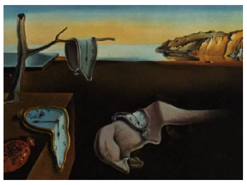

Journal of Statistical Software

June 2015, Volume 65, Code Snippet 3. *http://www.jstatsoft.org/*

An Improved Evaluation of Kolmogorov's Distribution

Luis Carvalho
Boston University

Abstract

We propose a new algorithm for computing extreme probabilities of Kolmogorov's goodness-of-fit measure, D_n. This algorithm is an improved version of the method originally proposed by Wang, Tsang, and Marsaglia (2003) based on a result from Durbin (1973). The new algorithm keeps the same numerical precision of the Wang *et al.* (2003) method, but is more efficient: it features linear instead of quadratic space complexity and has better time complexity for a common range of input parameters of practical importance. The proposed method is implemented in the R package **kolmim**, which also includes an improved routine to perform one-sample two-sided exact Kolmogorov-Smirnov tests.

Keywords: Kolmogorov-Smirnov test.

1. Introduction

Given an ordered set of n uniform $[0, 1)$ variates, $x_1 < \cdots < x_n$, Kolmogorov (1933) suggested $D_n = \max\{D_n^-, D_n^+\}$ as a goodness-of-fit measure, where:

$$D_n^- = \max_{i=1,\ldots,n} \left\{ x_i - \frac{i-1}{n} \right\} \quad \text{and} \quad D_n^+ = \max_{i=1,\ldots,n} \left\{ \frac{i}{n} - x_i \right\}. \tag{1}$$

Wang *et al.* (2003) proposed an algorithm to compute $\mathsf{P}(D_n < d)$ based on a result discussed by Durbin (1973) that represents this probability as the middle element in the diagonal of the nth power of a certain matrix. To be precise, if $k = \lceil nd \rceil$ and $0 \le h = k - nd < 1$ so that $d = (k-h)/n$, and $m = 2k - 1$, then

$$K(n, d) \doteq \mathsf{P}(D_n < d) = \left(\frac{n!}{n^n} H^n \right)_{kk}, \tag{2}$$

where H is a $m \times m$ matrix that depends on h only.

The general form of H is simple, but a concrete example is illustrative: for $k = 3$ we have $m = 5$ and

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

where $(x)_+ = \max\{0, x\}$. In general, H is specified in block form as

$$H = \begin{bmatrix} \mathbf{v}_{1:m-1} & L \\ v_m & \mathbf{v}_{1:m-1}^\top \end{bmatrix}, \tag{4}$$

where the vector $\mathbf{v}$ is given by

$$v_j = \begin{cases} (1-h^j)/j!, & j = 1, \dots, m-1, \\ (1-2h^j+[(2h-1)_+]^j)/j!, & j = m, \end{cases} \tag{5}$$

and L is a unit diagonal lower triangular Toeplitz matrix. Note that for $0 \le h < 1/2$ we have $v_m = (1-2h^m)/m!$ and for $1/2 \le h < 1$ we have $v_m = (1-2h^m+(2h-1)^m)/m!$. As we can see, even though H is dense in structure it is sparse in information; we exploit this fact in Section 2 to offer an improved way to compute $K(n, d)$. In Section 3, we discuss a C implementation of the improved method in the R (R Core Team 2015) package **kolmim** (Carvalho 2015), and present example applications and a performance evaluation. Package **kolmim** is available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=kolmim`. Finally, in Section 4, we conclude with a discussion on the performance gains of the new method.

2. Improvements in evaluating $K(n, d)$

Wang *et al.* (2003) provide a C program to compute $K(n, d)$ using an explicit representation of H as a $m \times m$ matrix and taking its nth power iteratively (only $\lfloor \log_2 n \rfloor$ multiplications are necessary.) However, since we only need the kth entry in the diagonal of $T = n!/n^n H^n$, we can just compute and store the kth *row* of T. Moreover, we compress the information in H in two vectors, $\mathbf{v}$ from Equation 5 and $\mathbf{w} = (1/j!)_{j=1}^{m-2}$ capturing the lower diagonals of L, since

$$H = \begin{bmatrix} v_1 & 0 & \mathbf{0}_2 & \cdots & \mathbf{0}_{m-2} & \mathbf{0}_{m-2} \\ \mathbf{v}_{2:m-1} & \mathbf{w}_{1:m-2} & \mathbf{w}_{1:m-3} & \cdots & w_1 & 0 \\ v_m & v_{m-1} & v_{m-2} & \cdots & v_2 & v_1 \end{bmatrix} + S, \tag{6}$$

where $S = (\delta_{i+1,j})_{i,j=1}^m$, δ the Kronecker delta, is the *right shift operator*. Note that the left summand is a lower triangular matrix, and that $\mathbf{w}_{1:m-2} = \mathbf{w}$.

Let us then denote by $\mathbf{q}_i$ the kth row of $i!/n^i H^i$. Since $\mathbf{q}_i = (i/n)\mathbf{q}_{i-1}H$, we profit from the

```
function K($n, d$);
begin
    $k \leftarrow \lceil nd \rceil$; $h \leftarrow k - nd$; $m \leftarrow 2k - 1$; $s \leftarrow 1$;
    Initialize $\mathbf{v}$ according to Equation 5, $\mathbf{w} = (1/j!)_{j=1}^{m-2}$, and $\mathbf{q} = (\delta_{kj})_{j=1}^{m}$;
    for $i = 1, \ldots, n$ do
        $s \leftarrow s \cdot (i/n)$; $u \leftarrow q_1$;
        $q_1 \leftarrow$ DOT($\mathbf{v}, \mathbf{q}$) $\cdot s$;
        for $j = 2, \ldots, m - 1$ do
            $a \leftarrow u$; $u \leftarrow q_j$;
            $q_j \leftarrow$ [DOT($\mathbf{w}_{1:m-j}, \mathbf{q}_{j:m-1}$) $+ v_{m-j+1} q_m + a] \cdot s$;
        end
        $q_m \leftarrow (v_1 q_m + u) \cdot s$;
    end
    return $q_k$;
end
```

Algorithm 1: Computing $K(n, d) = (\mathbf{q}_n)_{\lceil nd \rceil}$ by iteratively updating $\mathbf{q}_i$.

formulation in Equation 6 to compute $\mathbf{q}_i$ as follows:

$$(\mathbf{q}_i)_j = \begin{cases} (i/n) \cdot \mathbf{v}^\top \mathbf{q}_{i-1}, & j = 1, \\ (i/n) \cdot [(\mathbf{w}_{1:m-j})^\top (\mathbf{q}_{i-1})_{j:m-1} + v_{m-j+1} (\mathbf{q}_{i-1})_m + (\mathbf{q}_{i-1})_{j-1}], & j = 2, \ldots, m-1, \\ (i/n) \cdot [(\mathbf{q}_{i-1})_m v_1 + (\mathbf{q}_{i-1})_{m-1}], & j = m. \end{cases} \tag{7}$$

In practice, we do not need to store $\mathbf{q}_i$ for each i, but just use one vector representing $\mathbf{q}_i$ and update it in-place. However, due to the right shift operator in H we need to compute $\mathbf{q}_i$ from $j = 1$ to m. We summarize the procedure in Algorithm 1. Routine `DOT` returns the dot product between two vectors and can be found as a level 1 routine in many (high performance) **BLAS** implementations as `ddot`.

From Algorithm 1 we see that we incur in a $O(nm^2)$ time complexity due to the n iterations of m dot products; the original procedure of Wang *et al.* (2003) requires $O(m^3 \log_2 n)$ due to $\lfloor \log_2 n \rfloor$ matrix multiplications for the power calculations. Since $m = O(nd)$, these complexities translate to $O(n^3 d^3 \log_2 n)$ and $O(n^3 d^2)$ in terms of n and d for the original and proposed methods, respectively.

3. Implementing $K(n, d)$

The original C program from Wang *et al.* (2003) is coded in the routine `pKolmogorov2x` in the R base package **stats**. We implement Algorithm 1 in a C routine, `pkolmim`, that is used in a homonymous R function in package **kolmim**. Since `pkolmim` computes K, the cumulative distribution function of D_n, it follows the same common signature of R CDF routines and can take vectors as arguments.

To attain the exact same numerical precision of `pKolmogorov2x`, C routine `pkolmim` follows the practice in Wang *et al.* (2003) of scaling $\mathbf{q}$ iteratively – in their case, the whole intermediate power matrix – by 10^{140}, multiplying or dividing, to avoid under- and overflows, respectively.

Routine `pKolmogorov2x` is used for one-sample two-sided exact Kolmogorov-Smirnov tests in `ks.test`, also in package **stats**. We provide a similar R function in package **kolmim**, `ks.test.imp`, to conduct one-sample two-sided exact Kolmogorov-Smirnov tests using function `pkolmim`.

Examples

We start by assessing the precision of `pkolmim` with a simple example borrowed from the man page of `ks.test`:

```
R> set.seed(1234)
R> x <- rnorm(100, 2)
R> ks.test(x, "pgamma", 3, 2, exact = TRUE)


        One-sample Kolmogorov-Smirnov test

data:  x
D = 0.199, p-value = 0.0006002
alternative hypothesis: two-sided

R> ks.test.imp(x, "pgamma", 3, 2)

        One-sample two-sided exact Kolmogorov-Smirnov test

data:  x
D = 0.199, p-value = 0.0006002
alternative hypothesis: two-sided
```

The numerical precision is exactly the same, as expected. We can also confirm the p-value directly using the Kolmogorov-Smirnov statistic:

```
R> 1 - pkolmim(ks.test(x, "pgamma", 3, 2)$stat, 100)

[1] 0.000600162
```

As another example, in Figure 1 we plot $K(n, d)$, as computed by `pkolmim`, against the faster right-tail approximation suggested by Wang *et al.* (2003):

$$K(n,d) \approx \max\left\{0,\, 1 - 2\exp\left\{-\left(2.000071 + \frac{0.331}{\sqrt{n}} + \frac{1.409}{n}\right)nd^2\right\}\right\}.$$

As can be seen in Figure 1, the values agree well in the region suggested by Wang *et al.* (2003), that is, $nd^2 > 3.76$ when $n > 99$. However, the proposed routine is fast enough to render this approximation unnecessary.

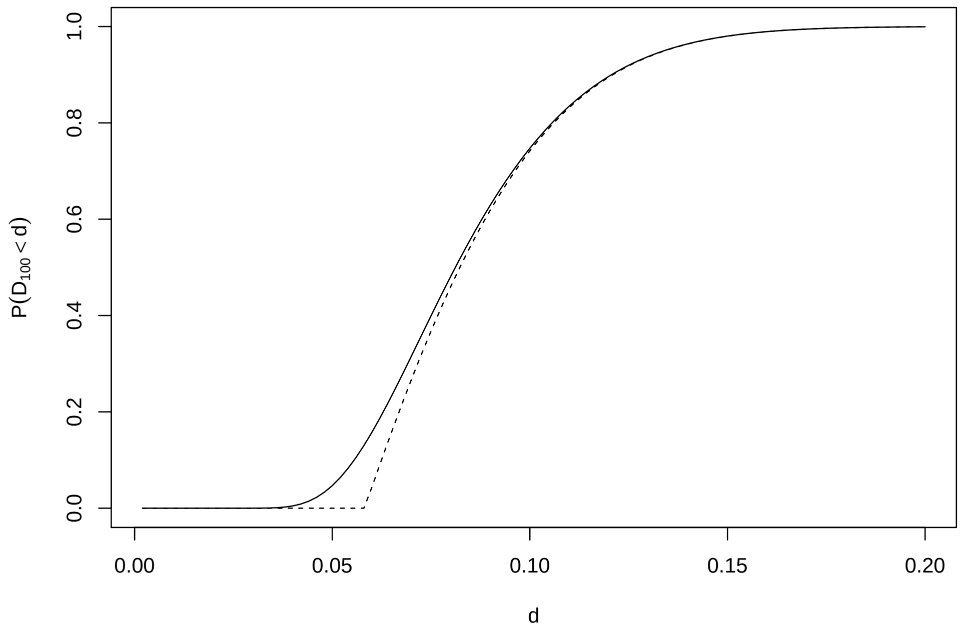

Figure 1: Comparison of probabilities $K(n,d)$, $n = 100$, computed according to `pkolmim` (solid line) and using Wang *et al.* (2003) approximation (dashed line.) Note that the approximation works well in the region suggested by Wang *et al.* (2003), that is, with $n = 100$, when $nd^2 > 3.76$ or $d > 0.194$.

Evaluating performance

The main advantage of `pkolmim` is computational performance. To compare running times, we conduct two experiments by generating ten replicates of ten datasets x_{ij}, $i = 1,\ldots,10$, $j = 1,\ldots,10$, of varying lengths and run `ks.test` and `ks.test.imp` on each dataset while measuring execution times in seconds. In both experiments we adopt the setup from the previous section, again borrowed from the examples in the man page of `ks.test`: we sample $x_{ij} \sim N(2, I_n)$ and compare it to a shifted gamma distribution with shape 3 and rate 2. In the first experiment, we set $n \in \{100, 150, 200, \ldots, 500\}$ to measure running time differences in small datasets, while in the second experiment we take $n \in \{500, 600, \ldots, 1000\}$ to see the effect on larger datasets. The results are summarized in Figure 2, and show that `ks.test.imp` has a running time proportional to n^3 on average. Thus, in practice, for larger datasets the running time of `ks.test` can be orders of magnitude higher than the running time of `ks.test.imp`. The more extreme values for $n = 1000$ in the bottom plot (second experiment) are due to memory swapping to disk and highlight a practical bottleneck from the higher space complexity of `ks.test`.

4. Discussion

The main advantage of the proposed routine arises from the representation in Equation 6: since we store H implicitly using two vectors, $\mathbf{v}$ and $\mathbf{w}$, and the kth row of $T = n!/n^n H^n$ using $\mathbf{q}$, we just need $O(nd)$ space to compute $K(n,d)$ instead of the original $O(n^2d^2)$ space complexity required to store H and T in Wang *et al.* (2003). As a consequence, we also save in

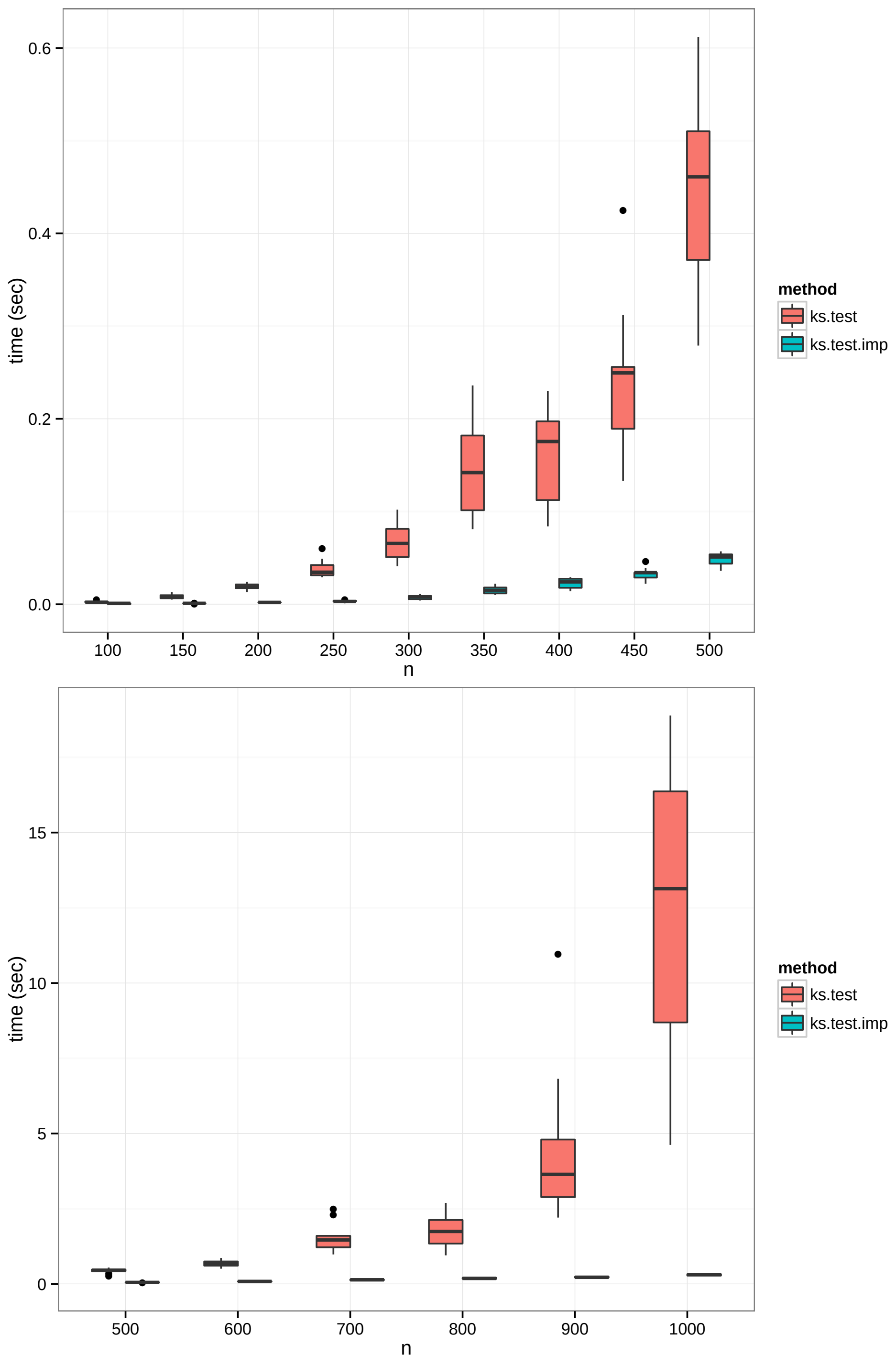

Figure 2: Comparison of running times (in seconds) for one-sample two-sided exact tests implemented using Wang *et al.* (2003) routine in `ks.test` and the improved routine in `ks.test.imp`. We run the experiments on 10 dataset replicates of size n. Top panel focuses on smaller datasets ($n \in \{100, \ldots, 500\}$), while bottom panel covers a broader scale of running times for larger datasets ($n \in \{500, \ldots, 1000\}$).

time complexity since only vector multiplications are needed: while we incur in a $O(n^3d^2)$ time complexity, the original procedure requires $O(n^3d^3 \log_2 n)$ and thus, for $d > O((\log_2 n)^{-1})$, we achieve better time complexity. We expect that these cases cover most common applications since they yield probabilities close to one and that correspond to very significant p-values in two-sided Kolmogorov-Smirnov tests. Moreover, in practice, the original routine might take much longer than the proposed method as n and d—and thus m—grow since the allocation of large matrices can result in swapping memory to disk.

The gains in complexity, both in space and time, from the proposed routine are more critical as n and d get larger. In fact, we are now able to compute more efficiently and with more precision values of $K(n, d)$ that are close to one since we do not have to resort to the practical approximation adopted by Wang *et al.* (2003) when $nd^2 > 7.24$ or when $nd^2 > 3.76$ and $n > 99$.

Acknowledgments

We would like to thank one referee for useful comments on an earlier draft of this article. Carvalho was supported by grant DMS-1107067 from the National Science Foundation.

References

Carvalho L (2015). ***kolmim**: An Improved Evaluation of Kolmogorov's Distribution.* R package version 1.0, URL `http://CRAN.R-project.org/package=kolmim`.

Durbin J (1973). *Distribution Theory for Tests Based on the Sample Distribution Function.* 9. Society for Industrial Mathematics.

Kolmogorov AN (1933). "Sulla determinazione empirica di una legge di distribuzione." *Giornale dell'Istituto Italiano degli Attuari*, **4**(1), 83–91.

R Core Team (2015). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL `http://www.R-project.org/`.

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

Affiliation:

Luis Carvalho
Department of Mathematics and Statistics
Boston University

111 Cummington Mall
Boston, MA, United States of America
E-mail: lecarval@math.bu.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 65, Code Snippet 3 *Submitted:* 2013-05-23
June 2015 *Accepted:* 2014-10-17