (iii): $1/2 < f < 1$. Here we have

$$\begin{array}{lll}
A_{2i} = i - f & A_{2i+1} = i - 1 + f, & \text{for } i = 1, 2, \dots \\
\lfloor A_{2i} - t \rfloor = i - 2 - \ell & \lceil A_{2i} + t \rceil = i + \ell, & \text{for } i = 1, 2, \dots \\
\lfloor A_{2i+1} - t \rfloor = i - 1 - \ell & \lceil A_{2i+1} + t \rceil = i + 1 + \ell, & \text{for } i = 0, 1, 2, \dots
\end{array}$$

With floating-point numbers that follow the IEEE-754 64-bit standard for double precision, the algorithm cannot be used in this naive form as soon as $n > 140$, because for some regions of x, the terms $V_{i,j}$ will underflow to 0 while the $n!$ factor will overflow, even though the corresponding probability is finite. For this reason, we must periodically renormalize all elements of row i of $\mathbf{V}$ (in our implementation, we multiply all elements of the row by 2^{350}) when the smallest element of the row becomes too small (smaller than 10^{-280} in our implementation) and we keep track of the logarithm of the renormalization factors to recover the correct numbers.

2.3. The Durbin matrix formula

Durbin (1973) proposed a formula for $F_n(x)$ based on the calculation of a $k \times k$ matrix $\mathbf{H}$ raised to the power n, where $k = \lceil t \rceil = \lceil nx \rceil$; see his equations (2.4.3) and (2.4.4); see also Brown and Harvey (2007). Then $F_n(x)$ is given by $n!\,T/n^n$, where T is the element $k \times k$ of the matrix $\mathbf{H}^n$. Marsaglia *et al.* (2003) have recently implemented this algorithm in a C program which we shall call MTW (Marsaglia-Tsang-Wang). We reuse their program for this method, with small corrections.

2.4. Comparison between Pomeranz and Durbin algorithms for small n

One of our objectives in this work was to provide a reliable implementation of Pomeranz algorithm and compare its speed and precision with the Durbin matrix algorithm implemented in MTW (its main competitor), for small n. We implemented the Pomeranz algorithm in C and Java, while MTW is implemented in C. For n not too large, these two implementations both return 13 to 15 decimal digits of precision (as measured by comparing with the exact values provided by the Mathematica programs of Brown and Harvey 2007). We also compared their speed for computing $F_n(x)$ at values of (n, x) selected as follows. The mean of D_n (under $\mathcal{H}_0$) is close to

$$\mu_0 \overset{\text{def}}{=} \ln(2)\sqrt{\pi/(2n)} \approx 0.868731160636/\sqrt{n}$$

and we selected the pairs (n, x) around this mean. More specifically, we took $x = a\mu_0$ for $a = 1/4$, $1/3$, $1/2$, 1, 2, and 3, for n ranging from 10 to 1000. The values of $F_n(x)$ for these pairs (n, x) are given in Table 1. For each pair (n, x) in the table, we measured the CPU time (in seconds) needed to compute $F_n(x)$ 10^6 times with the C programs. All the timings reported in this paper were made on a computer with an Advanced Micro Devices (AMD)

$n\backslash x$	$\mu_0/4$	$\mu_0/3$	$\mu_0/2$	μ_0	$2\mu_0$	$3\mu_0$
10	1.9216×10^{-8}	5.7293×10^{-5}	0.021523	0.63157	0.99769	0.9999999
50	2.2810×10^{-9}	1.9914×10^{-5}	0.014262	0.59535	0.99618	0.9999987
100	1.0020×10^{-9}	1.3267×10^{-5}	0.012461	0.58616	0.99587	0.9999982
200	4.9331×10^{-10}	9.5266×10^{-6}	0.011212	0.57949	0.99566	0.9999980
500	2.3705×10^{-10}	6.8500×10^{-6}	0.010131	0.57343	0.99549	0.9999978
1000	1.5699×10^{-10}	5.7174×10^{-6}	0.009597	0.57032	0.99541	0.9999977

Table 1: Values of $F_n(x)$ for selected pairs (n, x).

$n\backslash x$	$\mu_0/4$	$\mu_0/3$	$\mu_0/2$	μ_0	$2\mu_0$	$3\mu_0$
10	0.17	0.17	2.5	3.5	5.2	6.5
50	9.6	11	15	32	74	124
100	22	26	42	100	266	498
200	57	79	128	325	1257	3266
500	221	324	592	2314	1.9×10^4	3.1×10^4
1000	708	1112	2070	1.8×10^4	5.6×10^4	7.9×10^4

Table 2: CPU time (seconds) to compute $F_n(x)$ 10^6 times with the Pomeranz algorithm.

$n\backslash x$	$\mu_0/4$	$\mu_0/3$	$\mu_0/2$	μ_0	$2\mu_0$	$3\mu_0$
10	0.57	0.57	1.6	3.4	21	60
50	2.5	5.3	9.6	48	264	782
100	5.9	5.7	21	106	754	2468
200	14	27	62	331	2200	7012
500	37	85	221	1601	1.2×10^4	7.4×10^4
1000	96	242	615	4608	3.8×10^4	1.3×10^5

Table 3: CPU time (seconds) to compute $F_n(x)$ 10^6 times with the Durbin matrix algorithm.

Athlon processor 4000+ at clock speed of 2400 MHz, running Red Hat Linux. The results are shown in Table 2 for the Pomeranz algorithm and in Table 3 for the Durbin matrix algorithm. We see that the CPU times increase with n and with x (for fixed n). For small n, the Durbin matrix program is faster for $x < \mu_0$ (roughly), while the Pomeranz program is faster for $x \geqslant \mu_0$. When n is large, especially for $x > \mu_0$, these exact algorithms are too slow and have to be replaced by approximations.

3. Asymptotic approximations

Kolmogorov (1933) has proved the following expansion which provides an approximation of $F_n(z/\sqrt{n})$ for large n:

$$\lim_{n\to\infty} F_n(z/\sqrt{n}) = \lim_{n\to\infty} \mathsf{P}[\sqrt{n}D_n \leqslant z \mid \mathcal{H}_0] = K_0(z) = 1 + 2\sum_{k=1}^{\infty}(-1)^k e^{-2k^2z^2}. \tag{5}$$

Pelz and Good (1976) generalized Kolmogorov's approximation (5) to an asymptotic series in $1/\sqrt{n}$ of the form

$$\lim_{n\to\infty} \mathsf{P}[\sqrt{n}D_n \leqslant z \mid \mathcal{H}_0] = K_0(z) + \frac{K_1(z)}{n^{1/2}} + \frac{K_2(z)}{n} + \frac{K_3(z)}{n^{3/2}} + O\left(\frac{1}{n^2}\right), \tag{6}$$

where

$$\begin{aligned}
K_0(z) &= \frac{\sqrt{2\pi}}{z}\sum_{k=1}^{\infty} e^{-\pi^2(2k-1)^2/(8z^2)},\\
K_1(z) &= \frac{1}{6z^4}\sqrt{\frac{\pi}{2}}\sum_{k=-\infty}^{\infty}\left\{\pi^2(k+\tfrac{1}{2})^2 - z^2\right\} e^{-\pi^2(k+1/2)^2/(2z^2)},\\
K_2(z) &= \frac{1}{72z^7}\sqrt{\frac{\pi}{2}}\sum_{k=-\infty}^{\infty}\left\{(6z^6+2z^4) + \pi^2(2z^4-5z^2)(k+\tfrac{1}{2})^2\right.\\
&\qquad \left. + \pi^4(1-2z^2)(k+\tfrac{1}{2})^4\right\} e^{-\pi^2(k+1/2)^2/(2z^2)}\\
&\qquad - \frac{1}{36z^3}\sqrt{\frac{\pi}{2}}\sum_{k=-\infty}^{\infty}\pi^2k^2e^{-\pi^2k^2/(2z^2)}, \quad \text{and}
\end{aligned}$$

$$\begin{aligned}K_3(z) = \frac{1}{6480z^{10}}\sqrt{\frac{\pi}{2}}\sum_{k=-\infty}^{\infty}\Big\{&\pi^6(k+\tfrac{1}{2})^6(5-30z^2)+\pi^4(k+\tfrac{1}{2})^4(-60z^2+212z^4)\\&+\pi^2(k+\tfrac{1}{2})^2(135z^4-96z^6)-(30z^6+90z^8)\Big\}\,e^{-\pi^2(k+1/2)^2/(2z^2)}\\&+\frac{1}{216z^6}\sqrt{\frac{\pi}{2}}\sum_{k=-\infty}^{\infty}(-\pi^4k^4+3\pi^2k^2z^2)e^{-\pi^2k^2/(2z^2)}.\end{aligned}$$

These authors claim that their approximation is accurate to 5 decimal digits for $n \geqslant 100$ (the absolute error of their approximation is in fact smaller than 10^{-5} for $n \geqslant 100$, but in our work we focus on the relative error). Notice that $K_0(z)$ in the above equation is equivalent to (5) through some of Jacobi's identities on theta functions.

Another approximation useful for x close to 1 is based on the complementary cdf of the *one-sided* KS statistic, defined by

$$D_n^+ = \sup_x\{\hat{G}_n(x) - G(x)\},$$

where n, $\hat{G}_n$, and G are defined as in Section 1. The upper tail probability of D_n^+, defined as $p_n^+(x) = \mathsf{P}[D_n^+ \geqslant x \mid \mathcal{H}_0]$, is much faster and easier to compute than $F_n(x)$. Smirnov's stable formula gives

$$p_n^+(x) = x\sum_{j=0}^{\lfloor n(1-x)\rfloor}\binom{n}{j}\left(\frac{j}{n}+x\right)^{j-1}\left(1-x-\frac{j}{n}\right)^{n-j} \tag{7}$$

Miller (1956) has shown that for $p_n^+(x)$ not too large, a very good approximation for the distribution of D_n in the upper tail is

$$\bar{F}_n(x) = \mathsf{P}[D_n \geqslant x \mid \mathcal{H}_0] \approx 2p_n^+(x). \tag{8}$$

But as $p_n^+(x)$ increases (and x decreases), the reliability of this approximation deteriorates quickly as we shall see in the next section.

4. The selected method as a function of (n, x)

In our implementation, we have selected the method that seems to make the best compromise between speed and precision as a function of (n, x), in each area of the $\mathbb{N}\times[0,1]$ space. We partitioned this space as illustrated in Figure 2, where the minimal number of decimal digits of precision of our algorithm is given in parenthesis in each area of the partition (we use the relative error everywhere). For practical reasons, we use approximations when n is large or when x is too close to 1, because the exact methods are then far too slow. The precision of these approximations is very poor for small n or x, and improves as n or x increases. We start by partitioning in three regions according to the value of n: (i) $n \leqslant 140$, (ii) $140 < n \leqslant 10^5$, and (iii) $n > 10^5$. Note that the horizontal line below the Pelz-Good region in Figure 2 is at $n = 140$. Next, we explain what we do in each of those three regions; more details can be found in the appendix at the end of the paper.

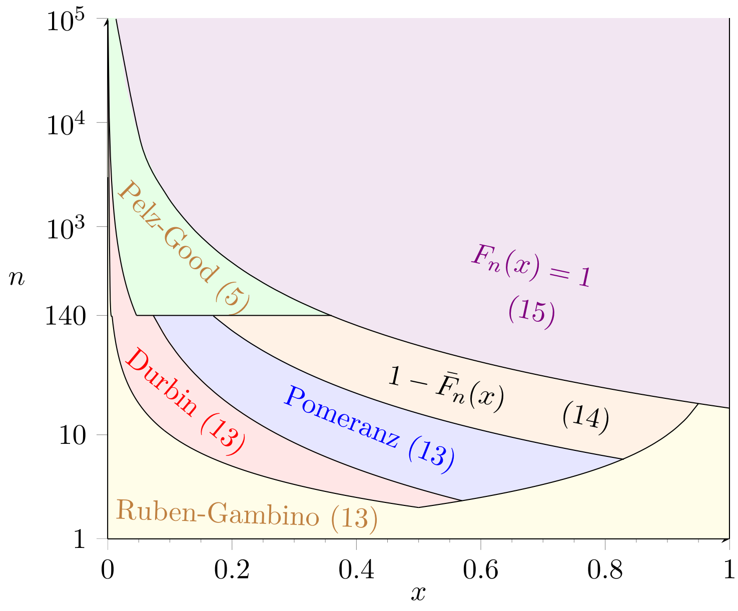

Figure 2: Choice of method to compute (or approximate) $F_n(x)$ and minimal number of decimal digits of precision for $F_n(x)$, as a function of (n, x), in each region of $\mathbb{N} \times [0, 1]$.

(i) $n \leqslant 140$**.** For $n \leqslant 140$, we use the Ruben-Gambino formula (1) whenever it applies, which is for $x \leqslant 1/n$ and for $x \geqslant 1 - 1/n$, the Durbin matrix algorithm for $1/n < nx^2 < 0.754693$, and the Pomeranz algorithm for $0.754693 \leqslant nx^2 < 4$. When $4 \leqslant nx^2 < 18$, we compute the approximation of the complementary cdf $\bar{F}_n(x)$ given by (8) and we return $F_n(x) = 1 - \bar{F}_n(x)$. This is in the region labeled "$1 - \bar{F}_n(x)$" in the figure. When $nx^2 \geqslant 18$, we just return $F_n(x) = 1$. The program returns at least 13 to 15 decimal digits of precision everywhere for $n \leqslant 140$. The reasons for this partition are as follows.

A look at Kolmogorov's original approximation (5) or the Pelz-Good formula (6) shows that the tail probability of $F_n(x)$ is approximately a function of $z = \sqrt{n}x$ only. A closer examination reveals that in general, except when n is very small, the curves $nx^2 = c$, where c is a constant, are curves of almost constant probability (see Table 1 for some typical values of x). Thus regions of constant nx^2 seem relevant for the different approximations of $F_n(x)$.

If $\bar{F}_n(x) < 2.2 \times 10^{-16}$, the double precision representation of $F_n(x)$ cannot be distinguished from 1, so we can just return $F_n(x) = 1$. The second term in Kolmogorov's formula (5) is $2e^{-2z^2}$ and is less than 2.2×10^{-16} when $z^2 > 18.37$. More precise calculations show that in fact $\bar{F}_n(x) < 5 \times 10^{-16}$ whenever $nx^2 \geqslant 18$. Thus we shall just return $F_n(x) = 1$ whenever $nx^2 \geqslant 18$, and this gives 15 digits of precision in this region, labeled "$F_n(x) = 1$" in Figure 2.

Comparing the results of the approximation of $\bar{F}_n(x)$ by (8) with the exact values from the Brown and Harvey (2007) Mathematica programs, we find that it returns at least 10 decimal digits of precision as long as $\bar{F}_n(x) < 10^{-3}$ when $n \leqslant 140$. To determine the region where $\bar{F}_n(x) < 10^{-3}$ (approximately) in terms of n and x, we first use Kolmogorov's formula (5) which gives $K_0(2) = 0.99933$ for $nx^2 = 4$. More precise calculations with exact methods show that for $n \leqslant 140$, we have $\bar{F}_n(x) < 0.0006$ (and thus $F_n(x) > 0.9994$) whenever $nx^2 \geqslant 4$. When we use the approximation (8) in the region $nx^2 \geqslant 4$, the absolute error on $\bar{F}_n(x)$ is always smaller than 10^{-14} and thus computing $F_n(x) = 1 - \bar{F}_n(x)$ gives 14 decimal digits of precision for $F_n(x)$.

In the region that remains, we use one of the exact methods of Durbin and Pomeranz. Based on the speed comparison in Section 2.4, we choose Durbin when $nx^2 < 0.754693$ and Pomeranz otherwise. We avoid the Pelz-Good approximation because it returns less than 5 correct digits for $n \leq 140$.

(ii) $140 < n \leqslant 10^5$**.** Whenever $nx^2 \geqslant 18$, we simply return $F_n(x) = 1$ and this gives 15 digits of precision as explained above. Otherwise, we use the Pelz-Good asymptotic series (6) everywhere in the remaining region, except for x very close to 0 where it is not very good and where we use the Durbin matrix algorithm. As n increases, the region where the Pelz-Good approximation is good gets larger and closer to $x = 0$. We could use again a curve $nx^2 =$ *constant* as a separation between these two regions, but we use instead the empirical curve $nx^{3/2} = 1.4$ as separation, which permits the use of the Pelz-Good approximation in a slightly larger region near $x = 0$; this is more efficient since the Durbin matrix algorithm is very slow for larger n. Thus we use the Durbin matrix algorithm for $nx^{3/2} < 1.4$ and the Pelz-Good asymptotic series for $nx^{3/2} \geqslant 1.4$. Our program returns at least 5 decimal digits of precision everywhere.

(iii) $n > 10^5$**.** We use $F_n(x) = 1$ for $nx^2 \geqslant 18$ and the Pelz-Good asymptotic series everywhere for $nx^2 < 18$. Let $u = F_n(x)$. Then for $n = 100001$, our program returns at least 5 decimal digits of precision for all $u > 10^{-16}$, at least 2 decimal digits of precision for $u > 10^{-56}$, and at least 1 decimal digit of precision for all $u > 10^{-108}$. The Pelz-Good approximation becomes better as n increases and, for a given n, it also becomes better as x increases. For example, for $n = 10^6$, our program returns at least 5 decimal digits of precision for all $u > 10^{-33}$, at least 2 decimal digits of precision for $u > 10^{-120}$, and at least 1 decimal digit of precision for all $u > 10^{-230}$. For such small values of u, a high precision for $F_n(x)$ is less important since 1 or 2 decimal digits of precision suffices to reject a statistical hypothesis with confidence.

5. The complementary cdf

The complementary cdf is defined by $\bar{F}_n(x) = 1 - F_n(x)$. Direct use of this formula in the upper tail, where $F_n(x) \approx 1$, would cause loss of precision. For this reason, we use specialized methods for x close to 1.

When x is close enough to 0 or 1, we use the complementary of the Ruben-Gambino exact formula (1). We split the remaining region in two: $n \leqslant 140$ and $n > 140$.

(i) $n \leqslant 140$. For x close to 1, we base our computation of $\bar{F}_n(x)$ on the distribution of the *one-sided* Kolmogorov-Smirnov statistic, as given by the Miller approximation (8). In the region where $nx^2 \geqslant 4$, we always have $\bar{F}_n(x) < 0.0006$. Comparing the approximation (8) with the exact values obtained from Brown and Harvey's Mathematica program, we see that the approximation returns at least 11 decimal digits of precision for $\bar{F}_n(x)$ in this region.

When $nx^2 < 4$, we simply take $\bar{F}_n(x) = 1 - F_n(x)$ with $F_n(x)$ computed by the method specified in Section 4. Since $F_n(x) < 0.99993$ for all x in this region as long as $n \geqslant 6$,

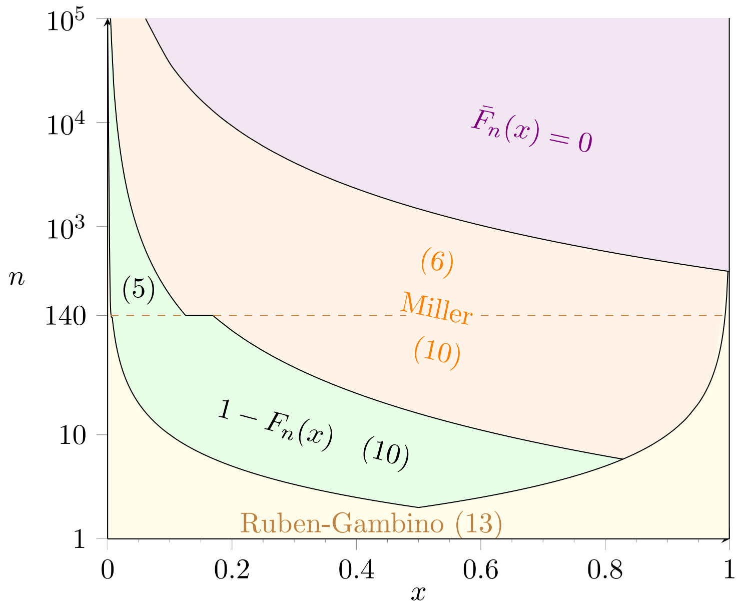

Figure 3: Choice of method to compute (or approximate) $\bar{F}_n(x)$ and minimal number of decimal digits of precision for $\bar{F}_n(x)$, as a function of (n, x), in each region of $\mathbb{N} \times [0, 1]$.

subtractive cancellation may cause the loss of at most 4 decimal digits of precision. For $n < 6$, x is either larger than 1 or in the region where we use the exact formula in (1), so there is no loss of precision there.

Overall, $n \leqslant 140$, our program returns at least 10 decimal digits of precision everywhere for $\bar{F}_n(x)$ (see Figure 3).

(ii) $n > 140$**.** The minimal value of a floating-point number represented in the IEEE-754 64-bit standard for double precision is $\approx 10^{-308}$. If we impose the constraint $2e^{-2z^2} < 10^{-308}$ to the dominant term in (5), we obtain the condition $z^2 > 355$. More precise calculations for different values of n with the Miller approximation (8) show that we can safely set $\bar{F}_n(x) = 0$ whenever $nx^2 \geqslant 370$, because $\bar{F}_n(x)$ is then always smaller than 10^{-308}; this corresponds to region "$\bar{F}_n(x) = 0$" in Figure 3.

Since for $n > 140$, the Pelz-Good approximation (6) returns 5 decimal digits of precision, we shall use the Miller approximation (8) with the Smirnov formula (7) in the larger region $nx^2 \geqslant 2.2$. In this region, we have $\bar{F}_n(x) < 0.025$ and the Miller approximation returns at least 6 decimal digits of precision.

For $nx^2 < 2.2$, we simply use $\bar{F}_n(x) = 1 - F_n(x)$. Since for $n > 140$, we have at least 5 decimal digits of precision on $F_n(x)$ in the region where $F_n(x) > 10^{-12}$, the program returns at least 5 decimal digits of precision everywhere for $\bar{F}_n(x)$ in the region $nx^2 < 2.2$.

Overall, for $140 < n \leq 200000$, our program for $\bar{F}_n(x)$ returns at least 5 decimal digits of precision everywhere. And for $n > 200000$, it returns a few correct decimal digits.

6. The C and Java programs

The C program in file `KolmogorovSmirnovDist.c` has two public functions:

- `KScdf(int n, double x)`, which computes the cdf $F_n(x)$ for given n and x, and
- `KSfbar(int n, double x)`, which computes the complementary cdf $\bar{F}_n(x)$ for given n and x.

The Java program in file `KolmogorovSmirnovDist.java` has two public static methods:

- `cdf(int n, double x)`, which computes the cdf $F_n(x)$ for given n and x, and
- `fbar(int n, double x)`, which computes the complementary cdf $\bar{F}_n(x)$ for given n and x.

The programs are available along with this manuscript and at `http://www.iro.umontreal.ca/~simardr/ksdir/`.

7. Speed comparison

Tables 4 and 5 report the CPU times needed to compute $F_n(x)$ 10^6 times, for selected values of n and x as in Table 1, with MTW and our C program. Note that for $x = 3\mu_0$, MTW

$n\backslash x$	$\mu_0/4$	$\mu_0/3$	$\mu_0/2$	μ_0	$2\mu_0$	$3\mu_0$
10	0.57	0.57	1.6	3.4	21	60
100	5.9	5.7	21	106	754	0.1
140	6.7	12	41	206	1315	0.1
141	7.0	12.4	44	225	1444	0.1
1000	96	242	615	4.6×10^3	3.8×10^4	0.1
10000	2.8×10^3	6.2×10^3	2.1×10^4	1.9×10^5	3.5×10^6	0.1
100000	1.1×10^5	2.7×10^5	1.1×10^6	4.3×10^7	4.4×10^8	0.1

Table 4: CPU time (seconds) to compute $F_n(x)$ 10^6 times with the Marsaglia-Tsang-Wang C program.

$n\backslash x$	$\mu_0/4$	$\mu_0/3$	$\mu_0/2$	μ_0	$2\mu_0$	$3\mu_0$
10	0.17	0.17	1.6	3.4	5.1	0.66
100	6.5	6.3	23	98	260	22
140	7.3	13	41	174	500	32
141	7.6	13	44	1.4	2.3	3.3
1000	100	247	0.95	1.4	2.3	3.3
10000	2875	0.95	0.95	1.4	2.3	3.3
100000	0.95	0.95	0.95	1.4	2.3	3.3

Table 5: CPU time (seconds) to compute $F_n(x)$ 10^6 times with the Simard-L'Ecuyer C program.

uses the quick approximation given by the authors, which returns only 7 decimal digits of precision. We do not use their approximation since for $n \leqslant 140$, the cdf is much less precise than our program, and using this formula to compute $\bar{F}_n(x) = 1 - F_n(x)$ will give a very bad approximation. For $n > 140$, our program is considerably faster than MTW. On the other hand, the MTW program often gives more precision than our asymptotic expansions when it returns sensible values for $n > 140$. However, the MTW program loses all precision when it is used to compute $\bar{F}_n(x)$ and the latter is smaller than 10^{-15}, whereas our program returns sensible values for $\bar{F}_n(x)$ even when the latter is as small as 10^{-307}.

References

Brown JR, Harvey ME (2007). "Rational Arithmetic Mathematica Functions to Evaluate the One-Sided One-Sample K-S Cumulative Sampling Distribution." *Journal of Statistical Software*, **19**(6), 1–40. URL http://www.jstatsoft.org/v19/i06/.

Drew JH, Glen AG, Leemis LM (2000). "Computing the Cumulative Distribution Function of the Kolmogorov-Smirnov Statistic." *Computational Statistics & Data Analysis*, **34**, 1–15.

Durbin J (1968). "The Probability that the Sample Distribution Function Lies between Two Parallel Straight Lines." *The Annals of Mathematical Statistics*, **39**, 398–411.

Durbin J (1973). "Distribution Theory for Tests Based on the Sample Distribution Function." In *SIAM CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, PA.

Kallman R (1977). "Three Algorithms for Computing Kolmogorov-Smirnov Probabilities with Arbitrary Boundaries and a Certification of Algorithm 487." *ACM Transactions on Mathematical Software*, **3**(3), 285–294. URL http://calgo.acm.org/519.gz.

Kolmogorov A (1933). "Sulla Determinazione Empirica di una Legge di Distribuzione." *Giornale dell' Istituto Italiano degli Attuari*, **4**, 83–91.

Marsaglia G, Tsang WW, Wang J (2003). "Evaluating Kolmogorov's Distribution." *Journal of Statistical Software*, **8**(18), 1–4. URL http://www.jstatsoft.org/v08/i18/.

Miller AJ (1999). "Module Kolmogorov-Smirnov." URL http://www.cmis.csiro.au/Alan_Miller/KS2.f90.

Miller LH (1956). "Table of Percentage Points of Kolmogorov Statistics." *Journal of the American Statistical Association*, **51**, 111–121.

Pelz W, Good IJ (1976). "Approximating the Lower Tail-Areas of the Kolmogorov-Smirnov One-sample Statistic." *Journal of the Royal Statistical Society B*, **38**(2), 152–156.

Pomeranz J (1974). "Exact Cumulative Distribution of the Kolmogorov-Smirnov Statistic for Small Samples (Algorithm 487)." *Communications of the ACM*, **17**(12), 703–704. URL http://calgo.acm.org/487.gz.

R Development Core Team (2010). *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/.

Ruben H, Gambino J (1982). "The Exact Distribution of Kolmogorov's Statistic D_n for $n \leq 10$." *Annals of the Institute of Statistical Mathematics*, **34**, 167–173.

The MathWorks, Inc (2009). *MATLAB – The Language of Technical Computing, Version 7.9.0 (R2009b).* The MathWorks, Inc., Natick, Massachusetts. URL http://www.mathworks.com/products/matlab/.

A. Selection of regions

Here we give a few details on the calculations used to select the regions in Section 4 and Section 5. First, we computed the distribution function $F_n(x)$ at many values of x for $n \leqslant 1000$ both with the Durbin matrix formula and the Pomeranz formula (in our C program); we also compared these results with the exact Brown-Harvey Mathematica program at a few random values of x and n: all three always agreed to at least 13 decimal digits of precision. For $n > 1000$, the Mathematica program that computes $F_n(x)$ becomes very slow, so we used mostly the Durbin matrix program to check how good the other approximations are for large n.

Region $nx^2 \geqslant 18$. Consider the values of $x = \sqrt{18/n}$ for several values of n. We compute the KS complementary distribution function $\bar{F}_n(x)$ both with the Brown-Harvey Mathematica program and also with the Miller approximation (8). The results are given in Table 6 with the relative error between the two, defined as $|(u-v)/u|$, where u is the Mathematica result and v the Miller approximation. (For $n \geqslant 10000$, the Mathematica program is so slow that it takes many days to compute $\bar{F}_n(x)$; this explains the blank entries in Table 6.) Since $\bar{F}_n(x)$ is a decreasing function of x, we conclude that $\bar{F}_n(x)$ is smaller than 5×10^{-16} for $nx^2 \geqslant 18$ and $n \leqslant 10^9$, and almost certainly for any n (from Table 6). Thus, setting $F_n(x) = 1$ gives 15 decimal digits of precision for $F_n(x)$ in this region.

Region $nx^2 \geqslant 4$ and $n \leqslant 140$. Consider the values of $x = \sqrt{4/n}$ for several values of n. We again compare the KS complementary distribution function computed with the Brown-Harvey Mathematica program and also with the Miller approximation (8). The results are given in Table 7 with both the *relative* error ($\approx 10^{-11}$) and the *absolute* error ($< 10^{-14}$). Since the Miller approximation becomes better as x increases for a given n, we conclude that it gives at least 11 decimal digits of precision for $\bar{F}_n(x)$, and at least 14 decimal digits of precision for $F_n(x) = 1 - \bar{F}_n(x)$ in this region.

Region $nx^2 \geqslant 2.2$ and $140 < n \leqslant 10^5$. Consider the values of $x = \sqrt{2.2/n}$ for several values of n. We compare the Miller approximation (8) for the KS complementary distribution function $\bar{F}_n(x)$ both with the Brown-Harvey Mathematica program and with the Durbin matrix C program. The results are shown in Tables 8 and 9 with the relative error of the Miller approximation. Since the Miller approximation becomes better as x increases for a given n, we conclude that it gives at least 6 decimal digits of precision for $\bar{F}_n(x)$ in this region.

Region $nx^{3/2} \geqslant 1.4$ and $140 < n \leqslant 10^5$ for $F_n(x)$. Here we compare the Pelz-Good approximation on the separating curve $nx^{3/2} = 1.4$ with the Mathematica program, and also with the Durbin matrix C program for a few values of (n, x). The results are shown in Tables 10 and 11. After testing many values of $F_n(x)$ at different $n > 140$ and x, we concluded that the Pelz-Good approximation becomes better as x increases for a given n, and that it gives at least 5 decimal digits of precision for $F_n(x)$ in the region $nx^{3/2} \geqslant 1.4$. The Pelz-Good approximation becomes worse as x gets close to 0, and so for $nx^{3/2} < 1.4$, we use the Durbin matrix program.

n	x	Miller	Mathematica	Rel. err.
50	0.6	9.634070456142e-18	9.63407045614234e-18	3.5e-14
100	0.424264068711929	7.606532198486e-17	7.60653219848661e-17	8.0e-14
500	0.189736659610103	3.09340954272e-16	3.09340954272345e-16	2.4e-12
1000	0.134164078649987	3.6959926424e-16	3.69599264245350e-16	5.3e-12
5000	0.06	4.3371233237e-16	4.33712332378453e-16	2.5e-11
10000	0.042426406871193	4.448626200e-16		
50000	0.018973665961010	4.56828378e-16		
10^5	0.013416407864999	4.59148616e-16		
10^6	0.004242640687119	4.6253138e-16		
10^7	0.001341640786500	4.634834e-16		
10^8	0.000424264068712	4.637718e-16		
10^9	0.000134164078650	4.6386e-16		

Table 6: Values of $\bar{F}_n(x)$ for $x = \sqrt{18/n}$.

n	x	Miller	Mathematica	Rel. err.	Abs. err.
20	0.44721359549996	0.000362739697817368	0.000362739697817367	3.0e-15	1.1e-18
40	0.31622776601684	0.000469148796139823	0.000469148796139491	7.1e-13	3.3e-16
60	0.25819888974716	0.000513418298233135	0.000513418298231541	3.1e-12	1.6e-15
80	0.22360679774998	0.000538602147624629	0.000538602147621453	5.9e-12	3.2e-15
100	0.2	0.000555192732807431	0.000555192732802810	8.3e-12	4.6e-15
120	0.18257418583506	0.000567103285090544	0.000567103285084519	1.1e-11	6.0e-15
140	0.16903085094570	0.000576152104012519	0.000576152104005186	1.2e-11	7.3e-15

Table 7: Values of $\bar{F}_n(x)$ for $x = \sqrt{4/n}$.

n	x	Miller	Mathematica	Rel. err.
141	0.124911316058364	0.0223963592	0.0223963330223726	1.2e-6
300	0.0856348838577675	0.0230987058	0.0230986730185827	1.4e-6
500	0.066332495807108	0.0234361007	0.0234360648085745	1.5e-6
1000	0.0469041575982343	0.0237703789	0.0237703399363784	1.6e-6
5000	0.020976176963403	0.0242079719	0.0242079291326927	1.8e-6

Table 8: Values of $\bar{F}_n(x)$ for $x = \sqrt{2.2/n}$.

n	x	Miller	Durbin matrix	Rel. err.
500	0.066332495807108	0.0234361007	0.0234360648085807	1.5e-6
1000	0.046904157598234	0.0237703789	0.0237703399363874	1.6e-6
5000	0.020976176963403	0.0242079719	0.0242079291327157	1.8e-6
10000	0.0148323969741913	0.0243102062	0.0243101626961063	1.8e-6
50000	0.0066332495807108	0.0244457597	0.0244457151043362	1.8e-6
100000	0.0046904157598234	0.0244777310	0.0244776861027715	1.8e-6

Table 9: Values of $\bar{F}_n(x)$ for $x = \sqrt{2.2/n}$.

Region $n > 10^5$. Here we compare the Pelz-Good approximation for $n = 100001$ with the Durbin matrix C program for a few values of x. The results are shown in Table 12. The Pelz-Good approximation is not so good for small x but becomes better as x increases. It also becomes better as n increases.

n	x	Pelz-Good	Mathematica	Rel. err.
140	0.0464158883361278	0.09025921823	0.0902623294750042	3.4e-5
500	0.0198657677675854	0.01302426466	0.0130242540021059	8.2e-7
1000	0.0125146494913519	0.00289496816	0.00289493725169814	1.1e-5
5000	0.0042799499222603	1.42356151e-5	1.42355083146456e-5	7.5e-6

Table 10: Values of $F_n(x)$ for $x = (1.4/n)^{2/3}$.

n	x	Pelz-Good	Durbin	Rel. err.
140	0.0464158883361278	0.09025921823	0.0902623294750041	3.4e-5
500	0.0198657677675854	0.01302426466	0.0130242540021058	8.2e-7
1000	0.0125146494913519	0.00289496816	0.00289493725169812	1.1e-5
5000	0.00427994992226032	1.42356151e-5	1.42355083146454e-5	7.5e-6
10000	0.00269619949977585	4.83345438e-7	4.83345410767114e-7	5.6e-8
50000	0.00092208725841169	3.71471703e-12	3.71479094405454e-12	2.0e-5
100000	0.00058087857335637	2.21229903e-15	2.21236052547566e-15	2.8e-5

Table 11: Values of $F_n(x)$ for $x = (1.4/n)^{2/3}$.

x	x	Pelz-Good	Durbin matrix	Rel. err.
$1/14\sqrt{n}$	0.000225875846349904	6.09086278e-103	1.07874093328718e-102	0.44
$1/12\sqrt{n}$	0.000263521820741555	1.572209174e-75	1.87885894249649e-75	0.16
$1/10\sqrt{n}$	0.000316226184889866	2.269812367e-52	2.35008915128113e-52	0.034
$1/8\sqrt{n}$	0.000395282731112333	1.962478061e-33	1.96902657319316e-33	0.0033
$1/6\sqrt{n}$	0.00052704364148311	1.018350563e-18	1.01845452774208e-18	0.0001
$1/4\sqrt{n}$	0.000790565462224666	2.9070737934e-8	2.90707424915525e-8	1.6e-7
$1/2\sqrt{n}$	0.00158113092444933	0.0363919975970	0.0363919976016742	1.3e-10
$1/\sqrt{n}$	0.00316226184889866	0.7305646847185	0.730564684714965	4.8e-12
$2/\sqrt{n}$	0.00632452369779733	0.9993319333086	0.999331933307205	1.4e-12

Table 12: Values of $F_n(x)$ for $n = 100001$.

Affiliation:

Richard Simard, Pierre L'Ecuyer
Département d'Informatique et de Recherche Opérationnelle
Université de Montréal
C.P. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, Canada
E-mail: lecuyer@iro.umontreal.ca, simardr@iro.umontreal.ca
URL: http://www.iro.umontreal.ca/~lecuyer/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 39, Issue 11 *Submitted:* 2010-03-17
March 2011 *Accepted:* 2010-12-13