

## COMPUTING THE JOINT DISTRIBUTION OF GENERAL LINEAR COMBINATIONS OF SPACINGS OR EXPONENTIAL VARIATES

Fred W. Huffer and Chien-Tai Lin

*Florida State University and Tamkang University*

*Abstract:* We present an algorithm for computing exact expressions for the distribution of the maximum or minimum of an arbitrary finite collection of linear combinations of spacings or exponential random variables with rational coefficients. These expressions can then be manipulated or evaluated using symbolic math packages such as Maple. As examples, we apply this algorithm to obtain the distributions of the maximum and minimum of a moving average process, and the distribution of the Kolmogorov-Smirnov statistic.

*Key words and phrases:* Kolmogorov-Smirnov statistic, moving average process, symbolic computations.

### 1. Introduction

Suppose  $X_1, X_2, \dots, X_n$  are i.i.d. from a uniform distribution on the interval  $(0, 1)$ , and let  $X_{(1)} \leq X_{(2)} \leq \dots \leq X_{(n)}$  be the corresponding order statistics. We define the spacings  $S_1, S_2, \dots, S_{n+1}$  to be the successive differences between the order statistics  $S_i = X_{(i)} - X_{(i-1)}$ , where we take  $X_{(0)} = 0$  and  $X_{(n+1)} = 1$ . Finally, we define  $\mathbf{S}^{(n)} = (S_1, S_2, \dots, S_{n+1})'$ . This paper is concerned with the evaluation of probabilities involving linear combinations of spacings with arbitrary rational coefficients. We present an algorithm for evaluating

$$P(\mathbf{A}\mathbf{S}^{(n)} > t\mathbf{b}), \quad (1)$$

where  $\mathbf{A}$  is any matrix of rational values,  $\mathbf{b}$  is any vector of rational values, and  $t > 0$  is a real-valued scalar. (For vectors  $\mathbf{x} = (x_i)$  and  $\mathbf{y} = (y_i)$ , we take  $\mathbf{x} > \mathbf{y}$  to mean that  $x_i > y_i$  for all  $i$ .) This algorithm produces an exact expression for the probability in (1) which is piecewise polynomial in the argument  $t$ . With this expression, we can use symbolic math packages such as Maple to evaluate (1) to any required degree of precision. The computer programs we have written are also convenient for evaluating quantities which can be expressed as sums of probabilities of the form (1).

Our methods and programs can also be used for computations involving linear combinations of exponential random variables. In fact, by simply reinterpreting the symbols, the expression we obtain for (1) is valid for the corresponding problem with exponential variates obtained by replacing  $\mathbf{S}^{(n)}$  by a vector  $\mathbf{Z}$  of i.i.d. exponentials.

Our approach in this paper has much in common with earlier work reported in Huffer and Lin (1997a, 1999a). However, in this earlier work,  $\mathbf{A}$  had to belong to a special class of binary matrices, and  $\mathbf{b}$  had to be a vector of ones. The method in Lin (1993) can compute (1) for some, but not all, rational matrices  $\mathbf{A}$ . Unfortunately, one cannot characterize the problems that can be solved by this method. With the new algorithm, we can solve a much greater variety of problems.

The approach we use to evaluate (1) depends on the repeated, systematic use of two basic recursions given later as equations (16) and (17). Each recursion is used to re-express a probability like that in (1) by decomposing it into a sum of similar, but simpler components. The same recursions are then applied to each of these components and so on. The process is continued until we obtain components which are simple and easily expressed in closed form.

In Section 2 we present two examples to illustrate our methods. Section 3 gives some definitions and results we use in our algorithm. Section 4 contains a detailed description of the algorithm. Finally, Section 5 contains some remarks on the implementation and performance of the algorithm.

## 2. Examples

It is convenient to regard the probability in (1) as being defined even when the number of columns in  $\mathbf{A}$  is less than  $n + 1$ , the number of entries in  $\mathbf{S}^{(n)}$ . Let  $k$  be the number of columns in  $\mathbf{A}$ . If  $k < n + 1$ , then in computing  $\mathbf{A}\mathbf{S}^{(n)}$  we simply discard the extra entries of  $\mathbf{S}^{(n)}$ , or equivalently, we pad the matrix  $\mathbf{A}$  with extra columns of zeros and define

$$\mathbf{A}\mathbf{S}^{(n)} = (\mathbf{A} | \mathbf{0})\mathbf{S}^{(n)}. \quad (2)$$

Our expressions for (1) are written in terms of a function  $R(j, \lambda)$  defined, for integers  $j \geq 0$  and real values  $\lambda \geq 0$ , by

$$R(j, \lambda) = \begin{cases} \binom{n}{j} t^j (1 - \lambda t)^{n-j} & \text{for } \lambda t < 1, \\ 0 & \text{for } \lambda t \geq 1. \end{cases} \quad (3)$$

The dependence of  $R$  on  $n$  and  $t$  can be left implicit because these values are fixed in any given application of our methods. If we replace  $\mathbf{S}^{(n)}$  in (1) by a vector  $\mathbf{Z}$

of i.i.d. exponential random variables with mean 1, then our expressions remain valid so long as we redefine  $R$  to be  $R(j, \lambda) = t^j e^{-\lambda t} / j!$ .

**Example 1.** For our first example, we compute the probability of the event

$$\bigcap_{i=0}^3 \{S_{i+1} + 2S_{i+2} + 3S_{i+3} + 2S_{i+4} + S_{i+5} > t\}. \tag{4}$$

This problem has the form in (1) with  $\mathbf{A}$  and  $\mathbf{b}$  given by

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 & 2 & 3 & 2 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{5}$$

The probability of (4) as a function of  $t$  gives the distribution of the minimum (call it  $L$ ) of a particular finite moving average of spacings (or exponential random variables). Moving averages of i.i.d. exponential random variables occur when considering the null distribution of spectral estimates in time series. See, for example, Theorem 6.1.1 and formula 7.6.18 in Priestley (1981).

For the probability of (4) we get

$$\begin{aligned} & -17415/64 R(0, 2/3) - 243/8 R(1, 2/3) + 27/4 R(2, 2/3) + 5120/3 R(0, 3/4) \\ & - 21875/24 R(0, 4/5) - 1944/1 R(0, 5/6) + 823543/576 R(0, 6/7) - 40/3 R(0, 1/1) \\ & - 17/12 R(1, 1/1) - 11/12 R(2, 1/1) + 3125/576 R(0, 6/5) + 3/64 R(0, 2/1). \end{aligned} \tag{6}$$

This expression is easy to manipulate and evaluate using symbolic math packages such as Maple. For example, it is easy to study the cdf and density of  $L$  by plotting (6) and its derivative as functions of  $t$ . It is also routine to use (6) to obtain the exact moments of  $L$ .

A closely related problem is to calculate the probability of the event

$$\bigcap_{i=0}^3 \{S_{i+1} + 2S_{i+2} + 3S_{i+3} + 2S_{i+4} + S_{i+5} \leq t\}. \tag{7}$$

As a function of  $t$ , this probability gives the distribution of the maximum of the moving average process.

Since this problem involves “ $\leq$ ” instead of “ $>$ ”, it does not quite fit the form in (1). However, it is easily converted into this form in one of two ways. The first is to note that the event  $\mathbf{A}\mathbf{S}^{(n)} \leq t\mathbf{b}$  is equivalent to  $(-\mathbf{A})\mathbf{S}^{(n)} \geq t(-\mathbf{b})$  which does have the required form (except for the unimportant difference between “ $>$ ” and “ $\geq$ ”). The second is to use an inclusion-exclusion argument to write

$$P(\mathbf{A}\mathbf{S}^{(n)} \leq t\mathbf{b}) = 1 + \sum_{\pi} (-1)^{\#(\pi)} P(\mathbf{A}_{[\pi]}\mathbf{S}^{(n)} > t\mathbf{b}). \tag{8}$$

Here, the sum is taken over all non-empty subsets  $\pi$  of  $\{1, \dots, r\}$  where  $r$  is the number of rows of  $\mathbf{A}$ . We define  $\#(\pi)$  to be the number of elements in  $\pi$ , and use  $\mathbf{A}_{[\pi]}$  to denote the matrix consisting of the given subset of the rows of  $\mathbf{A}$ . Now the terms on the right hand side have the required form.

The second method turns out to require less computational effort in this case. Using (8), we express the probability of the event (7) as a sum of 15 terms and evaluate each of these terms to get

$$\begin{aligned} &1 - 81/1 R(0, 1/3) + 9375/64 R(0, 2/5) - 2696/27 R(0, 1/2) + 4/1 R(1, 1/2) \\ &+ 1701/20 R(0, 2/3) + 40625/72 R(0, 4/5) - 3125/72 R(1, 4/5) - 648/1 R(0, 5/6) \\ &+ 823543/576 R(0, 6/7) - 65536/45 R(0, 7/8) + 319/6 R(0, 1/1) + 1/3 R(1, 1/1) \\ &- 3125/288 R(0, 6/5) + 512/27 R(0, 5/4) - 8/3 R(0, 3/2) + 1/16 R(0, 2/1). \quad (9) \end{aligned}$$

**Example 2.** As in Section 1, let  $X_{(i)} = S_1 + S_2 + \dots + S_i$  for  $i = 1, \dots, n$  denote the order statistics from a sample of size  $n$  from the uniform distribution on  $(0, 1)$ . Our algorithm can be used to compute

$$P\left(\bigcap_{i=1}^n \{c_i < X_{(i)} < d_i\}\right) \quad (10)$$

for any rational values  $c_i$  and  $d_i$ . To put this in standard form (1), we simply rewrite each of the inequalities  $c_i < X_{(i)} < d_i$  as a pair of inequalities  $X_{(i)} > c_i$  and  $-X_{(i)} > -d_i$ , and then use the obvious choices of  $\mathbf{A}$  and  $\mathbf{b}$  to represent the entire collection of inequalities. (Note that if  $c_i \leq 0$  or  $d_i \geq 1$ , the corresponding inequality can be omitted.) In this situation we do not need the variable  $t$  in (1), that is, we set  $t = 1$ .

For a second example, we compute the distribution of the Kolmogorov-Smirnov (K-S) statistic  $D_n = \sup_x |F_n(x) - F(x)|$ , where  $F_n$  is the empirical cdf of  $n$  i.i.d. observations from a continuous distribution  $F$ . The distribution of  $D_n$  does not depend on  $F$ , so we can assume that  $F$  is the uniform distribution on  $(0, 1)$ . Then it is easy to show that

$$\{D_n < y/n\} = \bigcap_{i=1}^n \left\{ (i-y)/n < X_{(i)} < (i-1+y)/n \right\} \quad (11)$$

so that we may apply the discussion of the previous paragraph to compute the distribution of  $D_n$ . For instance, when  $n = 10$  and  $y = 4$ , this discussion leads

to  $P(D_{10} < 4/10) = P(\mathbf{AS}^{(n)} > \mathbf{b})$  where

$$\mathbf{A} = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \text{ and } \mathbf{b} = \begin{pmatrix} -4/10 \\ -5/10 \\ -6/10 \\ -7/10 \\ 1/10 \\ -8/10 \\ 2/10 \\ -9/10 \\ 3/10 \\ 4/10 \\ 5/10 \\ 6/10 \end{pmatrix}. \tag{12}$$

Our algorithm finds the exact value of this probability to be 0.9410107548 by setting  $t = 1$ .

We can handle much larger values of  $n$ . For example, using a program written in Maple we obtained the following numerical results:

$$P(D_{50} < 5/50) \approx 0.3376887295, \quad P(D_{70} < 10/70) \approx 0.8960384325 \tag{13}$$

(rounded to 10 places). However, computation time increases with  $n$ . Using a laptop with a Pentium III processor, the time required to obtain these values was about 17 and 84 seconds, respectively.

It is, of course, well known how to compute the exact distribution of  $D_n$ , at least when  $n$  is not too large. For instance, the original approach of Kolmogorov (see Birnbaum (1952)) can also be used to obtain the results in (13). Explicit piecewise polynomial expressions for the cdf of  $D_n$  may be computed using the work of Drew, Glen, and Leemis (2000). Our algorithm has no advantage over these other approaches for this particular problem. The advantage of our method is its generality and ease of use; we can deal with many variations of the basic K-S statistic just by modifying the sequences  $c_i$  and  $d_i$ , whereas other approaches (such as those of Drew, Glen, and Leemis) require a great deal of thought and re-programming to adapt them to new situations. For example, we may compute the cdf of  $C_n$ , a variant of  $D_n$  introduced by Pyke (1959) and Brunk (1962), simply by replacing (11) by

$$\{C_n < y/(n + 1)\} = \bigcap_{i=1}^n \left\{ (i - y)/(n + 1) < X_{(i)} < (i + y)/(n + 1) \right\}.$$

As another example, Barr and Davidson (1973) give a version of the K-S statistic for censored data in which only the smallest  $m$  order statistics are observed. We

compute the distribution of their statistic simply by replacing the intersection from 1 to  $n$  in (11) by an intersection from 1 to  $m$ , that is, by taking  $c_i = 0$  and  $d_i = 1$  for  $i > m$ . Our algorithm may also be used for computing the distribution of one-sided versions of the K-S statistic.

**Other Applications.** Our approach can be used to evaluate the joint distribution of statistics which can be expressed as linear combinations of spacings or exponential random variables. For example, we can obtain the joint distribution of the sample mean and sample median for random samples from the uniform distribution.

Secondly, we can evaluate various probabilities and moments related to the number of clumps or gaps among randomly distributed points on an interval or circle. In particular, we can use our procedure to compute the distribution of the scan statistic on the interval or the circle. (See Glaz and Balakrishnan (1999) for a review of research on the scan statistic.) Huffer and Lin (1997b, 1999b) discuss these applications. Many of these problems can be solved using the earlier algorithm in Huffer and Lin (1997a), but not all. In particular, our earlier algorithm could not handle problems involving random points on a circle. See Example 3 of Huffer and Lin (1997a). There are other approaches to computing the distribution of the scan statistic on the circle. In particular, we note the work of Weinberg (1980) and Takács (1996). However, these approaches produce answers only when the length of the scanning arc (or window) is rational, and, moreover, they require a separate calculation for each window length. For clumps of a given size, our algorithm leads to a single expression (of the type in (6)) valid for all window lengths.

Another potential application of our method is to the Bayesian bootstrap. Suppose  $\mathbf{A}$  is a  $(n+1) \times p$  matrix whose columns are an i.i.d. sample from an unknown  $p$ -variate distribution  $F$  with mean vector  $\mu$ . The Bayesian bootstrap distribution for  $\mu$ , which is a certain limit of posterior distributions, coincides with the distribution of  $\mathbf{AS}^{(n)}$  (see Choudhuri (1998)). Similarly, the Bayesian bootstrap distribution for the lower moments of  $F$  is the same as that of  $\mathbf{AS}^{(n)}$  with a different choice of the matrix  $\mathbf{A}$  (see Gasparini (1995)). Thus, our algorithm may prove to be of use in calculating characteristics of Bayesian bootstrap distributions.

### 3. Recursions and Elementary Properties

In this section we present the two recursions upon which our algorithm is based. We also list some elementary properties which are used in the course of the algorithm. The recursions (given in equations (16) and (17) below) are stated in terms of a function  $Q$  defined by

$$Q(\mathbf{A}, \mathbf{b}, \lambda, p) = p! R(p, \lambda) P \left( (1 - \lambda t) \mathbf{AS}^{(n-p)} > t\mathbf{b} \right). \quad (14)$$

Note that (3) implies that both  $R$  and  $Q$  are zero when  $\lambda t \geq 1$ . We chose this particular definition of  $Q$  because it leads to a fairly simple form for the recursion (17) below. As in the definition of  $R$ , the dependence of  $Q$  on  $n$  and  $t$  is left implicit. The algorithm works by using the recursions to successively reduce the dimensionality of  $\mathbf{A}$  and  $\mathbf{b}$ . When the dimensions reach zero and both  $\mathbf{A}$  and  $\mathbf{b}$  are empty, we define

$$Q(\emptyset, \emptyset, \lambda, p) = p! R(p, \lambda). \tag{15}$$

Let  $\mathbf{A}$  be an arbitrary matrix. Let  $r$  and  $q$  be the number of rows and columns of  $\mathbf{A}$ . For any  $r \times 1$  vector  $\mathbf{x}$ , define  $A_{i,\mathbf{x}}$  to be the matrix obtained by replacing the  $i$ th column of  $\mathbf{A}$  by  $\mathbf{x}$ . Let  $\mathbf{c} = (c_1, \dots, c_q)'$  be any  $q \times 1$  vector satisfying  $\sum_{i=1}^q c_i = 1$ . Define  $\boldsymbol{\xi} = \mathbf{A}\mathbf{c}$ . Then

$$Q(\mathbf{A}, \mathbf{b}, \lambda, p) = \sum_{i=1}^q c_i Q(\mathbf{A}_{i,\boldsymbol{\xi}}, \mathbf{b}, \lambda, p). \tag{16}$$

This recursion is an immediate consequence of the more general recursion given in Huffer (1988). See Huffer (1988), Lin (1993), and Huffer and Lin (1999a) for its applications.

Suppose  $\mathbf{A} = (a_{ij})$  and  $\mathbf{b} = (b_j)$  satisfy the following (for some  $k \geq 1$ ):

- (R1)  $a_{1j} = 0$  for  $j > k$ ,
- (R2)  $a_{ij} = a_{i1}$  for  $j \leq k$ , (i.e., the first  $k$  columns of  $\mathbf{A}$  are identical),
- (R3)  $a_{11} > 0$  and  $b_1 > 0$ .

Then

$$Q(\mathbf{A}, \mathbf{b}, \lambda, p) = \sum_{i=0}^{k-1} \frac{\delta^i}{i!} Q(\mathbf{A}_{(-i)}^*, \mathbf{b}^* - \delta \mathbf{a}^*, \lambda + \delta, p + i), \tag{17}$$

where  $\delta = b_1/a_{11}$ ,  $\mathbf{A}^*$  is a matrix obtained by deleting the first row of  $\mathbf{A}$ ,  $\mathbf{A}_{(-i)}^*$  is a matrix obtained by deleting the first  $i$  columns of  $\mathbf{A}^*$ ,  $\mathbf{b}^*$  is a vector obtained by deleting the first entry of  $\mathbf{b}$ , and  $\mathbf{a}^*$  is a vector obtained by taking the first column of  $\mathbf{A}$  and deleting the first entry. We give a proof of this recursion at the end of this section. Under the stated conditions, (17) allows us to reduce the dimension of  $\mathbf{A}$  (and  $\mathbf{b}$ ) by deleting one row.

In connection with (17), we also use the following simple recursion. If conditions R1 and R2 hold, but instead of R3 we have  $a_{11} < 0$  and  $b_1 < 0$ , then

$$Q(\mathbf{A}, \mathbf{b}, \lambda, p) = Q(\mathbf{A}^*, \mathbf{b}^*, \lambda, p) - Q(\mathbf{A}^o, \mathbf{b}^o, \lambda, p), \tag{18}$$

where  $\mathbf{A}^*$  and  $\mathbf{b}^*$  are as described above, and  $\mathbf{A}^o$  and  $\mathbf{b}^o$  are obtained by negating the first row of  $\mathbf{A}$  and the first entry of  $\mathbf{b}$  respectively. Note that  $\mathbf{A}^o$  and  $\mathbf{b}^o$  now satisfy R1–R3, so that (17) can be applied to  $Q(\mathbf{A}^o, \mathbf{b}^o, \lambda, p)$ . The recursion (18) is merely a special case of the fact that  $P(C \cap D) = P(C) - P(C \cap D^c)$  for any events  $C$  and  $D$ .

We now list some other properties which are useful in the process of evaluating  $Q$ . These properties are straightforward and proofs are omitted. For any permutation matrix  $\mathbf{G}$ ,

$$(E1) \quad Q(\mathbf{A}, \mathbf{b}, \lambda, p) = Q(\mathbf{AG}, \mathbf{b}, \lambda, p),$$

$$(E2) \quad Q(\mathbf{A}, \mathbf{b}, \lambda, p) = Q(\mathbf{GA}, \mathbf{Gb}, \lambda, p).$$

For any diagonal matrix  $\mathbf{D}$  with strictly positive entries on the diagonal,

$$(E3) \quad Q(\mathbf{A}, \mathbf{b}, \lambda, p) = Q(\mathbf{DA}, \mathbf{Db}, \lambda, p).$$

Let  $(\mathbf{A}, \mathbf{b})$  denote the set of inequalities in (14). Property E1 follows from the exchangeability of the spacings. Property E2 reflects the fact that we can arbitrarily re-order the inequalities in  $(\mathbf{A}, \mathbf{b})$ . Property E3 ensures that we can always make the entries in  $\mathbf{b}$  to be  $\pm 1$  or 0. Note that property E1 allows us to delete any columns in  $\mathbf{A}$  which consist entirely of zeros; we permute the columns of  $\mathbf{A}$  so that the columns of zeros are at the end, and then delete them by using the convention in (2).

The value of  $Q$  remains the same when we delete redundant inequalities from  $(\mathbf{A}, \mathbf{b})$ . Also, if any inequalities in  $(\mathbf{A}, \mathbf{b})$  are contradictory, then  $Q(\mathbf{A}, \mathbf{b}, \lambda, p) = 0$ . In particular, we can delete the  $i$ th row of  $\mathbf{A}$  and the  $i$ th entry of  $\mathbf{b}$  without changing the value of  $Q$  whenever any of the following conditions are true:

$$(S1) \quad a_{ik} \geq a_{jk} \text{ for all } k \text{ and } b_i \leq b_j,$$

$$(S2) \quad a_{ik} \geq 0 \text{ for all } k \text{ (with } a_{ik} > 0 \text{ for some } k) \text{ and } b_i \leq 0,$$

$$(S3) \quad a_{ik} = 0 \text{ for all } k \text{ and } b_i < 0.$$

The value of  $Q$  is 0 whenever any of the following conditions are true:

$$(S4) \quad a_{ik} + a_{jk} \leq 0 \text{ for all } k \text{ and } b_i + b_j \geq 0,$$

$$(S5) \quad a_{ik} \leq 0 \text{ for all } k \text{ and } b_i \geq 0.$$

Properties S3 and S5 allow us to eliminate terms  $Q$  in which  $\mathbf{A}$  has any rows consisting entirely of zeros.

The remainder of this section is devoted to a proof of the recursion in (17). Readers can skip this proof without loss of continuity.

The expression  $\mathbf{AS}^{(n)} > \mathbf{tb}$  stands for a conjunction of inequalities. Under R1–R3, the first of these inequalities is  $D \equiv \{a_{11}(S_1 + \cdots + S_k) > b_1 t\} = \{X_{(k)} > \delta t\}$ . (Recall that  $X_{(k)}$  is the  $k$ th order statistic of our  $n$  random points.) Clearly  $D = \cup_{i=0}^{k-1} D_i$  where  $D_i$  is the event that exactly  $i$  of the random points fall in the interval  $(0, \delta t]$ , that is,  $D_i = \{X_{(i)} \leq \delta t < X_{(i+1)}\}$ . Thus

$$P(\mathbf{AS}^{(n)} > \mathbf{tb}) = P(D \cap \{\mathbf{A}^* \mathbf{S}^{(n)} > \mathbf{tb}^*\}) = \sum_{i=0}^{k-1} P(D_i) P(\mathbf{A}^* \mathbf{S}^{(n)} > \mathbf{tb}^* | D_i). \quad (19)$$

Condition on the event  $D_i$  where  $i < k$ . Define the  $(n - i + 1)$ -dimensional vector  $\mathbf{T}^i$  by  $\mathbf{T}^i = (X_{(i+1)} - \delta t, S_{i+2}, \dots, S_{n+1})'$ . The vector  $\mathbf{T}^i$  gives the spacings



between the  $n - i$  random points lying in the interval  $(\delta t, 1)$ . Under R1–R3, the inequalities in  $\mathbf{A}^* \mathbf{S}^{(n)} > t\mathbf{b}^*$  are

$$\begin{aligned} \left\{ \sum_{\ell=1}^{n+1} a_{j\ell} S_\ell > b_j t \right\} &= \left\{ a_{j1} X_{(i+1)} + \sum_{\ell=i+2}^{n+1} a_{j\ell} S_\ell > b_j t \right\} \\ &= \left\{ a_{j1} (X_{(i+1)} - \delta t) + \sum_{\ell=i+2}^{n+1} a_{j\ell} S_\ell > (b_j - \delta a_{j1}) t \right\} \\ &= \left\{ \sum_{\ell=i+1}^{n+1} a_{j\ell} T_{\ell-i}^i > (b_j - \delta a_{j1}) t \right\} \end{aligned}$$

for  $j \geq 2$ . Thus we have  $P(\mathbf{A}^* \mathbf{S}^{(n)} > t\mathbf{b}^* | D_i) = P(\mathbf{A}_{(-i)}^* \mathbf{T}^i > t(\mathbf{b}^* - \delta \mathbf{a}^*) | D_i)$ .

Now note that, conditional on  $D_i$ , the points in  $(\delta t, 1)$  are i.i.d. uniformly distributed on this interval. Thus, using  $\mathcal{L}$  to mean “the law of”, we have  $\mathcal{L}(\mathbf{T}^i | D_i) = \mathcal{L}((1 - \delta t)\mathbf{S}^{(n-i)})$  which leads to  $P(\mathbf{A}^* \mathbf{S}^{(n)} > t\mathbf{b}^* | D_i) = P((1 - \delta t)\mathbf{A}_{(-i)}^* \mathbf{S}^{(n-i)} > t(\mathbf{b}^* - \delta \mathbf{a}^*))$ . Substituting this in (19), we obtain

$$P(\mathbf{A}\mathbf{S}^{(n)} > t\mathbf{b}) = \sum_{i=0}^{k-1} \binom{n}{i} (\delta t)^i (1 - \delta t)_+^{n-i} P\left((1 - \delta t)\mathbf{A}_{(-i)}^* \mathbf{S}^{(n-i)} > t(\mathbf{b}^* - \delta \mathbf{a}^*)\right).$$

Here we use  $(x)_+ = \max(x, 0)$  to denote the positive part of  $x$ .

This last equation holds for all  $n$  and all matrices  $\mathbf{A}$ . If we replace  $n$  by  $n - p$  and  $\mathbf{A}$  by  $(1 - \lambda t)\mathbf{A}$ , and also multiply both sides of the equation by  $p!R(p, \lambda)$ , then we obtain (after a little algebra) the recursion (17). Note that, when replacing  $\mathbf{A}$  by  $(1 - \lambda t)\mathbf{A}$ , we must also replace the quantities  $\delta = b_1/a_{11}$  and  $\mathbf{a}^*$  (which depend on  $\mathbf{A}$ ) by  $\delta/(1 - \lambda t)$  and  $(1 - \lambda t)\mathbf{a}^*$  respectively.

When  $\mathbf{A}$  consists of a single row, then  $\mathbf{A}^*$  and  $\mathbf{b}^*$  are “empty”. The definition in (15) has been chosen so that (17) remains true even in this case.

#### 4. The Algorithm

Our goal is to evaluate  $P(\mathbf{A}\mathbf{S}^{(n)} > t\mathbf{b}) = Q(\mathbf{A}, \mathbf{b}, 0, 0)$  through a series of steps. At each step, (16) or (17) is used. Each application of these recursions produces terms (on the right hand side of (16) or (17)) which are “simpler” than the parent term (on the left hand side). The net effect is to successively reduce the dimensionality of the  $\mathbf{A}$ -matrices in these terms until at last we arrive at terms which can be evaluated using (15) and (3).

We assume the matrix  $\mathbf{A}$  consists of blocks arranged in a lower triangular fashion. A typical example is given in Figure 1, here  $a, b, c, \dots$ , denote distinct nonzero rational numbers. We say that such a matrix has been put in “standard form”. To be precise, a matrix  $\mathbf{A}$  is in standard form if, for some value of  $B \geq 1$ , we have the following.

- (i)  $\mathbf{A}$  is a  $B \times B$  array of blocks. We refer to these blocks by their positions  $(\ell, m)$  in this array where  $1 \leq \ell, m \leq B$ .
- (ii) Blocks above the diagonal (with  $\ell < m$ ) consist entirely of zeros.
- (iii) Blocks along the diagonal (with  $\ell = m$ ) contain no zeros.

$$\left( \begin{array}{cc|ccc|cc} a & a & 0 & 0 & 0 & 0 & 0 \\ b & b & c & d & d & 0 & 0 \\ 0 & 0 & e & e & f & 0 & 0 \\ \hline g & h & 0 & 0 & q & r & r \\ s & t & u & u & u & v & w \end{array} \right)$$

Figure 1. Example of a matrix in standard form.

Any matrix  $\mathbf{A}$  can be arranged in standard form as follows. First, the matrix is simplified (or the corresponding term is deleted) by using properties S1–S5. In particular, any rows or columns consisting entirely of zeros are eliminated. Then the permutation properties E1 and E2 are employed to arrange the matrix into blocks satisfying (ii) and (iii). There are many ways to do this. In our algorithm, we try to make the region of zeros in the upper right as large as possible, but the success of our algorithm does not depend on our finding the best such arrangement. The approach in our current program is to use E2 to arrange the rows according to the number of nonzero entries in each row, the rows with the fewest nonzero entries being placed at the top of the matrix. Then we use E1 to move the zero entries to the rightmost columns of the matrix as much as possible.

Note that the blocks of a matrix in standard form can (and often do) consist of a single row or column, or even a single entry. Also, if a matrix contains no zero entries, then it is already in standard form (with  $B = 1$ ).

Our standard form is similar to various conditions found in the literature under names like lower block triangular form. However, the latter often carries the requirement that the blocks along the diagonal be square matrices. Also, we require that diagonal blocks be entirely free of zeros and this is not a part of the usual definition of lower block triangular. The use of row and column permutations to put matrices in lower block triangular form (as we do above) is well known. See Duff (1977) for example.

We begin with  $\mathbf{A}$  in standard form and, each time (16) or (17) is used, the resulting  $\mathbf{A}$ -matrices are put back into standard form.

To motivate the form of our algorithm, note that

$$\begin{array}{l} \text{we can always reduce the dimensionality of terms which} \\ \text{satisfy conditions R1 and R2.} \end{array} \quad (20)$$

Here are the various cases. If  $a_{11} > 0$  and  $b_1 > 0$ , apply (17) to this term and reduce the number of rows. If  $a_{11} < 0$  and  $b_1 < 0$ , apply (18) followed by (17) to reduce the number of rows. If  $a_{11} > 0$  and  $b_1 \leq 0$ , then (using property S2) we can delete the first row of  $\mathbf{A}$  and the first entry of  $\mathbf{b}$  without changing the value of the term. Finally, if  $a_{11} < 0$  and  $b_1 \geq 0$ , then (by property S5) the value of the term is zero.

The strategy is to use repeated applications of (16) to enlarge the region of zeros (the union of those blocks above the diagonal) and “drive” the terms closer to conditions R1 and R2. Then, when terms satisfy R1 and R2, we use (17) to reduce the dimensionality. Many different algorithms can be constructed using this basic strategy. We present one such scheme below.

We first describe how (16) gets used in our algorithm. Suppose that  $\mathbf{D}$  is one of the blocks in the standard form for  $\mathbf{A}$ , and that the first row of  $\mathbf{D}$  contains distinct values  $\alpha$  and  $\beta$  in columns  $i$  and  $j$  of  $\mathbf{A}$ . To be precise, assume that  $\mathbf{D}$  is the  $(\ell, m)$  block and that it begins in row  $r$  of  $\mathbf{A}$ . Then  $\alpha = a_{ri}$  and  $\beta = a_{rj}$ . We can use (16) to simplify  $\mathbf{A}$  in either of the following two cases.

- Case 1.*  $\ell = m$ . Take the vector  $\mathbf{c}$  in (16) to have entries  $c_i = \beta/(\beta - \alpha)$ ,  $c_j = -\alpha/(\beta - \alpha)$ , and  $c_k = 0$  for  $k \neq i, j$ .
- Case 2.*  $\ell > m$  and  $a_{ki} = a_{kj}$  for  $k < r$ . Suppose the last nonzero entry in row  $r$  of  $\mathbf{A}$  occurs in column  $s$ , and the value of this last nonzero entry is  $\gamma$ . Take the vector  $\mathbf{c}$  in (16) to have entries  $c_i = \gamma/(\beta - \alpha)$ ,  $c_j = -\gamma/(\beta - \alpha)$ ,  $c_s = 1$ , and  $c_k = 0$  for  $k \neq i, j, s$ .

In Case 1 or Case 2, the vector  $\boldsymbol{\xi} = \mathbf{A}\mathbf{c}$  will have zeros in entries 1 through  $r$ . Thus, if we use this vector in (16) and rearrange the resulting terms back to standard form, we obtain new terms in which the region of zeros is strictly larger than in the parent term.

We illustrate these cases using the example matrix in Figure 1. The  $(2, 2)$  block gives an instance of Case 1 with  $i = 3$ ,  $j = 4$ , and  $r = 2$ . The  $(3, 1)$  block gives an instance of Case 2 with  $i = 1$ ,  $j = 2$ , and  $r = 4$ .

Combining the ingredients given above, we can now give a complete description of our algorithm. To evaluate  $Q(\mathbf{A}, \mathbf{b}, \lambda, p)$ , do the following.

- (a) Search the nonzero blocks of  $\mathbf{A}$  and find the first block which contains two distinct values in the same row. If there are no such blocks, go to (b). The blocks may be searched in any order so long as a block is never searched until all the blocks lying directly above it have already been searched. (One possible search order is that in Figure 2.) Using property E2, move the row containing the two distinct values so that it becomes the top row of the block. The resulting block must satisfy the conditions of Case 1 or Case 2. Now apply (16).
- (b) If no blocks contain any rows with distinct values, then R1 and R2 must be true. Apply the discussion following (20) to simplify this term.

|   |   |   |    |
|---|---|---|----|
| 1 |   |   |    |
| 2 | 5 |   |    |
| 3 | 6 | 8 |    |
| 4 | 7 | 9 | 10 |

Figure 2. One possible ordering for searching the blocks.

Now apply this same procedure to any terms produced in (a) or (b) above. As the algorithm proceeds, terms which can be evaluated using (15) are collected together. At the termination of the algorithm, these terms make up our answer.

It is clear that every term  $Q(\mathbf{A}, \mathbf{b}, \lambda, p)$  where  $\mathbf{A}$  is in standard form can be simplified by using (a) or (b) above. Moreover, all new terms produced by (a) or (b) involve matrices having either a larger region of zeros or a smaller number of rows than  $\mathbf{A}$ . Thus the algorithm terminates after finitely many steps.

## 5. Implementation and Performance

When writing a computer program to implement this algorithm, there are many issues that must be dealt with. We shall briefly comment on some of these. In our current work we use a C program (available from the authors) to carry out the algorithm and construct expressions like those in (6). We then use Maple to manipulate and evaluate these expressions.

When our C program is executed, a large number of terms are generated by the successive application of (16) and (17). At any intermediate point during execution, the terms that remain to be evaluated may be thought of as a sum of the form

$$\sum_i w_i Q(\mathbf{A}_i, \mathbf{b}_i, \lambda_i, p_i), \quad (21)$$

where the values  $w_i$  are rational numbers. The computer program must decide which of the terms in (21) is the next to be simplified using the process in (a) and (b) of Section 4. Our current program orders the terms in (21) primarily according to the dimension of the matrix  $\mathbf{A}_i$ : the term with the largest dimension is evaluated first. The terms are ordered first according to the number of rows in  $\mathbf{A}_i$ , terms having the same number of rows are then ordered by the number of columns. Next, terms having the same number of rows and columns are ordered by the total number of entries in the blocks which lie on or below the diagonal of  $\mathbf{A}_i$ ; terms with the largest total are evaluated first. (For later use, we introduce notation for this total. Let  $\mathbf{A}$  be any matrix arranged in standard form. Let  $r_\ell \times c_m$  be the dimension of the  $(\ell, m)$  block of  $\mathbf{A}$ . Define the total  $\mathcal{T} = \mathcal{T}(\mathbf{A})$  by  $\mathcal{T}(\mathbf{A}) = \sum_{\ell \geq m} r_\ell c_m$ .) Finally, any terms having the same matrix  $\mathbf{A}_i$  in (21) are grouped together; this improves the efficiency of the program since terms with the same  $\mathbf{A}_i$  are simplified in the same way during applications of (16).

The ordering of terms in (21) described above serves two purposes. First, it facilitates searching the list of terms. Secondly, it improves the efficiency of the algorithm by guaranteeing that no term  $Q(\mathbf{A}_i, \mathbf{b}_i, \lambda_i, p_i)$  will arise more than once in the course of the evaluation process. To see this, remember that all the new terms produced by (a) or (b) of Section 4 involve matrices having either a larger region of zeros (and hence a smaller value of  $\mathcal{T}$ ) or a smaller number of rows than in the parent term.

During the evaluation process, new terms are combined with old terms whenever possible. That is, whenever a new term  $w'_i Q(\mathbf{A}_i, \mathbf{b}_i, \lambda_i, p_i)$  is created which is identical to an old term  $w_i Q(\mathbf{A}_i, \mathbf{b}_i, \lambda_i, p_i)$  (except perhaps for  $w_i$ ), they are combined into a single term  $(w'_i + w_i) Q(\mathbf{A}_i, \mathbf{b}_i, \lambda_i, p_i)$ . The rational values  $w_i$  and  $\lambda_i$ , and the entries in  $\mathbf{A}_i$  and  $\mathbf{b}_i$  are all represented by pairs of integers (the numerator and denominator) and manipulated using exact rational arithmetic.

The algorithm can, in principle, compute (1) exactly for any  $\mathbf{A}$  and  $\mathbf{b}$  with rational entries. However, as we increase the dimensionality of  $\mathbf{A}$ , more and more computer time and memory is required. This is probably unavoidable in any algorithm based primarily on (16) and (17).

It is very difficult to predict how much time and memory will be needed to compute (1) in any particular case. There is no simple relationship between the dimension of the matrix  $\mathbf{A}$  and the amount of computational effort required. One reason for this is that the total number of terms that arise during the course of the algorithm is greatly affected by how often we are able to combine identical terms and by how often we are able to employ the simplification properties S1–S5 in Section 3. The amount of combination and simplification which occurs seems to vary greatly from problem to problem.

By considering  $\mathcal{T}(\mathbf{A})$ , it is possible to give a very crude upper bound for the total number of terms that arise. At each step of the algorithm, one of three things happens: (i) recursion (16) is applied as in Case 1 of Section 4; (ii) recursion (16) is applied as in Case 2; (iii) recursion (17) is used. For each of these possibilities, the new terms which are produced have a strictly smaller value of  $\mathcal{T}$  than the original term. In the worst case, (i) leads to two new terms having values of  $\mathcal{T}$  one less than in the parent term. Repeated application of (i) then gives (in the worst case) successive doublings of the number of terms accompanied by successive reductions of  $\mathcal{T}$  by one. It can be seen that repeated application of possibilities (ii) or (iii) results in a smaller number of terms than this. (A single application of (ii) or (iii) can produce three or more new terms, but this greater number of terms is offset by a greater reduction in the value of  $\mathcal{T}$ .) Arguing in this way, we find that the total number of terms which arise during the evaluation of  $P(\mathbf{A}\mathbf{S}^{(n)} > t\mathbf{b})$  can grow no faster than  $constant \times 2^{\mathcal{T}(\mathbf{A})}$ . This rate of growth is very pessimistic and assumes, for instance, that no combination

or simplification occurs in the course of the algorithm. In real problems, the computational resources required to compute (1) can increase very rapidly with the dimension of  $\mathbf{A}$ , but do not seem to climb as fast as the upper bound would suggest.

To illustrate these remarks, we list in Table 1 the times required by our C program to compute (1) for some matrices  $\mathbf{A}$  having a “moving average” pattern like that in (5). We take  $\mathbf{b}$  to be a vector of ones throughout. The computations were done on a laptop computer with a Pentium III processor and 64 megabytes of RAM. For  $\mathbf{A}$  in (5), computations to (6) took about 2 seconds. If we continue the pattern in (5) by adding another row (with nonzero entries (1, 2, 3, 2, 1)) to the matrix, computation time increases dramatically to about 210 seconds. Adding a sixth row causes a further dramatic increase to about 29,000 seconds. These results, given in the third row of Table 1, show how rapidly computation time can grow with the size of the problem. If, however, in each row of these matrices we replace the pattern (1, 2, 3, 2, 1) by (1, 1, 1, 1, 1), the situation, described in the first row of Table 1, is very different. Computation times are now much smaller, and rate of growth (with the size of the problem) is much slower (the pattern (1, 1, 1, 1, 1) leads to much more combination and simplification of terms).

Table 1. Elapsed time (in seconds) required for various moving average computations (similar to Example 1) performed on a laptop computer with a Pentium III processor.

| Pattern             | 4 rows | 5 rows | 6 rows  |
|---------------------|--------|--------|---------|
| (1) (1, 1, 1, 1, 1) | .010   | .013   | .016    |
| (2) (2, 1, 1, 1, 1) | .12    | 7.1    | 930 †   |
| (3) (1, 2, 3, 2, 1) | 2.0    | 210    | 29000 † |

It is clear from the second and third rows of Table 1 that, for problems of a general nature, our algorithm will be useful only for matrices  $\mathbf{A}$  with fairly modest dimensions. For some special classes of problems, such as those in the first line of Table 1, we can handle somewhat larger matrices. There is one special class of problems where we can handle quite large matrices; these are the problems of type (10) considered in Example 2. (Recall that we were able to calculate tail probabilities for the Kolmogorov-Smirnov statistic with  $n = 70$ .) These problems are special because their solution only requires (17). To see why this is so, consider the matrix  $\mathbf{A}$  in (12) that satisfies conditions R1 and R2 of Section 3. We can immediately apply (17). Moreover, all resulting terms satisfy R1 and R2 so that (see the discussion following (20)) we can apply (17) to them also, and so on. Since we do not have to use (16) for this class of problems, the total number of terms grows more slowly with the dimension of the problem.

Also, since each application of (17) reduces the number of rows by exactly one, we know, at the start of any problem of this type, how many steps will be involved in its solution.

Our current C programs do not actually implement a true arbitrary precision rational arithmetic; we simply store integers (the numerators and denominators) as variables of type “double” or “long double”, depending on the version of the program. As a consequence, if integers become too large, we lose the rightmost digits, and the resulting answers are no longer exact. This happened in the two cases marked with a dagger (†) in Table 1. The answers obtained in these two cases, although not exact, seem to be highly accurate; consistency checks suggest that numerical values computed using these answers are accurate to at least 9 decimal places. In other cases where such errors have occurred we have obtained similar accuracy. However, when answers are not exact, it is generally difficult to determine their accuracy; we do not have any theoretical results allowing us to bound the magnitude of the errors. This problem can be avoided (at the cost of program speed) by implementing the entire algorithm within Maple, which does have exact rational arithmetic. Some progress has been made in this direction; we have written a Maple program implementing that subset of our algorithm needed to handle the problems of type (10) discussed above. We used this program for the calculations in Example 2.

We would like to conclude with a few remarks comparing the algorithm of this paper with that in Huffer and Lin (1997a) and the earlier unpublished algorithm in Lin (1993). We shall refer to them, in the order just mentioned, as A1, A2, and A3. These algorithms are similar in that they all rely on (16). However, this recursion is used rather differently in each of the algorithms. While (17) plays a very important role in A1, there is nothing analogous to it in A2 or A3. Similarly, the formulation of our problem in terms of the function  $Q$ , which is expressly designed for use with (17), is new to A1.

The algorithm A2 is specialized: it can only be used to calculate (1) for a special class of binary matrices  $\mathbf{A}$ . However, for this special class of problems, it is more efficient than either A1 or A3. In particular, A2 is the best algorithm for computing the distribution of the scan statistic on the interval and for the applications considered in Huffer and Lin (1997b). The algorithm A3 is more general than A2, and can compute (1) for a variety of matrices  $\mathbf{A}$  with rational entries. Unfortunately, A3 does not work for all rational matrices (for instance, it fails for the Kolmogorov-Smirnov problems considered in Example 2, and for problems like those in rows 2 and 3 of Table 1), and we have no results to tell us in advance whether it will work on any given problem. Our current algorithm A1 remedies this; it is guaranteed to produce a solution (in finitely many steps) for any rational matrix  $\mathbf{A}$ .

Algorithm A3 uses (16) in a complicated and somewhat ad hoc fashion while A1 uses (16) in a very simple way: the number of nonzero entries in the vector  $\mathbf{c}$  is always two or three. However, the more complicated approach in A3 does seem to be more efficient than A1 on some problems. For instance, A3 is better than A1 on problems involving the circular scan statistic (but even for this restricted class of problems we cannot prove that A3 will always work). It should be possible to modify A1 to include additional ways to use (16). If these new methods of applying (16) are used only when they lead to a decrease in the value of  $\mathcal{T}(\mathbf{A})$ , the argument given in this paper will continue to hold and guarantee that the algorithm leads to a solution. Changing A1 in this way might increase its efficiency, at least for special classes of problems. We plan to investigate these possibilities.

### Acknowledgements

We thank the two referees and an associate editor for suggestions which have improved our paper.

### References

- Barr, D. R. and Davidson, T. (1973). A Kolmogorov-Smirnov test for censored samples. *Technometrics* **15**, 739-757.
- Birnbaum, Z. W. (1952). Numerical tabulation of the distribution of Kolmogorov's statistic for finite sample size. *J. Amer. Statist. Assoc.* **47**, 425-441.
- Brunk, H. D. (1962). On the range of the difference between hypothetical distribution function and Pyke's modified empirical distribution function. *Ann. Math. Statist.* **33**, 525-532.
- Choudhuri, N. (1998). Bayesian bootstrap credible sets for multidimensional mean functional. *Ann. Statist.* **26**, 2104-2127.
- Drew, J. H., Glen, A. G. and Leemis, L. M. (2000). Computing the cumulative distribution function of the Kolmogorov-Smirnov statistic. *Comput. Statist. Data Anal.* **34**, 1-15.
- Duff, I. S. (1977). Permutations to block triangular form. *J. Inst. Math. Appl.* **19**, 339-342.
- Gasparini, M. (1995). Exact multivariate Bayesian bootstrap distributions of moments. *Ann. Statist.* **23**, 762-768.
- Glaz, J. and Balakrishnan, N. (1999). Introduction to scan statistics. In *Scan Statistics and Applications* (Edited by J. Glaz and N. Balakrishnan), 3-24. Birkhäuser, Boston.
- Huffer, F. W. (1988). Divided differences and the joint distribution of linear combinations of spacings. *J. Appl. Probab.* **25**, 346-354.
- Huffer, F. W. and Lin, C. T. (1997a). Computing the exact distribution of the extremes of sums of consecutive spacings. *Comput. Statist. Data Anal.* **26**, 117-132.
- Huffer, F. W. and Lin, C. T. (1997b). Approximating the distribution of the scan statistic using moments of the number of clumps. *J. Amer. Statist. Assoc.* **92**, 1466-1475.
- Huffer, F. W. and Lin, C. T. (1999a). An approach to computations involving spacings with applications to the scan statistic. In *Scan Statistics and Applications* (Edited by J. Glaz and N. Balakrishnan), 141-163. Birkhäuser, Boston.
- Huffer, F. W. and Lin, C. T. (1999b). Using moments to approximate the distribution of the scan statistic. In *Scan Statistics and Applications* (Edited by J. Glaz and N. Balakrishnan), 165-190. Birkhäuser, Boston.



- Lin, C. T. (1993). The computation of probabilities which involve spacings, with applications to the scan statistic. Ph.D. Thesis, Florida State University, Tallahassee.
- Priestley, M. B. (1981). *Spectral Analysis and Time Series*. Academic Press, London.
- Pyke, R. (1959). The supremum and infimum of the Poisson process. *Ann. Math. Statist.* **30**, 569-576.
- Takács, L. (1996). On a test for uniformity of a circular distribution. *Math. Methods Statist.* **5**, 77-98.
- Weinberg, C. R. (1980). A test for clustering on the circle. Ph.D. Thesis, University of Washington, Seattle.

Department of Statistics, Florida State University, Tallahassee, FL 32306, U.S.A.

E-mail: huffer@stat.fsu.edu

Department of Mathematics, Tamkang University, Tamsui 251, Taiwan.

E-mail: chien@math.tku.edu.tw

(Received February 2000; accepted March 2001)