*Journal of Statistical Software*

*MMMMMM YYYY, Volume VV, Code Snippet II.* doi: 10.18637/jss.v000.i00

# Replication Material on: Computing the Kolmogorov-Smirnov Distribution when the Underlying Cdf is Purely Discrete, Mixed or Continuous

**Dimitrina S. Dimitrova**
City, University of London

**Vladimir K. Kaishev**
City, University of London

**Senren Tan**
City, University of London

**Abstract**

This supplement contains replication material related to reproducing numerical results obtained using the Exact-KS-FFT method, the asymptotic formulae in Section 2.2 of the paper, the method of Wood and Altavela (1978), the method of Arnold and Emerson (2011), the method of Simard and L'Ecuyer (2011), and the method of Carvalho (2015).

*Keywords*: Kolmogorov-Smirnov test statistic, discontinuous (discrete or mixed) distribution, Fast Fourier Transform, double boundary non-crossing, rectangle probability for uniform order statistics.

## 1. Results obtained using the Exact-KS-FFT method

### 1.1. When $F(x)$ is discontinuous

The approach to computing the exact $\mathsf{P}(D_n \geq q)$ when $F(x)$ is discontinuous is outlined in the following procedure (also given in the paper of Dimitrova, Kaishev, Tan 2017, see Section 2.1 and references to the equations therein).

**Procedure Exact-KS-FFT**

(i) Specify a discontinuous cdf $F(x)$, a sample size $n$, and a quantile $q$.

(ii) As detailed in Step 1, compute $A_i$ and $B_i$ for $i = 1, ..., n$, based on (4), where the *limites* are coded using a very small $\epsilon$, e.g., $\epsilon = 10^{-10}$.

(iii) As detailed in Step 2, compute the upper and lower boundaries $g(t)$, $h(t)$ using (6).

(iv) Following Steps 3 and 4, apply FFT to compute $Q(1, n)$ defined in (10). Hence, calculate the double-boundary non-crossing probability with respect to the PP on the right-hand-side of (8) and respectively obtain the double-boundary non-crossing probability with respect to $\eta_n(t)$ on the left-hand-side of (8).

(v) Finally, compute the exact $\mathsf{P}(D_n \geq q)$ using (5) (cf., Steps 2 and 3).

*R implementation*

In order to compute $\mathsf{P}(D_n \geq q)$, when $F(x)$ is purely discrete using the R package **KSgeneral**, one needs to input `disc_ks_c_cdf(q, n, y, ..., exact = NULL, tol = 1e - 08, sim.size = 1e + 06, num.sim = 10)`, where `y` specifies the purely discrete cdf $F(x)$, possibly followed by a list of parameters `...` specifying $F(x)$, the input parameter `exact` is a logical variable specifying whether one wants to compute exact values for $\mathsf{P}(D_n \geq q)$ using the FFT-based method, i.e., `exact = TRUE` or wants to compute the approximate values for $\mathsf{P}(D_n \geq q)$ using the simulation-based algorithm of Wood and Altavela (1978), in which case `exact = FALSE`. When `exact = NULL` and `n <= 100000` by default, the exact $\mathsf{P}(D_n \geq q)$ will be computed using the Exact-KS-FFT method. The input parameter `tol` is the value of $\epsilon$ that is used to compute the values $A_i$ and $B_i$, $i = 1, ..., n$, as detailed in Step 1 of Section 2.1 in Dimitrova, Kaishev, Tan (2017) (see also (ii) of the **Procedure Exact-KS-FFT**). By default, `tol = 1e - 08`. Note that a value of `NA` or `0` will lead to an error. The input parameter `sim.size` is the required number of simulated trajectories in order to produce one Monte Carlo estimate (one MC run) of the asymptotic $p$ value using the algorithm of Wood and Altavela (1978). By default, `sim.size = 1e + 06`. The input parameter `num.sim` is the number of MC runs, each producing one estimate (based on `sim.size` number of trajectories), which are then averaged in order to produce the final estimate for the asymptotic $p$ value. This is done in order to reduce the variance of the final estimate. By default, `num.sim = 10`. For instance, if one wants to use the R package **KSgeneral** to compute the exact value for $\mathsf{P}(D_n \geq q)$, when $F(x)$ follows a $Binomial(3, 0.5)$ distribution as in Example 3.4, with $n = 400$, $q = 0.05$, one should run the following R code and obtain the corresponding result, as shown in the column Exact-KS-FFT of Table 3.

```
R> binom_3 <- stepfun(c(0 : 3), c(0, pbinom(0 : 3, 3, 0.5)))
R> disc_ks_c_cdf(0.05, 400, binom_3)

[1] 0.05611849
```

*C++ implementation*

For example, to obtain the probability $\mathsf{P}(D_n \geq q)$, for $n = 25$, $q = 0.20$ as shown in the column Exact-KS-FFT of Table 3, according to step (i) of the **Procedure Exact-KS-FFT**, we first define the cdf of a $Binomial(3, 0.5)$ distribution in the file "*crossprob.cc*" using the following code.

```
vector <double> TheoreticalDF (vector <double> pmf)
{
```

```
  double cumulative_sum;
  vector <double> TDF(pmf.size());

  for(int i = 0; i < pmf.size(); ++i){

        cumulative_sum += pmf[i];
        TDF[i] = cumulative_sum;

  }
  return TDF;
}

vector <double> BinomialPF(int trial, double prob)
{
  int num = trial + 1;
  vector <double> PF(num);
  for (int i = 0; i < num; ++i){
        PF[i] = exp(lgamma(num) - lgamma(num - i) - lgamma(i + 1)) *
             pow(prob, i) * pow(1 - prob, trial - i);
  }

  vector <double>::iterator it;
  it = PF.begin();
  PF.insert(it, 0.0);
  return PF;
}

vector <double> BinomialDF(int trial, double prob)
{
  vector <double> Bin_CDF = TheoreticalDF(BinomialPF(trial, prob));
  Bin_CDF[Bin_CDF.size()-1]= 1.0;
  return Bin_CDF;
}

vector <double> MixDF (vector <double> obs)
{
  vector <double> observed = obs;
  set<double> s;
  for (int i = 0; i < obs.size(); ++i){
        s.insert(obs[i]);
  }
  obs.assign(s.begin(), s.end());
  vector <double> DF(obs.size());
  /* The Binomial (3, 0.5) distribution in Table 3 */

  vector <double> Binom_pmf = BinomialDF(3, 0.5);
  for (int i = 0; i < obs.size(); ++i){
```

```
        if (obs[i] < 0.0){
            DF[i] = 0.0;
        }
        else if (obs[i] < 1.0){
            DF[i] = Binom_pmf[1];
        }
        else if (obs[i] < 2.0){
            DF[i] = Binom_pmf[2];
        }
        else if (obs[i] < 3.0){
            DF[i] = Binom_pmf[3];
        }
        else
        {
            DF[i] = 1.0;
        }
  }
  return DF;
}
```

Also, since the cdf of a $Binomial(3, 0.5)$ distribution has jumps at $0, 1, 2, 3$, we need to specify this by inputting `vector_input3 = {0.0, 1.0, 2.0, 3.0};` to the `int main()` function in the file "*crossprob.cc*".

Next, we first run `make` in one of the command line tools (e.g., bash) to build the program for the Exact-KS-FFT method, developed by Dimitrova, Kaishev, Tan (2017) and based on the code provided by Moscovich and Nadler (2017). Then, in the command line tool, we run the following line `./bin/crossprob ecdf 25 Boundary_Crossing_Time.txt`, where 25 is the input for the sample size. We will have the following screen prompts.

```
Please enter the distribution type: 1 for Continuous Distribution,
2 for Discontinuous Distributions:
```

We enter 2 since the cdf of a $Binomial(3, 0.5)$ distribution is not continuous.

```
2
```

Then, we can choose whether to calculate the K-S complementary cdf, $\mathsf{P}(D_n \geq q)$, or the $p$ value, $\mathsf{P}(D_n \geq d_n)$, corresponding to a value $d_n$ computed based on a user provided data sample.

```
Please enter the objective: 1 for K-S Complementary Distribution,
2 for P-Values:
```

Since we want to obtain the probability $\mathsf{P}(D_n \geq q)$, for $n = 25$, $q = 0.20$, we will enter 1.

```
1
```

Here, we enter the sample size $n$ and the quantile $q$.

```
Please enter the sample size and quantile:

25
0.20

Probability: 0.0468500212132494
Time taken: 0.0000740000000000
```

Now, steps (ii), (iii), (iv) and (v) of the **Procedure Exact-KS-FFT** are performed. The result for $\mathsf{P}(D_n \geq q)$, for $n = 25$, $q = 0.20$, is 0.046850021 as in the column Exact-KS-FFT of Table 3. The corresponding computation time is also printed.

Following the similar procedures as above, we can obtain other values for $\mathsf{P}(D_n \geq q)$ in: 1) the Exact-KS-FFT column of Table 1; 2) the Exact-KS-FFT column of Table 2; 3) the Exact-KS-FFT column of Table 3; 4) the Exact-KS-FFT column of Table 4; 5) the Exact-KS-FFT column of Table 5; 6) the Exact-KS-FFT column of Table 6.

**Remark 1.1.** Note that the distribution of the K-S test statistic $D_n$ depends on the hypothesized distribution $F(x)$ when $F(x)$ is not continuous. Hence, to obtain $\mathsf{P}(D_n \geq q)$ for different discontinuous $F(x)$, the users should: 1) define the mixed cdf in the file "*crossprob.cc*" each time, and 2) in the file "*crossprob.cc*", define the vector containing points where $F(x)$ has jumps, `vector_input3`.

## 1.2. When $F(x)$ is continuous

*R implementation*

In order to compute $\mathsf{P}(D_n \geq q)$, when $F(x)$ is continuous using the R package **KSgeneral**, one needs to input `cont_ks_c_cdf(q, n)`. For instance, in order to compute the value for $\mathsf{P}(D_n \geq q)$, for $n = 141$, $nq^2 = 2.1$, one should run the following R code and obtain the corresponding result as shown in Table 19 for $n = 141$ in the column Exact-KS-FFT.

```
R> cont_ks_c_cdf(sqrt(2.1/141), 141)

[1] 0.02743689
```

*C++ implementation*

Note that the distribution of $D_n$ is distribution-free, when $F(x)$ is continuous (cf., Equation 25). Hence, we can directly run the the following

```
./bin/crossprob ecdf n Boundary_Crossing_Time.txt
```

where $n$ is the input for the sample size in the command line tool. For example, if we want to obtain the probability $\mathsf{P}(D_n \geq q)$ for $nq^2 = 2.1$ and $n = 141$ as shown in the column Exact-KS-FFT of Table 19 in Appendix C, we run in the command line tool `./bin/crossprob ecdf 141 Boundary_Crossing_Time.txt`. We will have the following screen prompts.

```
Please enter the distribution type: 1 for Continuous Distribution,
2 for Discontinuous Distributions:
```

We enter 1 since the cdf $F(x)$ is continuous.

*1*

Here, we enter the sample size $n$.

```
Please enter the sample size:
```

*141*

Then, we can choose whether to calculate the K-S complementary cdf, $\mathsf{P}(D_n \geq q)$, or the $p$ value, $\mathsf{P}(D_n \geq d_n)$, corresponding to a value $d_n$ computed based on a user provided data sample.

```
Please enter the objective: 1 for K-S Complementary Distribution,
2 for P-Values:
```

Since we want to obtain the probability $\mathsf{P}(D_n \geq q)$, for $n = 141$, $q = 0.1220394077$, we will enter 1.

*1*

Here, we enter the quantile $q$.

```
Please enter the quantile:
```

*0.1220394077*

```
Probability: 0.0274368890595805
Time taken: 0.0003110000000000
```

Hence, the result for $\mathsf{P}(D_n \geq q)$, for $n = 141$, $nq^2 = 2.1$, is 0.02743688914 as in the column Exact-KS-FFT of Table 19.

Following the similar procedures as above, we can obtain other values for $\mathsf{P}(D_n \geq q)$ and $\mathsf{P}(D_n < q) = 1 - \mathsf{P}(D_n \geq q)$ in the column Exact-KS-FFT of Tables 2, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21 and 22.

### 1.3. Speed comparison

To obtain the CPU time to compute $\mathsf{P}(D_n \geq q)$ 100 times with the Exact-KS-FFT method, as shown in Table 24, we modify part of the *"crossprob.cc"* file. More precisely, the original code is

```
else if (command == "ecdf") {
  verify_boundary_is_valid(lower_bound_steps);
  verify_boundary_is_valid(upper_bound_steps);
  cout
<< 1.0 - ecdf_noncrossing_probability(n, lower_bound_steps, upper_bound_steps,
      use_fft)
<< endl;
}
```

and using a `for` loop, the above code has been implemented to compute the same probability $\mathsf{P}(D_n \geq q)$ 100 times.

```
else if (command == "ecdf") {
  verify_boundary_is_valid(lower_bound_steps);
  verify_boundary_is_valid(upper_bound_steps);
  int repetition = 100;
  for (int i = 0; i < repetition; ++i){
   1.0 - ecdf_noncrossing_probability(n, lower_bound_steps, upper_bound_steps,
      use_fft);
  }
}
```

We build the program again; and following the **Procedure Exact-KS-FFT**, we can obtain the CPU time to compute $\mathsf{P}(D_n \geq q)$ 100 times with the Exact-KS-FFT method, as shown in Table 24.

**Remark 1.2.** In above sections 1.1, 1.2, 1.3, we have briefly introduced how we could replicate one value in a specific table. We have also provided source files to replicate each table instead of just one specific value of the table.

If one wants to replicate the tables using R, one can use the file *"Tables_Replication.r"* provided in the replication material under the folder named *"Exact_KS_FFT_Replication_R"*. First of all, the R package **KSgeneral** should be installed. Then, the file *"Tables_Replication.r"* should be sourced in R. For example, if the file *"Tables_Replication.r"* is stored in the current working directory, then one can input in R the following command.

```
R> source('Tables_Replication.r')
```

If one wants to replicate Table X in Dimitrova, Kaishev, Tan (2017), one needs to run the following code in R

```
R> Tables(X),
```

where possible inputs of `X` are `1`, `2`, `3`, `4`, `5`, `6`, `8`, `9`, `10`, `11`, `12`, `13`, `14`, `15`, `16`, `17`, `18`, `19`, `20`, `21`, `22`.

If one wants to replicate the tables in C++, first of all, in the command line tool, one should set the current directory to where the folder *"Exact_KS_FFT_Replication_C++"* is located. Then, in the command line tool, one should run `make` to build the executable *"crossprob"*. Finally, one should run `./bin/crossprob TableXX Boundary_Crossing_Time.txt` in the command

line tool, where possible inputs of `TableXX` are `Table1`, `Table2`, `Table3`, `Table4`, `Table5`, `Table6`, `Table8_13`, `Table14_16`, `Table17`, `Table18`, `Table19_21`, `Table22`, or `Table24`.

The values of the (complementary)cdf and the corresponding computation times will be printed in the file *"Table_XX.txt"* located in the folder *"Exact_KS_FFT_Replication_C++"*.

## 2. Results obtained using the asymptotic formula (15)

As noted in Example 3.1 of the manuscript, $F_Y(y)$ has two jumps, (i.e., $J = 2$) at $x_1 = 0, x_2 = \log 2.5$, and $f_0 = f_1 = 0, f_2 = 0.5, f_3 = 0.8, f_4 = f_5 = 1$. Since the jump structure of $F_Y(\cdot)$ in Example 3.1 is flat-jump, increasing-jump segments, the first set of increasing-jump segments and the last set of flat-jump segments in (14) should be omitted. Therefore, one should apply formula (15) with $m = 2, \nu_1 = 0, \omega_1 = 1, \nu_2 = 1, \omega_2 = 0$, and $v_0 = 0, v_1 = 0, v_2 = 1, w_0 = 0, w_1 = 1, w_2 = 1$. We have implemented the asymptotic formula (15) in Mathematica 10 (Wolfram Research Inc. 2015). For example, if we want to obtain the asymptotic probability $\mathsf{P}(D_n \geq q)$ for $\lambda = 1$, the code is as following.

```
f[l_] := (-1) ^ (l) * 1 / (2 * Pi) *
  ((f2 - f1) * (f3 - f2)*  (f4 - f3)) ^ (- 1 / 2);
f1 = 0;
f2 = 1 / 2;
f3 = 4 / 5;
f4 = 1;
lambda = 1;
g[x1_, x2_, l_] :=
  Exp[-1 / 2 * (x1 ^ (2) / (f2 - f1) + x2 ^ (2) / (f4 - f3)) -
   1/2 * (x2 - (-1) ^ (l) * x1 - 2 * lambda * l) ^ (2) / (f3 - f2)];
Timing[For [sum = 0; i = - 20, i <= 20, i++,
  sum += NIntegrate[
    g[x1, x2, i], {x1, -lambda, lambda}, {x2, -lambda, lambda}] *
    f[i]] ; 1 - sum]
```

The output will be

```
{0.73, 0.174525238}
```

The first element, 0.73, shows the CPU time to compute asymptotic $\mathsf{P}(D_n \geq q)$ with Formula (15), whereas the second element, 0.174525238, refers to the asymptotic probability as shown in the column Asympt. (15) of Table 1. Similarly, we can obtain asymptotic probabilities $\mathsf{P}(D_n \geq q)$ for $\lambda = 3$, 2, 0.5, 0.2, 0.15, as well as the CPU time to compute them, as shown in the column Asympt. (15) of Table 1, by modifying the line `lambda = 1` to the corresponding values of $\lambda$.

**Remark 2.1.** When $\lambda = 3$, the probability $\mathsf{P}(D_n \geq q)$ is close to zero. Hence, in Mathematica 10, we may need to increase the working precision and the precision goal, for example, by adding `WorkingPrecision -> 30, PrecisionGoal -> 20` into the function `NIntegrate[]`. However, the computation time will be increased as a trade-off between the efficiency and accuracy.

# 3. Results obtained using the asymptotic formula (22)

## 3.1. When $F(x)$ follows a $Binomial(3, 0.5)$ distribution

Note that when $F(x)$ follows a $Binomial(3, 0.5)$ distribution, there are four jumps in $F(x)$ (i.e., $J = 4$). Since $F(x)$ is purely discrete, one should apply formula (22). We have implemented the asymptotic formula (22) in Mathematica 10. For example, if we want to obtain the asymptotic probability $\mathsf{P}(D_n \geq q)$ for $\lambda = 1$ as shown in the column Asympt. (22) of Table 3, the code is as following.

```
a1 = (3!) / (0! * (3 - 0)!) * (1 / 2) ^ 3;
a2 = (3!) / (1! * (3 - 1)!) * (1 / 2) ^ 3;
a3 = (3!) / (2! * (3 - 2)!) * (1 / 2) ^ 3;
a4 = (3!) / (3! * (3 - 3)!) * (1 / 2) ^ 3;
lambda = 1;
c = (2 * Pi) ^ ((1 - 4) / 2) * (a1 * a2 * a3 * a4) ^ (-1/2);
f[x1_, x2_, x3_] :=
  Exp[-0.5 * (x1 ^ 2 / a1 + (x2 - x1) ^ 2 / a2 + (x3 - x2) ^ 2 / a3 + x3 ^ 2 / a4)];
Timing[1 -
  N[c * Integrate[
    f[x1, x2, x3], {x1, -lambda, lambda}, {x2, -lambda,
    lambda}, {x3, -lambda, lambda}], 16]]
```

The output will be

```
{5.30, 0.049438582298194}
```

The first element, 5.30, shows the CPU time to compute asymptotic $\mathsf{P}(D_n \geq q)$ with formula (22), whereas the second element, 0.049438582298194, refers to the asymptotic probability as shown in the column Asympt. (22) of Table 3 when $\lambda = 1$. Similarly, we can obtain asymptotic probabilities $\mathsf{P}(D_n \geq q)$ for $\lambda = 3$, 2, 0.5, 0.2, 0.1, as well as the CPU time to compute them, as shown in the column Asympt. (22) of Table 3, by modifying the line `lambda = 1` to the corresponding values of $\lambda$.

## 3.2. When $F(x)$ follows a $Binomial(7, 0.5)$ distribution

Note that when $F(x)$ follows a $Binomial(7, 0.5)$ distribution, there are eight jumps in $F(x)$ (i.e., $J = 8$). We have implemented the asymptotic formula (22) in Mathematica 10. For example, if we want to obtain the asymptotic probability $\mathsf{P}(D_n \geq q)$ for $\lambda = 1$ as shown in the column Asympt. (22) of Table 4, the code is as following.

```
a1 = (7!) / (0! * (7 - 0)!) * (1 / 2) ^ 7;
a2 = (7!) / (1! * (7 - 1)!) * (1 / 2) ^ 7;
a3 = (7!) / (2! * (7 - 2)!) * (1 / 2) ^ 7;
a4 = (7!) / (3! * (7 - 3)!) * (1 / 2) ^ 7;
a5 = (7!) / (4! * (7 - 4)!) * (1 / 2) ^ 7;
a6 = (7!) / (5! * (7 - 5)!) * (1 / 2) ^ 7;
a7 = (7!) / (6! * (7 - 6)!) * (1 / 2) ^ 7;
```

```
a8 = (7!) / (7! * (7 - 7)!) * (1 / 2) ^ 7;
lambda = 1;
c = (2 * Pi) ^ ((1 - 8) / 2) * (a1 * a2 * a3 * a4 * a5 * a6 * a7 * a8) ^ (-1 / 2);
g[x1_, x2_, x3_, x4_, x5_, x6_, x7_] :=
  Exp[-0.5 * (x1 ^ 2 / a1 + (x2 - x1) ^ 2 / a2 + (x3 - x2) ^ 2 / a3 +
    (x4 - x3) ^ 2 / a4 + (x5 - x4) ^ 2 / a5 +
    (x6 - x5) ^ 2 / a6 + (x7 - x6) ^ 2 / a7 + x7 ^ 2 / a8)];
Timing[1 -
  c * NIntegrate[
    g[x1, x2, x3, x4, x5, x6, x7], {x1, -lambda,
    lambda}, {x2, -lambda, lambda}, {x3, -lambda,
    lambda}, {x4, -lambda, lambda}, {x5, -lambda,
    lambda}, {x6, -lambda, lambda}, {x7, -lambda, lambda},
    WorkingPrecision -> 20]]
```

The output will be

```
{473.92, 0.070168353127716}
```

The first element, 473.92, shows the CPU time to compute asymptotic $\mathsf{P}(D_n \geq q)$ with formula (22), whereas the second element, 0.070168353127716, refers to the asymptotic probability as shown in the column Asympt. (22) of Table 4 when $\lambda = 1$. Similarly, we can obtain asymptotic probabilities $\mathsf{P}(D_n \geq q)$ for $\lambda = 3$, 2, 0.5, 0.2, 0.1, as well as the CPU time to compute them, as shown in the column Asympt. (22) of Table 4, by modifying the line `lambda = 1` to the corresponding values of $\lambda$.

## 4. Results obtained using the method of Wood and Altavela (1978)

As explained in Section 3.2 of Dimitrova, Kaishev, Tan (2017), for a discrete $F(x)$ with $J$ number of jumps, one should simulate from the $(J-1)$-variate normal random vector $(Z_1, Z_2, ..., Z_{J-1})$, where

$$\mathsf{E}(Z_i) = 0, \quad \mathsf{E}(Z_i, Z_k) = \min(f_{2i}, f_{2k}) - f_{2i} f_{2k}, \quad i, k = 1, ..., J-1,$$

and estimate the probability in $\Phi(\lambda)$ in (9) as

$$\frac{\sum_{i=1}^{N} \mathbb{1}_{\{(Z_1, Z_2, ..., Z_{J-1}) \in [-\lambda, \lambda]^{J-1}\}}}{N},$$

where $N$ is the number of simulations, $\mathbb{1}_{\{\cdot\}}$ is an indicator function, and $[-\lambda, \lambda]^{J-1}$ is the $J-1$ dimensional hypercube. We have implemented Wood and Altavela (1978)'s method in the package **KSgeneral** in R.

For example, if one wants to use the simulation-based method of Wood and Altavela (1978) in order to approximate the asymptotic value for $\mathsf{P}(D_n \geq q)$, when $F(x)$ follows a $Binomial(3, 0.5)$ distribution, with $n = 400$, $q = 0.05$, one should use the W&A (a) method that provides better approximation, by running the following R code and obtain the corresponding result as shown in the column W&A(a) of Table 3.

```
R> binom_3 <- stepfun(c(0 : 3), c(0, pbinom(0 : 3, 3, 0.5)))
R> disc_ks_c_cdf(0.05, 400, binom_3, exact = FALSE, tol = 1e-08,
+ sim.size = 1e+06, num.sim = 10)

[1] 0.0561864
```

Following the same procedures above, we can obtain the simulated probability $\mathsf{P}(D_n \geq q)$ due to $W\&A(a)$ and $W\&A(b)$ methods, as shown in Tables 3, 4, 5, and 6.

## 5. Results obtained using the method of Arnold and Emerson (2011)

As described in Example 3.5, hypothesizing that the underling $F(x)$ in (1) follows a discrete uniform distribution on $[1, 10]$, we have simulated random samples of size $n$, $25 \leq n \leq 100000$. The simulated samples are stored in text files named "*discrete_uniform_n.txt*", where $n$ is the corresponding sample size. These text files have been provided with the replication material.

For example, when $n = 100$, we first set the working directory to where the file *"discrete_uniform_100.txt"* is located and then input in R the following.

```
R> data <- read.csv("discrete_uniform_100.txt", header = F)
R> dgof::ks.test(data[,1 : 100], ecdf(1 : 10), exact = T)
```

The output is

```
        One-sample Kolmogorov-Smirnov test

data:  data[, 1:100]
D = 0.2, p-value < 2.2e-16
alternative hypothesis: two-sided

Warning message:
In dgof::ks.test(data1[, 1:100], ecdf(1:10), exact = T) :
  numerical instability may affect p-value
```

We can see that when $n = 100$, the K-S test statistic for the simulated sample is $D_n = 0.2$, the $p$ value obtained from the R function `dgof::ks.test` is `p-value < 2.2e-16`, while the $p$ value obtained from the Exact-KS-FFT method is 0.00021. The function `dgof::ks.test` becomes numerically unstable, as noted also by Arnold and Emerson (2011). To avoid instability, for large $n$ the R function `dgof::ks.test` allows for estimating $p$ values via simulation, by inputting

```
R> dgof::ks.test(data[,1:100], ecdf(1:10), simulate.p.value = T, B = 2000)
```

where `B = 2000` indicates that the number of simulations is 2000. We have executed the above code a few times. The outputs are

```
One-sample Kolmogorov-Smirnov test
```

```
data:  data[, 1:100]
D = 0.2, p-value < 2.2e-16
alternative hypothesis: two-sided

One-sample Kolmogorov-Smirnov test

data:  data[, 1:100]
D = 0.2, p-value < 2.2e-16
alternative hypothesis: two-sided

One-sample Kolmogorov-Smirnov test

data:  data[, 1:100]
D = 0.2, p-value = 5e-04
alternative hypothesis: two-sided
```

Hence, we can obtain the estimated $p$ values in the column `ks.test`(simulation) of Table 6. Since the $p$ value is estimated from simulation, the value of the $p$ value is not constant, and hence may be insufficiently accurate. To improve the accuracy of the simulated $p$ value, we can increase the number of simulations used, but with a cost of lost efficiency.

## 6. Results obtained using the method of Simard and L'Ecuyer (2011)

Simard and L'Ecuyer (2011) developed a C program in file `KolmogorovSmirnovDist.c`, which can be downloaded from http://www.iro.umontreal.ca/~simardr/ksdir. There are two public functions defined in the file `KolmogorovSmirnovDist.c`: 1) `KScdf(int n, double x)`, which computes $\mathsf{P}(D_n \leq q)$ for given $n$ and $q$, and 2) `KSfbar(int n, double x)`, which computes $\mathsf{P}(D_n \geq q)$ for given $n$ and $q$.

For example, if we want to obtain the probability $\mathsf{P}(D_n \leq q)$ when $n = 20$, $nq^2 = 0.75$, as shown in the column Simard & L'Ecuyer of Table 9, we can implement the following C++ code in IDE.

```
#include <iostream>
#include <iomanip>
#include <math.h>
#include "KolmogorovSmirnovDist.h"
using namespace std;

int main()
{
  int number1;
//    double number2;
  double KS;
  double mu0;
  cout << "Enter the sample size: ";
//    cin >> number1 >> number2;
```

```
  cin >> number1;
//    mu0 = 1.0 - 1.0/number1;
//    mu0 = 1.0/number1;
  mu0 = pow(0.75/number1, 1.0/2.0);
//    mu0 = 1.0/(1.0 * sqrt(number1));
  int repetition = 100;
  clock_t tStart = clock();

  for (int i = 0; i < repetition; ++i){
//       KS = KSfbar(number1, mu0);
    KS = KScdf(number1, mu0);
  }
  cout << fixed << setprecision(15) << std::scientific
    << "The p-value of the KS test is: " << KS << endl;

  printf("Time taken: %.5fs\n", (double)(clock() - tStart)/CLOCKS_PER_SEC);
  return 0;
}
```

Then, we enter the sample size $n = 20$ from the screen prompt.

```
Enter the sample size: 20
```

The output is

```
The p-value of the KS test is: 6.089841201378628e-01
Time taken: 0.00126s
Program ended with exit code: 0
```

Hence, the output is $6.089841201378628E-01$ as shown in Table 9. Similarly, we can obtain the probability $\mathsf{P}(D_n \leq q)$ due to Simard and L'Ecuyer (2011) method, as shown in the column Simard & L'Ecuyer of Tables 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, by modifying the line `mu0 = pow(0.75/number1, 1.0/2.0);` in the above code to the corresponding $q$, and inputting the sample size $n$.

On the other hand, if we want to obtain the probability $\mathsf{P}(D_n \geq q)$, we call the function `KSfbar(int n, double x)` instead. For example, to compute $\mathsf{P}(D_n \geq q)$ when $n = 141$, $nq^2 = 2.1$, as shown in the column Simard & L'Ecuyer of Table 19, we inplement the following code.

```
#include <iostream>
#include <iomanip>
#include <math.h>
#include "KolmogorovSmirnovDist.h"
using namespace std;

int main()
{
```

```
  int number1;
//    double number2;
  double KS;
  double mu0;
  cout << "Enter the sample size: ";
//    cin >> number1 >> number2;
  cin >> number1;
    //mu0 = 1.0 - 1.0/number1;
    //mu0 = 1.0/number1;
  mu0 = pow(2.1/number1, 1.0/2.0);
    //mu0 = 1.0/(1.0 * sqrt(number1));
  int repetition = 100;
  clock_t tStart = clock();

  for (int i = 0; i < repetition; ++i){
    KS = KSfbar(number1, mu0);
//       KS = KScdf(number1, mu0);
  }
  cout << fixed << setprecision(15) << std::scientific
    << "The p-value of the KS test is: " << KS << endl;

  printf("Time taken: %.5fs\n", (double)(clock() - tStart)/CLOCKS_PER_SEC);
  return 0;
}
```

And then, we input the sample size $n = 141$.

```
Enter the sample size: 141
```

The output is

```
The p-value of the KS test is: 2.743688914193088e-02
Time taken: 0.06638s
Program ended with exit code: 0
```

Hence, the output is $2.743688914193088E-02$ as shown in Table 19. Similarly, we can obtain the probability $\mathsf{P}(D_n \geq q)$ due to Simard and L'Ecuyer (2011) method, as shown in the column Simard & L'Ecuyer of Tables 18, 19, 20, 21, 22, by modifying the line `mu0 = pow(2.1/number1, 1.0/2.0);` in the above code to the corresponding $q$, and inputting the sample size $n$.

Also, the CPU times to compute $\mathsf{P}(D_n \geq q)$ due to Simard and L'Ecuyer (2011) method as shown in Table 23 can be obtained. Recall that $\lambda = qn^{1/2}$. Hence, we need to modify the line `mu0 = pow(2.1/number1, 1.0/2.0);` in the above code to `mu0 = lambda/(1.0 * sqrt(number1));`. For example, if we want to obtain the CPU times to compute $\mathsf{P}(D_n \geq q)$ for $\lambda = 2$, we need to modify the line `mu0 = pow(2.1/number1, 1.0/2.0);` in the above code to `mu0 = 2.0/(1.0 * sqrt(number1));`. Following the same procedure as above, we can obtain the CPU times to compute $\mathsf{P}(D_n \geq q)$ due to Simard and L'Ecuyer (2011) method as shown in Table 23.

## 7. Results obtained using the method of Carvalho (2015)

Carvalho (2015) developed an R function `pkolmim(d, n)` in the package **kolmim**, where `d` is the argument for the cumulative distribution function of $D_n$, and `n` is the sample size. For example, if we want to obtain the probability $\mathsf{P}(D_n \leq q)$ when $n = 20$, $nq^2 = 0.75$, as shown in the column Carvalho of Table 9, we first install the package **kolmim** in R, and then run

```
R> pkolmim(sqrt(0.75/20), 20)
```

The output is

```
[1] 0.6089841201379
```

The value for the probability $\mathsf{P}(D_n \leq q)$ when $n = 20$, $nq^2 = 0.75$ is 0.6089841201379, as shown in the column Carvalho in Table 9. Similarly, we can obtain the probability $\mathsf{P}(D_n \leq q)$ due to Carvalho (2015) method, as shown in the column Carvalho of Tables 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, by modifying the inputs `d` and `n` in the R function `pkolmim(d, n)`.

To obtain the probability $\mathsf{P}(D_n \geq q)$ due to Carvalho (2015) method, as shown in the column Carvalho of Tables 18, 19, 20, 21, 22, we implement the code `1 - pkolmim(d, n)`, with corresponding values for `d` and `n`. For example, if we want to obtain the probability $\mathsf{P}(D_n \geq q)$ when $n = 141$, $nq^2 = 2.1$, as shown in the column Carvalho of Table 19, we implement the following code.

```
R> 1 - pkolmim(sqrt(2.1/141), 141)
```

The output is

```
[1] 0.02743688914199
```

The value for the probability $\mathsf{P}(D_n \geq q)$ when $n = 141$, $nq^2 = 2.1$ is 0.02743688914199 as shown in the column Carvalho of Table 19.

To obtain the CPU times to compute $\mathsf{P}(D_n \geq q)$ due to Carvalho (2015) method as shown in Table 25, we first implement the following R code.

```
R> carvalho <- function(n, l){
+    #repetition <- 100
+    i <- 1
+    for (i in 1 : 100){
+      kol <- 1 - pkolmim(sqrt(l/n), n)
+      i <- i + 1
+    }
+    return(kol)
+}
```

The above code defines an R function `carvalho(n, l)`, where `n` is the sample size, and `l` is the value for $\lambda$. For example, if we want to obtain the CPU times to compute $\mathsf{P}(D_n \geq q)$ when $n = 141$, $\lambda = 2$, as shown in Table 25, we implement the following R code `system.time(carvalho(141, 2))`. By substituting corresponding values for $n$ and $\lambda$ into

`carvalho(n, l)`, we can obtain the CPU times to compute $\mathsf{P}(D_n \geq q)$ due to Carvalho (2015) method as shown in Table 25.

## References

Arnold TA, Emerson JW (2011). "Nonparametric Goodness-of-Fit Tests for Discrete Null Distributions." *The R Journal*, **3**(2), 34–39.

Carvalho L (2015). "An Improved Evaluation of Kolmogorov's Distribution." *Journal of Statistical Software*, **65**(3), 1–7.

Moscovich A, Nadler B (2017). "Fast Calculation of Boundary Crossing Probabilities for Poisson Processes." *Statistics & Probability Letters*, **123**, 177–182.

Simard R, L'Ecuyer P (2011). "Computing the Two-Sided Kolmogorov-Smirnov Distribution." *Journal of Statistical Software*, **39**(11), 1–18.

Venables, W N and Ripley, B D (2002). *Modern Applied Statistics with S*. Fourth edition. Springer, New York.

Wolfram Research Inc (2015). *Mathematica*. Wolfram Research Inc., Champaign, Illinois, 10.3 edition.

Wood CL, Altavela MM (1978). "Large-Sample Results for Kolmogorov–Smirnov Statistics for Discrete Distributions." *Biometrika*, **65**(1), 235–239.

### Affiliation:

Dimitrina S. Dimitrova
Faculty of Actuarial Science and Insurance
Cass Business School
City, University of London
106 Bunhill Row, EC1Y 8TZ London, UK
E-mail: D.Dimitrova@city.ac.uk
URL: http://www.cass.city.ac.uk/experts/D.Dimitrova

Vladimir K. Kaishev
Faculty of Actuarial Science and Insurance
Cass Business School
City, University of London
106 Bunhill Row, EC1Y 8TZ London, UK
E-mail: v.kaishev@city.ac.uk
URL: www.cass.city.ac.uk/experts/V.Kaishev

Senren Tan
Faculty of Actuarial Science and Insurance
Cass Business School
City, University of London
106 Bunhill Row, EC1Y 8TZ London, UK
E-mail: Senren.Tan@cass.city.ac.uk

*Journal of Statistical Software* http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Code Snippet II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd