Evaluating Kolmogorov’s Distribution

George Marsaglia∗
The Florida State University
Wai Wan Tsang†
Jingbo Wang
The University of Hong Kong

Abstract

Kolmogorov’s goodness-of-fit measure, $D_n$, for a sample CDF has consistently been set aside for methods such as the $D^+_n$ or $D^-_n$ of Smirnov, primarily, it seems, because of the difficulty of computing the distribution of $D_n$. As far as we know, no easy way to compute that distribution has ever been provided in the 70+ years since Kolmogorov’s fundamental paper. We provide one here, a C procedure that provides $\Pr(D_n < d)$ with 13-15 digit accuracy for $n$ ranging from 2 to at least 16000. We assess the (rather slow) approach to limiting form, and because computing time can become excessive for probabilities $> .999$ with $n$’s of several thousand, we provide a quick approximation that gives accuracy to the 7th digit for such cases.

1 Introduction

For an ordered set $x_1 < \cdots < x_n$ of purported uniform [0,1) variates, Kolmogorov [5] suggested

$$D_n = \max(x_1 - \frac{0}{n}, x_2 - \frac{1}{n}, \ldots, x_n - \frac{n-1}{n}, \frac{1}{n} - x_1, \frac{2}{n} - x_2, \ldots, \frac{n}{n} - x_n)$$

as a goodness-of-fit measure. The distribution of $D_n$ is difficult. It has been discussed extensively in the literature, but to date no easily-applied method has been made available. We offer one here. The alternatives proposed by Smirnov, either $D^+_n$, the maximum of the first half of the above list, or $D^-_n$, the maximum of the second half, have a common, easier, distribution. They are widely used, particularly in statistical computing, because of Knuth’s recommended use of $K^+_n = \sqrt{n}D^+_n$ and $K^-_n = \sqrt{n}D^-_n$ on the grounds that they “seem most convenient for computer use”,[4] p57.

Concerning the distribution of $D_n$, Drew, Glen and Leemis report in a recent article that after an extensive review, “There appears to be no source that produces exact distribution functions for any distribution where $n > 3$ in the literature”,[2] p3. They then undertake to provide such by extending Birnbaum’s development [1] of $\Pr(D_n < d)$ as a spline function: polynomials of degree $n$ between knots at $\frac{1}{2n}, \frac{2}{2n}, \ldots, 1$, using multiple integrals. They succeed in reducing the required successive integrations of Birnbaum’s method—for example from 44540 to 800 when $n = 10$—and provide the polynomials to $n = 6$ with a comment that they had found all such polynomials up to $n = 30$, available on request at www.math.vmi.edu/~leemis. (Our request yielded “Access not authorized” and an email request went unanswered.)

We provide here a relatively small C procedure, K(n,d), that will provide $\Pr(D_n < d)$ with far greater precision than is needed in practice. The method expresses $d$ in the form $d = (k - h)/n$ with $k$ a positive integer and $0 \leq h < 1$. The C procedure K(n,d) uses numerical values for $h$, but with just the symbol $h$, one can, for example in Maple or Mathematica, easily derive polynomials in $h$ that, with the substitution $h = k - nd$, yield the polynomials that make up the CDF between knots $\frac{1}{2n}, \frac{2}{2n}, \ldots, 1$.

2 Evaluating $\Pr(D_n < d)$

The method we use is based on a succession of developments that started with Kolmogorov’s viewing the steps of the sample CDF as a Poisson process and culminated in the masterful treatment by Durbin [3]. His monograph summarizes and extends the results of numerous authors who had made progress on the problem in the years 1933-73. The result is a method that expresses the required probability as a certain element in the $n$th power of an easily formed matrix. History of the development is available through the monograph’s 136 references.

∗Professor Emeritus
†Research supported by the Hong Kong Research Grants Council, Grant HKU 7046/02E.
We want to evaluate $\Pr(D_n < d)$. Write

$$d = \frac{k - h}{n} \text{ with } k \text{ a positive integer and } 0 \leq h < 1.$$ 

Then

$$\Pr(D_n \leq d) = \frac{n!}{n^m} t_{kk}, \text{ where } t_{kk} \text{ is the } k,k \text{ element of the matrix } T = H^n,$n

and $H$ is an $m \times m$ matrix, $m = 2k - 1$, whose general form is easily inferred from this particular case when $m = 6$ and $h \leq 1/2$:

$$H = \begin{bmatrix}
(1 - h^1)/1! & 1 & 0 & 0 & 0 & 0 \\
(1 - h^2)/2! & 1/1! & 1 & 0 & 0 & 0 \\
(1 - h^3)/3! & 1/2! & 1/1! & 1 & 0 & 0 \\
(1 - h^4)/4! & 1/3! & 1/2! & 1/1! & 1 & 0 \\
(1 - h^5)/5! & 1/4! & 1/3! & 1/2! & 1/1! & 1 \\
(1 - 2h^6)/6! & (1 - h^5)/5! & (1 - h^4)/4! & (1 - h^3)/3! & (1 - h^2)/2! & (1 - h^1)/1! \\
\end{bmatrix}.$$

The above example is for $0 \leq h \leq 1/2$. For $1/2 < h < 1$ the bottom left element of the matrix should be $(1 - 2h^m + (2h - 1)^m)/m!$, so that $(1 - 2h^m + \max(0, (2h - 1)^m))/m!$ is the general form of that corner element. The bottom row of the matrix reflects the first column in reverse order. Aside from the first column and last row, the $i,j$th element is $1/(i-j+1)!$ if $i-j+1 \geq 0$, else 0.

Example: Suppose $n = 10$ and we want $\Pr(D_{10} \leq .274)$. Express $d = .274$ as .274 = $\frac{3\cdot h}{10}$, so that $k = 3$, $m = 2k - 1 = 5$ and $h = .36$. Our $5 \times 5$ matrix $H$ is

$$H = \begin{bmatrix}
(1 - h) & 1 & 0 & 0 & 0 \\
(1 - h^2)/2 & 1/1! & 1 & 0 & 0 \\
(1 - h^3)/6 & 1/2! & 1/1! & 1 & 0 \\
(1 - h^4)/24 & 1/6 & 1/2! & 1/1! & 1 \\
(1 - 2h^5)/120 & (1 - h^4)/24 & (1 - h^3)/6 & (1 - h^2)/2 & (1 - h) \\
\end{bmatrix}.$$

If we express $h = .36$ as a floating point number, then the $3,3$ element of $\frac{10^5}{m^m} H^{10}$ yields, (using the C proc below):

$$\Pr(D_{10} \leq .274) = .6284796154565043$$

On the otherhand, expressing $h = \frac{274}{1000}$ as a rational, and assuming we have rational arithmetic, the $3,3$ element of $\frac{10^5}{m^m} H^{10}$ yields

$$\Pr(D_{10} \leq \frac{274}{1000}) = \frac{59936486764574456275603}{9536743164062500000000} = .628479615456504275298526691328 \cdots,$

confirming the accuracy of the floating point calculation.

Finally, if we merely use the symbol $h$ and have symbolic programming such as with Maple or Mathematica, we find that the $3,3$ element of $H^{10}$ is

$$\frac{26}{225} h^{10} - \frac{34}{27} h^9 - \frac{719}{90} h^8 - \frac{88}{3} h^7 + \frac{589}{15} h^6 - \frac{1036}{225} h^5 + \frac{1055}{4} h^4 - 6653 h^3 - \frac{59678}{360} h^2 - 687251 h^1 + \frac{28947001}{720} = \frac{14400}{27}.$$ 

Substituting $3 - 10d$ for $h$, then multiplying by $10^5/10^5$ gives $\Pr(D_n < d)$ for $5/20 < d < 6/20$:

$$419328 d^{10} - 801024 d^9 + \frac{3771936}{5} d^8 - \frac{11684736}{25} d^7 + \frac{24769584}{125} d^6 - \frac{32213664}{625} d^5 + \frac{3604041}{12500} d^4 - \frac{5133231}{50000} d^3 - \frac{25247817}{2500000} d^2 - \frac{15369417}{10000000} d - \frac{100000000}{100000000}.$$ 

If you wanted, for example, such a polynomial for $4/20 < d < 5/20$, (that is, $4/20 < (k - h)/10 < 5/20$, so that $k = 3$ and $1/2 < h < 1$), you could change the lower left element of $H$ to $(1 - 2h^5 + 2(2h - 1)^5)/5!$. Then the $3,3$ element of $H^{10}$ yields

$$-\frac{2}{9} h^{10} + \frac{98}{18} h^9 + \frac{439}{36} h^8 - \frac{1076}{48} h^7 + \frac{32731}{48} h^6 - \frac{41105}{18} h^5 + \frac{10607}{18} h^4 - \frac{52255}{72} h^3 - \frac{7984}{9} h^2 + \frac{288593}{144}.$$ 

Replacing $h$ by $3 - 10d$ and multiplying by $10^5/10^5$ then yields $\Pr(D_n < d)$ for $4/20 < d < 5/20$:

$$-806400 d^{10} + 1102080 d^9 - 594720 d^8 + \frac{177408}{5} d^7 + \frac{3421908}{25} d^6 - \frac{9773694}{125} d^5 + \frac{4771019}{2500} d^4 - \frac{13212297}{6250} d^3 + \frac{1035279}{12500} d^2 + \frac{848673}{62500} d - \frac{88389}{781250}.$$ 

2
3 Limiting Forms

The limiting form for the distribution function of Kolmogorov’s \( D_n \) is

\[ \lim_{n \to \infty} \Pr(\sqrt{n}D_n \leq x) = L(x) = 1 - 2 \sum_{i=1}^{\infty} (-1)^{i-1} e^{-2i^2x^2} = \frac{\sqrt{2\pi}}{x} \sum_{i=1}^{\infty} e^{-(2i-1)^2x^2/(8x^2)}, \]

the first representation given by Kolmogorov, the second coming from a standard relation for theta functions and better suited for small \( x \). The moments come from easily-integrated terms of \( xL'(x) \) and \( x^2L'(x) \).

The mean and variance of \( \sqrt{n}D_n \) approach

\[ \mu = \sqrt{\pi/2} \ln(2) = .8687311605 \ldots \quad \text{and} \quad \sigma^2 = \pi^2/12-\mu^2 = .067732044 \ldots, \sigma = 2.603328723 \ldots, \]

Since the mean and standard deviation of \( D_n \) are, roughly, \( .8687/\sqrt{n} \) and \( .26/\sqrt{n} \), we may compare distributions and their approaches to limiting form by plotting \( \Pr(D_n \leq x/\sqrt{n}) - L(x) \) for, say, \( n = 64, 256, 1024, 4096 \), with \( x \) over an effective range for \( L(x) \), say \( .2 < x < 2.5 \), (-2.6 to 6.3 sigmas). Such plots are in Figure 1. Approach to the limit is rather slow, with maximum error of about .278/\( \sqrt{n} \) near the 33rd percentile.

![Figure 1: Error plots: \( \Pr(D_n < x/\sqrt{n}) - L(x) \) for \( n = 64, 256, 1024, 4096 \).](image)

Our development of this procedure for Kolmogorov’s \( D_n \) was motivated by requests for its inclusion in the Diehard Battery of Tests of Randomness [6], which considers KS tests a generic class including Kolmogorov’s \( D_n \), Smirnov’s \( D_n^+ \), \( D_n^- \) or the Cramer-von Mises class, particularly the Anderson-Darling

\[ A_n = -n - \frac{1}{n} \left( \ln(x_1z_1) + 3\ln(x_2z_2) + 5\ln(x_3z_3) + \cdots + (2n-1)\ln(x_nz_n) \right) \] with \( z_i = 1 - x_{n+1-i} \).

That \( A_n \) is the current favorite for Diehard, but new versions will include both \( A_n \) and \( D_n \).

In practice (at least in our practice), we have a randomly produced \( D_n \) which we wish to convert to a uniform (0,1) variate \((p\text{-value})\) by means of the probability transformation \( p = K(n, D_n) \). The C procedure below lets us do this very accurately, as well as quickly—except for \( p \)'s near 1 and \( n \)'s several thousand.

In the following examples, we cite values and timings from the C proc below, as well as (20-digit) accuracies provided by a much slower Maple proc. For the C proc, \( K(2000, .04) = .999676094319171325 \) (99.6760943191713670985) takes about 1 second, \( K(2000, .06) = .9999998935692991 \) (99.999989356930568118) takes 4-5 seconds, but \( K(16000, .016) = .999452491380971 \) (99.945249138052085) takes around 100 seconds, and for \( n > 4000 \), getting probabilities such as .999999 can take many minutes.

If \( K(n, D_n) \) is used in the Diehard tests, we might encounter some bad RNGs that return values up to 10 \( \sigma \)'s from the mean, for which conversion to a p-value by means of \( K(n, D_n) \) might require minutes. For that reason, we include an optional line in the C program:

```c
if (a>=7.2411 & (a>=3.76&&a>=99)) return 1.2*exp(- (2.000071+.331/sqrt(n)+1.409/n)*a);
```

(As \( d\sqrt{n} \) exceeds about 1.94, \( K(n, d) \) will exceed .999 and is approximately \( 1 - 2e^{-2nd^2} \), which can be improved to \( 1 - 2e^{- ((2.000071+.331/sqrt(n)+1.409/n))d^2} \), with maximum error less than .0000005.)

Use of that line provides more than adequate accuracy for \( K(n, d) > .999 \) and \( n \geq 100 \), (roughly \( d\sqrt{n} > 1.94 \)), as well as protection from possible long computing time for any \( n \) when \( K(n, d) > .999999 \), (roughly, \( d\sqrt{n} > 2.69 \)). That extra line can be commented out for users who need the full 13-15 digit accuracy at the extreme right (and are willing to contend with potentially long running times). The extreme left causes no problems.

In computing \( H^n \), the required number of matrix multiplications is only \( \lfloor \log_2(n) \rfloor \) plus the number of 1’s in the binary representation of \( n \). A straightforward implementation encounters floating point exponent
overflow around $n = 714$. Detailed inspection shows that the elements of $H^n$ grow quickly as $n$ increases. Their magnitudes are not too diversified though, with largest values around the center of the matrix. To maintain floating point exponents within their allowable range, we keep a special matrix exponent. When the $k,k$ element of a current matrix becomes greater than $10^{140}$, we divide every element by $10^{140}$ and increase the matrix exponent by 140. The final matrix exponent is used to adjust the value of $\sum_{k,k} H^n_{k,k}$, where $T = H^n$.

The following C program contains the procedure $K(n,d)$, as well as supporting procedures for multiplying and exponentiating matrices. It is in compact form to save space. To use $K(n,d)$ you need only add a main program to a cut-and-paste version of the code listed below. Then make calls to $K(n,d)$ from an int main(){ }.

You should also lead with the usual #include <stdio.h>, #include <math.h> and #include <stdlib.h>.

4 The C program for $K(n,d) = \Pr(D_n < d)$

```c
void mMultiply(double *A, double *B, double *C, int m)
{ int i, j, k; double s;
  for (i=0; i<m; i++)
    for (j=0; j<m; j++)
      {s=0.; for (k=0; k<m; k++) s+=A[i*m+k]*B[k*m+j]; C[i*m+j]=s;}
}

void mPower(double *A, int eA, double *V, int *eV, int m, int n)
{ double *B; int eB, i;
  if (n==1) {for (i=0; i<m*m; i++) V[i]=A[i]; *eV=eA; return;}
  mPower(A, eA, V, eV, m, n/2);
  B=(double*)malloc((m*m)*sizeof(double));
  mMultiply(V, V, B, m);
  eB=2*(*eV);
  if (n%2==0) {for (i=0; i<m*m; i++) V[i]=B[i]; *eV=eB;}
  else {mMultiply(A, B, V, m); *eV=eA+eB;}
  if (V[(m/2)*m+(m/2)]>1e140) {for (i=0; i<m*m; i++) V[i]=V[i]*1e-140; *eV-=140;}
  free(B);
}

double K(int n, double d)
{ int k, m, i, j, g, eH, eQ;
  double h, s, *H, *Q;
  // Omit next line if you require >7 digit accuracy in the right tail
  s=d*d*n; if (s>7.24||((s>3.76&&n>99))) return 1-2*exp(-(2.000071+.331/sqrt(n)+1.409/n)*s);
  k=(int)(n*d)+1; m=2*k-1; h=k-n*d;
  H=(double*)malloc((m*m)*sizeof(double));
  Q=(double*)malloc((m*m)*sizeof(double));
  for (i=0; i<m; i++)
    for (j=0; j<m; j++)
      if (i-j+1<0) H[i*m+j]=0; else H[i*m+j]=1;
  for (i=0; i<m; i++)
    for (j=0; j<m; j++)
      for (g=1; g<=i-j+1; g++) H[i*m+j]/=g;
  eH=0; mPower(H, eH, Q, &eQ, m, n);
  s=Q[(k-1)*m+k-1];
  for (i=1; i<=n; i++)
    {s=s*i/n; if (s<1e-140) {s*=1e140; eQ-=140;}}
  s*=pow(10., eQ); free(H); free(Q); return s;
}
```

References


