Computational Statistics manuscript No.
(will be inserted by the editor)

Efficient computation of multivariate empirical distribution functions at the observed values

David Lee · Harry Joe

Received: date / Accepted: date

Abstract Consider the evaluation of model-based functions of cumulative distribution functions that are integrals. When the cumulative distribution function does not have a tractable form but simulation of the multivariate distribution is easily feasible, we can evaluate the integral via a Monte Carlo sample, replacing the model-based distribution function by the empirical distribution function. Given a simulation sample of size N, the naive method uses $O(N^2)$ comparisons to compute the empirical distribution function at all N sample vectors. To obtain faster computational speed when N needs to be large to achieve a desired accuracy, we propose methods modified from the popular merge sort and quicksort algorithms that preserve their average $O(N \log_2 N)$ complexity in the bivariate case. The modified merge sort algorithm can be extended to the computation of a d-dimensional empirical distribution function at the observed values with $O(N \log_2^{d-1} N)$ complexity. Simulation studies suggest that the proposed algorithms provide substantial time savings when N is large.

Keywords Integral evaluation · Joint probabilities · Monte Carlo simulation · Sorting algorithms

1 Introduction

There are some functionals of a multivariate cumulative distribution function (cdf) where evaluation is most easily implemented based on the empirical cdf evaluated at each point of a large simulated sample. In this case, it is important to efficiently compute the empirical cdf at the observed values.

Let $F(y_1,\dots,y_d)$ be a d-dimensional, absolutely continuous cdf. Analytic or numerical evaluation of integrals of the form

$$\int h\left(F(y_1,\dots,y_d)\right)\, g(y_1,\dots,y_d)\, \mathrm{d}F(y_1,\dots,y_d), \tag{1}$$

where $h : \mathbb{R}^d \to \mathbb{R}$ is a possibly non-linear function that involves F, is subject to the availability of a tractable cdf. If this is not the case, but simulation from F is computationally easy, then it may be practical to evaluate (1) via Monte Carlo (MC) simulation using the sample counterpart

$$\frac{1}{N}\sum_{i=1}^{N} h\left(\hat{F}(Y_{i1},\dots,Y_{id})\right)\, g(Y_{i1},\dots,Y_{id}), \tag{2}$$

where $\boldsymbol{Y}_1,\dots,\boldsymbol{Y}_N, \boldsymbol{Y}_i = (Y_{i1},\dots Y_{id})^\intercal$, is a random sample from F, and

$$\hat{F}(Y_{i1},\dots,Y_{id}) = \frac{1}{N}\sum_{m=1}^{N} 1\left\{Y_{m1} \le Y_{i1},\dots,Y_{md} \le Y_{id}\right\} \tag{3}$$

David Lee
Department of Statistics, University of British Columbia
Vancouver, BC, Canada V6T 1Z4
E-mail: david.lee@stat.ubc.ca, dav001@gmail.com

Harry Joe
Department of Statistics, University of British Columbia
Vancouver, BC, Canada V6T 1Z4

is the empirical cdf at $\boldsymbol{Y}_i$ based on the sample, with $1\{A\}$ being the indicator function for the event A. In the following, assume g, h are continuous and $h(F) \cdot g$ is integrable with respect to F, so that the convergence of (2) to (1) in probability as $N \to \infty$ is guaranteed via the Glivenko-Cantelli theorem and the law of large numbers. The following example gives a possible situation when such MC evaluation is necessary, together with a quantity of practical interest that has the form (1).

Model examples. Some multivariate models with parsimonious structures have cdf's that are numerically intractable but simulation is relatively easy. Examples include high-order factor copulas (Krupskii and Joe (2013)), Markov trees and truncated vine copulas (Bedford and Cooke (2001); Brechmann et al (2012)). Vine copulas have a tractable density so that likelihood inference is possible, but some joint marginal cdf's are not computationally tractable. The vine and factor models use bivariate copulas as building blocks for the construction of high-dimensional distributions. The bivariate linkages are between observed and latent variables in the case of factor copulas, and among the observed variables for Markov trees and vine copulas. Their stochastic representations allow easy model simulation, but a p-factor copula cdf has a p-dimensional integral, while a d-dimensional regular vine copula cdf is typically an integral with dimension $O(d)$. Thus, it may not be feasible to evaluate (1) directly.

Example of quantities involving the expectation of a function of the cdf. An example of quantity that has the form (1), with g being a constant, is the limiting variance of the Kendall's τ for a bivariate margin (j, k) with cdf F_{jk}. The limiting variance of the sample version, $\hat{\tau}_{jk}$, can be derived using theory of U-statistics (Hoeffding (1948)), and is given by

$$\lim_{n\to\infty} n\mathrm{Var}(\hat{\tau}_{jk}) = 16 \int \left[F_{jk}(y_j, y_k) + \overline{F}_{jk}(y_j, y_k)\right]^2 \mathrm{d}F_{jk}(y_j, y_k) - 4(\tau_{jk} + 1)^2,$$

where τ_{jk} is the true value and $\overline{F}_{jk}(y_j, y_k)$ is the bivariate survival function[1]. The models discussed in the preceding example may have F_{jk}'s that are intractable. If F is parametrized with a vector parameter $\boldsymbol{\theta}$ and the model is fitted by maximum likelihood, and the model-based Kendall's τ (denoted as $\hat{\tau}_{jk,\hat{\boldsymbol{\theta}}}$) is obtained based on the parameter estimate $\hat{\boldsymbol{\theta}}$, then under certain regularity conditions, we have

$$\sqrt{n}\left(\hat{\tau}_{jk,\hat{\boldsymbol{\theta}}} - \tau_{jk,\boldsymbol{\theta_0}}\right) \xrightarrow{d} N\left(0, \frac{\partial \tau_{jk,\boldsymbol{\theta}_0}}{\partial \boldsymbol{\theta}^\intercal} \boldsymbol{I}^{-1} \frac{\partial \tau_{jk,\boldsymbol{\theta}_0}}{\partial \boldsymbol{\theta}}\right)$$

by the delta method, where $\boldsymbol{\theta}_0$ is the true value of $\boldsymbol{\theta}$, $\boldsymbol{I}$ is the Fisher information matrix and

$$\tau_{jk,\boldsymbol{\theta}} = 4 \int F_{jk}(y_j, y_k; \boldsymbol{\theta})\, \mathrm{d}F_{jk}(y_j, y_k; \boldsymbol{\theta}) - 1 = 4 \int\int F_{jk}(y_j, y_k; \boldsymbol{\theta})\, f_{jk}(y_j, y_k; \boldsymbol{\theta})\, \mathrm{d}y_j \mathrm{d}y_k - 1$$

is the population Kendall's τ, with the latter expression valid when F_{jk} is absolutely continuous with density f_{jk}. Then

$$\begin{aligned} \frac{\partial \tau_{jk,\boldsymbol{\theta}_0}}{\partial \boldsymbol{\theta}} &= 4 \int\int \left[\frac{\partial F_{jk}(y_j, y_k; \boldsymbol{\theta}_0)}{\partial \boldsymbol{\theta}} f_{jk}(y_j, y_k; \boldsymbol{\theta}) + F_{jk}(y_j, y_k; \boldsymbol{\theta}_0) \frac{\partial f_{jk}(y_j, y_k; \boldsymbol{\theta}_0)}{\partial \boldsymbol{\theta}}\right] \mathrm{d}y_j \mathrm{d}y_k \qquad (4) \\ &= 4 \int \left[\overline{F}_{jk}(y_j, y_k; \boldsymbol{\theta}_0) + F_{jk}(y_j, y_k; \boldsymbol{\theta}_0)\right] \frac{\partial \log f_{jk}(y_j, y_k; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \mathrm{d}F_{jk}(y_j, y_k; \boldsymbol{\theta}_0), \end{aligned}$$

where the last equality follows via two applications of integration by parts on the first term inside the integrand in (4). This has the form of (1) with a bivariate cdf, and has to be evaluated numerically in order to obtain a variability estimate of $\hat{\tau}_{jk,\hat{\boldsymbol{\theta}}}$ if $\tau_{jk,\boldsymbol{\theta}_0}$ has no analytic expression.

To compute (2), we need to obtain the empirical cdf at each of the N simulated values or observations. For d fixed, the naive way to do this is to make $O(N)$ comparisons for each of the N observations, for a total of $O(N^2)$ complexity. This can be prohibitively costly when N is large, in order for the MC-based evaluation to reach a desired level of accuracy. We suggest more efficient methods for this purpose as modifications of either the merge sort or quicksort algorithms (see, e.g., Knuth (1998); Cormen et al (2009)). The proposed methods preserve the $O(N \log_2 N)$ average complexity of these two common sorting algorithms. To illustrate the main idea of the modified algorithms, we initially focus on bivariate empirical distributions. Then we show how the modified method based on the merge sort can be extended to higher dimensions.

The rest of the paper is organized as follows. In Section 2 we describe the modified merge sort and quicksort algorithms for the computation of bivariate empirical cdf's at the observed values through the use of counters. The modification of the merge sort algorithm for the calculation of higher-dimensional empirical cdf's is described in Section 3. Section 4 contains simulation studies that compare the computational efficiency of various methods, where it is demonstrated that the time savings are substantial for practical MC sample sizes. Section 5 has some concluding remarks.

[1] The relationship $1\left\{Y_{mj} \ge Y_{ij}, Y_{mk} \ge Y_{ik}\right\} = 1\left\{-Y_{mj} \le -Y_{ij}, -Y_{mk} \le -Y_{ik}\right\}$ allows one to obtain the empirical survival function at the same order of complexity as the empirical cdf. We therefore only focus on the cdf in this paper.

2 The modified merge sort and quicksort algorithms for computation of the bivariate empirical cdf

We provide an overview of the merge sort and quicksort algorithms, and their connection to the computation of bivariate empirical cdf's in Section 2.1. The proposed modifications of these algorithms are given in Sections 2.2 and 2.3. Section 2.4 addresses the issues concerning non-absolutely-continuous distributions, when ties are possible.

2.1 Computation of the bivariate empirical cdf using sorting algorithms

The merge sort and quicksort are two common sorting algorithms for a single variable. Based on the divide-and-conquer principle, each has $O(N \log_2 N)$ average complexity and is more efficient than some simple algorithms such as the insertion and selection sort algorithms, which have $O(N^2)$ average complexity. The merge sort algorithm starts with pairs of elements, which are compared and sorted. These sorted pairs are then merged, two at a time, to yield sorted subsequences each of length 4. The procedure is repeated until the full sorted sequence is retrieved. There are $O(N)$ comparisons in each layer of the procedure, but there are only $O(\log_2 N)$ layers as each merge reduces the number of groups by half. This results in a total of $O(N \log_2 N)$ comparisons to be made regardless of the initial degree of "sortedness" of the data. Meanwhile, the quicksort algorithm starts from the full sequence. At each iteration, a pivot is chosen and each element is compared to this pivot; all smaller elements are brought to one side of the pivot and the larger elements to the other side, thus forming two subsequences and completing the sorting of the pivot (i.e., the pivot is at its final position). This is repeated for each subsequence until every element has been sorted. The choice of the pivot can influence the efficiency of the quicksort. If the pivot is such that the resulting subsequences are very unbalanced, at most $O(N)$ iterations are necessary and the worst-case complexity is $O(N^2)$.

Computation of the bivariate empirical cdf (expression (3) with $d = 2$) requires only the ranks of the data to be known. Without loss of generality, assume that each of the vectors $\{Y_{11}, \dots, Y_{N1}\}$ and $\{Y_{12}, \dots, Y_{N2}\}$ is a scrambled sequence of the integers $\{1, \dots, N\}$, without ties due to the assumption of F being absolutely continuous. Suppose the vector $\{Y_{11}, \dots, Y_{N1}\}$ has been sorted by an efficient algorithm (i.e., with $O(N \log_2 N)$ complexity) to result in the sorted sequence $\{Y_{k_1,1}, \dots, Y_{k_N,1}\}$, where k_j is the index of the jth smallest observation, $j = 1, \dots, N$, so that the problem amounts to finding the sequential rank with respect to the second index. That is, for each of the $Y_{k_j,2}$'s, we find its rank among the elements $\{Y_{k_1,2}, \dots, Y_{k_j,2}\}$, as the rest $\{Y_{k_j+1,2}, \dots, Y_{k_N,2}\}$ are such that their corresponding first indices $Y_{k_j+1,1}, \dots, Y_{k_N,1}$ are all larger than $Y_{k_j,1}$ and hence the indicator function in (3) evaluates to zero. In the following, we describe the algorithms to achieve this, and for brevity in notation we assume $\{Y_{11}, \dots, Y_{N1}\}$ is the sorted increasing sequence with corresponding second variable $\{Y_{12}, \dots, Y_{N2}\}$.

2.2 The modified merge sort algorithm

To compute the sequential rank for each element in $\{Y_{12}, \dots, Y_{N2}\}$, we can use a modified merge sort algorithm by adding a counter associated with each element; the pseudocode is displayed in Algorithm 1. In each merge operation involving two vectors, "left" and "right", the counter associated with an element x in the "right" vector is incremented by the number of elements from the "left" vector already inserted into the output vector. The reasoning is that these elements are smaller than x, but have not been counted in the previous merge operations. Elements from the "right" vector inserted before x are not counted as they have already been considered in a previous merge operation. The counters for elements in the "left" vector are left untouched, as (locally) there are no elements preceding them. Figure 1 shows this procedure graphically with 8 observations. A number is added to the top right corner of a digit if its counter has to be incremented; for easier understanding, that number is the increment arising from that merge operation only, and the table on the right keeps track of the increments. The row labelled "total" gives the total number of lesser elements preceding each of them; to match the definition (3), 1 should be added to each final count to include the element itself, and then the count should be divided by N to obtain the empirical distribution function at (Y_{i1}, Y_{i2}).

After executing the `Sort` function in Algorithm 1, the ith entry of the vector z contains the sequential rank for the element i (not for the ith element). This can be rearranged to the order of appearance of the elements at $O(N)$ complexity. This algorithm makes use of the fact that the vector y used for sorting contains distinct integers, so that each observation (with value $y_1[i]$ or $y_2[j]$ in the algorithm) maps uniquely to one index of the z vector.

2.3 The modified quicksort algorithm

Alternatively, the sequential rank of a vector can be computed based on a modification of the quicksort algorithm. As mentioned in Section 2.1, the pivot should be chosen carefully to minimize the possibility of reaching $O(N^2)$ complexity. A fixed-location pivot is not suitable as trends in the data may result in very unbalanced subsequences. An example is when

Algorithm 1 The modified merge sort algorithm for a bivariate empirical cdf (modifications within functions are boxed)

```
1: Input is the matrix $(Y_{11}, Y_{12}), \ldots, (Y_{N1}, Y_{N2})$, where $\boldsymbol{Y}_1 = Y_{11}, \ldots, Y_{N1}$ are sorted and $\boldsymbol{Y}_2 = Y_{12}, \ldots, Y_{N2}$ are paired accordingly. Convert $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ to the marginal ranks $1, \ldots, N$. In the following, the vector $\boldsymbol{Y}_2$ is the input variable $y$.

2: **initialize** global variable $z$ a vector of zeros of the same length as $y$

3: **function** SORT($y$)                              ▷ The main merge sort function; recursive
4:     $N \leftarrow$ length($y$)
5:     **if** $N = 1$ **then**
6:         **return** $y$
7:     **else**
8:         $m \leftarrow \lfloor N/2 \rfloor$
9:         $y_1 \leftarrow$ SORT($y[1{:}m]$); $y_2 \leftarrow$ SORT($y[(m+1){:}N]$)
10:        $y \leftarrow$ MERGE($y_1, y_2$)
11:        **return** $y$
12:    **end if**
13: **end function**

14: **function** MERGE($y_1, y_2$)                    ▷ The function to sort and merge two subsequences
15:     $N_1 \leftarrow$ length($y_1$); $N_2 \leftarrow$ length($y_2$); $N \leftarrow N_1 + N_2$
16:     **initialize** $i \leftarrow 1, j \leftarrow 1, k \leftarrow 1$
17:     $\boxed{\textbf{initialize } b \leftarrow 0}$            ▷ Counts the number of "left" vector elements entered
18:     **initialize** $y$ of length $N$
19:     **while** $i \le N_1$ and $j \le N_2$ **do**
20:         **if** $y_1[i] \le y_2[j]$ **then**
21:             $y[k] \leftarrow y_1[i]$; $\boxed{b \leftarrow b + 1;}$ $i \leftarrow i + 1$
22:         **else**
23:             $y[k] \leftarrow y_2[j]$; $\boxed{z[y_2[j]] \leftarrow z[y_2[j]] + b;}$ $j \leftarrow j + 1$
24:         **end if**
25:         $k \leftarrow k + 1$
26:     **end while**
27:     **if** $i \le N_1$ **then**                   ▷ If "left" vector not yet exhausted
28:         fill in the rest of $y$ with remaining content of the "left" vector
29:     **end if**
30:     **if** $j \le N_2$ **then**                   ▷ If "right" vector not yet exhausted
31:         fill in the rest of $y$ with remaining content of the "right" vector
32:         $\boxed{\text{increment } z \text{ (at indices of remaining "right" elements) by } b}$
33:     **end if**
34:     **return** $y$
35: **end function**

36: Increment every element of $z$ by 1, and then divide by $N$ to obtain the empirical distribution function at the $N$ observations; the $r$th element of $z$ corresponds to the $s$th observation where $Y_{s2} = r$.
```

the bivariate observations have perfect positive or negative dependence, and the first or last element is chosen as the pivot. This amounts to sorting an already sorted vector and each step only reduces the length of the longer sequence by 1. Statistical applications typically involve bivariate observations that are related to each other (in the sense that a trend, not necessarily monotone, can be detected on a scatterplot). To mitigate this issue, we adopt a random pivot (Section 7.3 of Cormen et al (2009)) where each element of a subsequence has the same probability to be chosen.

There are various implementations of the quicksort algorithm. We use the one that results in a stable sort, i.e., preserving the order in case of ties, at the expense of extra storage space. Although in-place quicksort is possible with minimal additional storage, this variant is not stable and is much harder to modify to suit our needs.

In our modification, we again introduce a counter associated with each element. During the scanning of an input vector and placement of its elements into an output vector, we keep track of the number of elements (denoted as b) whose values are smaller than the pivot. When an element larger than the pivot is scanned, we increment the associated counter by b; when the

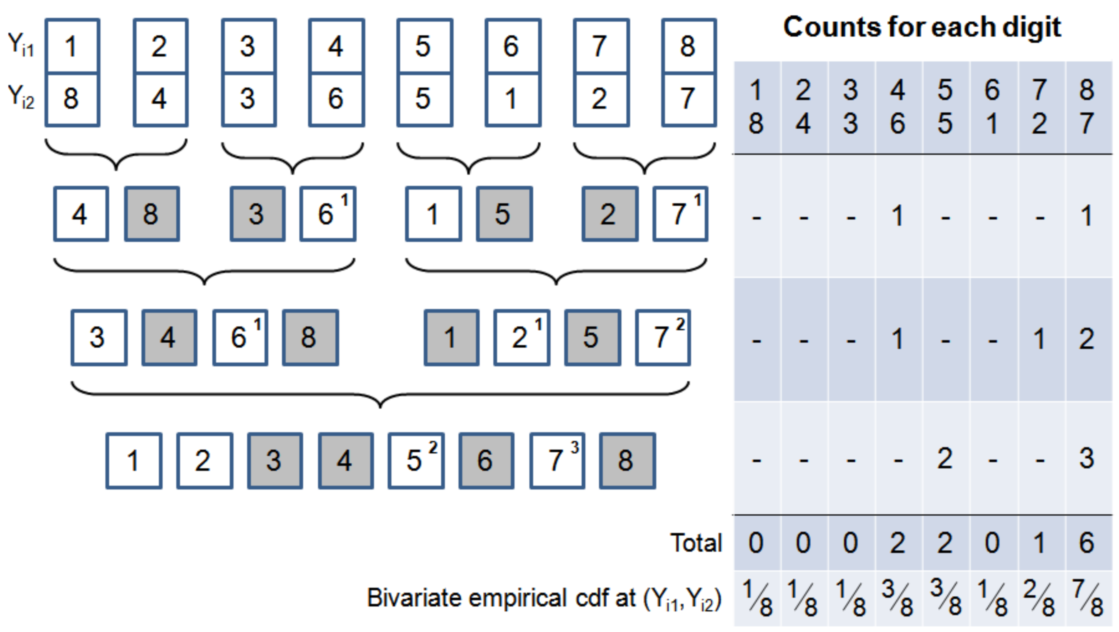


Fig. 1 Illustration of the modified merge sort with $N = 8$ elements. Elements in grey are those coming from the "left" vector and are thus eligible for counting if a larger element from the "right" vector enters after them. The smaller digit at the top right corner of each box is the counter *for that merge operation*. The row for "total" indicates the number of smaller elements to the left of each observation; the bivariate empirical distribution function at (Y_{i1}, Y_{i2}) is obtained by adding 1 to the "total" and then dividing the number by N.

pivot itself is scanned, we increment its counter by b and then increase b by 1, so that subsequent elements that are larger than the pivot will count the pivot as a smaller preceding number. This ensures that no increment is missed and that comparisons leading to the increment of an counter will not be made again at a later stage of the sorting. An example of the modified quicksort in action, based on the same 8 observations as in Section 2.2, is shown in Figure 2. For simplicity in illustration, the middle element (or the $(N/2)$-th element if the subsequence is of an even length N) is chosen as the pivot and highlighted in grey. Each step places elements sequentially to the left of the pivot if they are smaller, or to the right otherwise. The pivot is then highlighted in black, signifying that it has been sorted. The pseudocode of the modified quicksort is given in Algorithm 2. As with the modified merge sort, 1 is to be added to every element of the counter after the whole procedure, and then the count be divided by N to obtain the empirical distribution function at (Y_{i1}, Y_{i2}).

2.4 Accounting for ties for non-absolutely-continuous distributions

If the bivariate distribution is discrete or mixed discrete/continuous, it is possible to have ties in the observed values and their corresponding ranks. Both Algorithms 1 and 2 are stable sorts and preserve the order when there are ties. However, the mapping to the counter (variable z in these algorithms) should be done using a second array, so that identical values in the input vector map to different indices in z. In addition, there are several possibilities that must be considered, depending on whether the ties occur in the first or second variable:

– If ties occur in the first variable (i.e., Y_{m1}) but not the second (i.e., Y_{m2}, elements of the vector being fed into the proposed algorithms), then the second variable should act as the tiebreaker when sorting the first one.
– If ties occur in the second variable but not the first, then the algorithm is still applicable after implementing the aforementioned change of unique mapping to the z vector.
– If there are identical entries (same first and second variables), it is necessary to accommodate for them using alternative methods, such as pre-populating the z vector with non-zero entries. Since the sequential rank algorithm does not search beyond the current observation, it cannot detect the presence of identical entries that will lead to $1\{Y_{m1} \le Y_{i1}, Y_{m2} \le Y_{i2}\} = 1$ for some m after the current observation i.

3 Trivariate and higher-dimensional generalization

In this section, we illustrate how the modified merge sort can be adapted to compute the empirical cdf for trivariate distributions. By iterating, the method extends to higher-dimensional empirical cdfs. The idea is that the algorithm flags the left

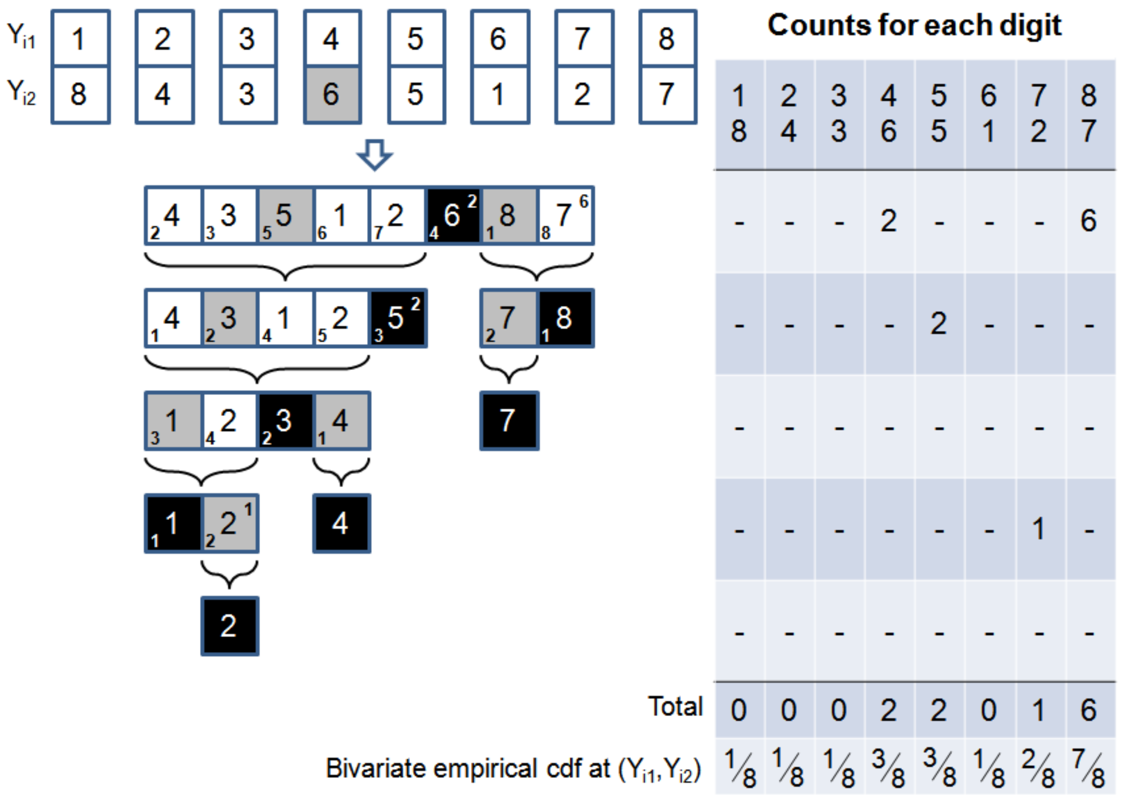


Fig. 2 Illustration of the modified quick sort with $N = 8$ elements. Elements in grey are the pivots chosen for the next step; they are highlighted in black in the next step as they rest in their final locations. The smaller digit at the top right corner of each box is the counter *for that scan operation*. The smaller digit at the bottom left corner is the order of scanning for that element, and is used to assist in counting the increment for the pivot and elements to the right of it. The row for "total" indicates the number of smaller elements to the left of each observation; the bivariate empirical distribution function at (Y_{i1}, Y_{i2}) is obtained by adding 1 to the "total" and then dividing the number by N.

elements for each sort/merge procedure of the second variable as being ineligible for counter increment, as there are no elements preceding them. The corresponding third variables are then sorted, starting by pairs and up to subsequences of the same length as the current iteration of the sorting for the second variable. Taking into account the relative locations of the elements for the second variable, the sorting procedure for the third variable tracks the number of smaller elements in both the second and third variables, and increments the counters accordingly. Specifically, the counter for an element x may only be incremented if it comes from the "right" vectors in both operations, with increment being the number of elements coming from the "left" vectors in both operations that enter before x. An example is given in Figure 3; elements in grey (circle) are those coming from the "left" vectors when sorting the second (third) variables. Same as the bivariate version, the algorithm outputs the total number of lesser elements preceding each of them; 1 should be added and the result divided by N to obtain the empirical cdf.

Algorithm 3 contains the pseudocode of this procedure. An extra indicator (w) is used for storing the eligibility of elements based on the sorting of the second variable. Within the `Merge` function, an inner merge sort on the third variable is performed using the `SortInner` and `MergeInner` functions, where the z counter is incremented.

Because a partial merge sort is nested within each iteration of the sorting for the second variable, the complexity of this algorithm is $O\left(\sum_{i=1}^{p}(2^p + 2^p \cdot i)\right) = O(N \log_2^2 N)$, where $p = \log_2 N$. This is asymptotically better than the naive method, which has $O(N^2)$ complexity. A similar method can be generalized to compute the d-variate empirical distribution function at a complexity of $O(N \log_2^{d-1} N)$.

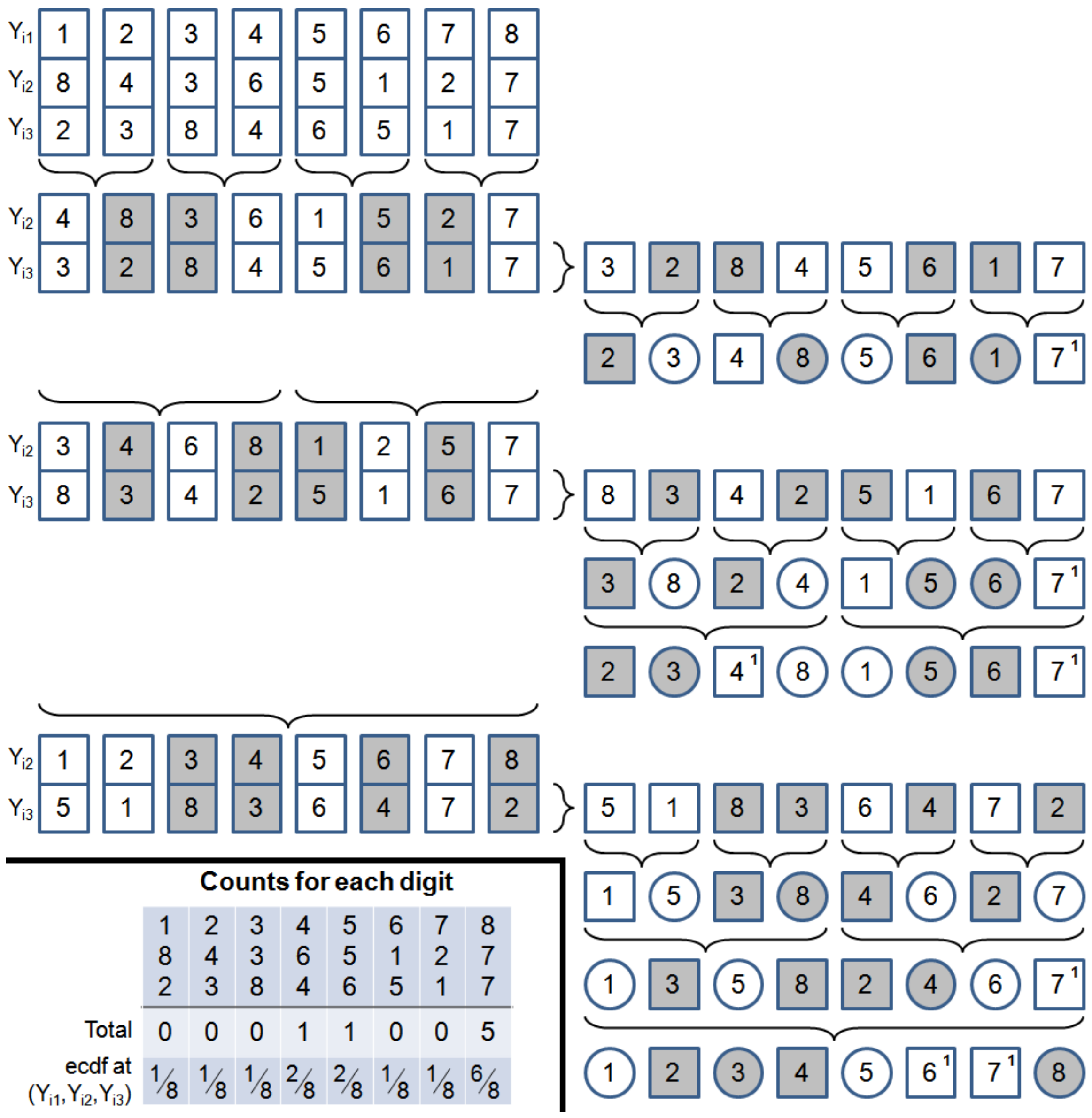
Algorithm 2 The modified quicksort algorithm for a bivariate empirical cdf (modifications within functions are boxed)

```
1: Input is the matrix $(Y_{11},Y_{12}),\dots,(Y_{N1},Y_{N2})$, where $\boldsymbol{Y}_1 = Y_{11},\dots,Y_{N1}$ are sorted and $\boldsymbol{Y}_2 = Y_{12},\dots,Y_{N2}$ are paired accordingly. Convert $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ to the marginal ranks $1,\dots,N$. In the following, the vector $\boldsymbol{Y}_2$ is the input variable $y$.

2: initialize global variable $z$ a vector of zeros of the same length as $y$

3: function Sort($y$)                                    ▷ Recursive function
4:     $N \leftarrow$ length($y$)
5:     if $N \leq 1$ then
6:         return $y$
7:     else
8:         $m \leftarrow$ a random integer in $1{:}N$
9:         initialize $k \leftarrow 1$, $\boxed{b \leftarrow 0}$
10:        initialize empty vectors $v_1, v_2, v_m$
11:        while $k \leq N$ do
12:            if $y[k] < y[m]$ then
13:                push $y[k]$ to the end of $v_1$
14:                $\boxed{b \leftarrow b + 1}$
15:            end if
16:            if $y[k] = y[m]$ then
17:                push $y[k]$ to the end of $v_m$
18:                $\boxed{z[y[k]] \leftarrow z[y[k]] + b;\ b \leftarrow b + 1}$
19:            end if
20:            if $y[k] > y[m]$ then
21:                push $y[k]$ to the end of $v_2$
22:                $\boxed{z[y[k]] \leftarrow z[y[k]] + b}$
23:            end if
24:            $k \leftarrow k + 1$
25:        end while
26:        $v_1 \leftarrow$ Sort($v_1$); $v_2 \leftarrow$ Sort($v_2$)
27:        $y \leftarrow$ concatenate $v_1, v_m, v_2$
28:        return $y$
29:    end if
30: end function

31: Increment every element of $z$ by 1, and then divide by $N$ to obtain the empirical distribution function at the $N$ observations; the $r$th element of $z$ corresponds to the $s$th observation where $Y_{s2} = r$.
```

Algorithm 3 The modified merge sort algorithm for a trivariate empirical cdf. In the following, the list() command combines several variables into a single object; these variables are extracted from the object using the $ operator.

```
1: Input is the matrix $(Y_{11},Y_{12},Y_{13}),\dots,(Y_{N1},Y_{N2},Y_{N3})$, where $\boldsymbol{Y}_1 = Y_{11},\dots,Y_{N1}$ are sorted and $\boldsymbol{Y}_2 = Y_{12},\dots,Y_{N2}$ as well as $\boldsymbol{Y}_3 = Y_{13},\dots,Y_{N3}$ are linked accordingly. Convert $\boldsymbol{Y}_1$, $\boldsymbol{Y}_2$ and $\boldsymbol{Y}_3$ to the marginal ranks $1,\dots,N$. In the following, the vectors $\boldsymbol{Y}_2$ and $\boldsymbol{Y}_3$ are the input variables $x$ and $y$, respectively. The vector $1{:}N$ is inputted as xind.

2: initialize global variables $z$, $w$ vectors of zeros of the same length as $x$
3: (comment: $z$ stores the total count and $w$ an indicator for “left" elements in $x$; they are modified by the following functions)

4: function Sort($x, y,$ xind)                  ▷ The main merge sort function; recursive
5:     $N \leftarrow$ length($x$)
6:     if $N = 1$ then
7:         return list($x = x, y = y,$ xind = xind)
8:     else
9:         $m \leftarrow \lfloor N/2 \rfloor$
10:        $u_1 \leftarrow$ Sort($x[1{:}m], y[1{:}m],$ xind$[1{:}m]$); $u_2 \leftarrow$ Sort($x[(m+1){:}N], y[(m+1){:}N],$ xind$[(m+1){:}N]$)
11:        $u \leftarrow$ Merge($u_1\$x, u_2\$x, u_1\$y, u_2\$y, u_1\$$xind$, u_2\$$xind)
12:        return $u$        ▷ Returns a list containing sorted $x$, associated $y$ and index of original $x$ vector (xind)
13:    end if
14: end function
```

```
function MERGE($x_1, x_2, y_1, y_2, \text{xind}_1, \text{xind}_2$)    ▷ The function for sorting/merging $x$
    $N_1 \leftarrow \text{length}(x_1)$; $N_2 \leftarrow \text{length}(x_2)$; $N \leftarrow N_1 + N_2$
    initialize $i \leftarrow 1, j \leftarrow 1, k \leftarrow 1$
    initialize $x, y, \text{xind}$ of length $N$
    while $i \leq N_1$ and $j \leq N_2$ do
        if $x_1[i] \leq x_2[j]$ then    ▷ Two $x$ elements in increasing order; left element may be eligible for counting
            $x[k] \leftarrow x_1[i]$; $y[k] \leftarrow y_1[i]$; $\text{xind} \leftarrow \text{xind}_1[i]$; $w[\text{xind}_1[i]] \leftarrow 1$; $i \leftarrow i + 1$
        else
            $x[k] \leftarrow x_2[j]$; $y[k] \leftarrow y_2[j]$; $\text{xind} \leftarrow \text{xind}_2[j]$; $w[\text{xind}_2[j]] \leftarrow 0$; $j \leftarrow j + 1$
        end if
        $k \leftarrow k + 1$
    end while
    if $i \leq N_1$ then
        fill in the rest of $x, y, \text{xind}$ with remaining content of the “left" vectors, i.e., $x_1, y_1, \text{xind}_1$
        set the associated indices of $w$ to 1
    end if
    if $j \leq N_2$ then
        fill in the rest of $x, y, \text{xind}$ with remaining content of the “right" vectors, i.e., $x_2, y_2, \text{xind}_2$
        set the associated indices of $w$ to 0
    end if
    SORTINNER($y, \text{xind}$)
    return list($x = x, y = y, \text{xind} = \text{xind}$)
end function

function SORTINNER($y, \text{xind}$)    ▷ The inner sort function for $y$; recursive
    $L \leftarrow \text{length}(y)$
    if $L = 1$ then
        return list($y = y, \text{xind} = \text{xind}$)
    else
        $m \leftarrow \lfloor L/2 \rfloor$
        $v_1 \leftarrow$ SORTINNER($y[1{:}m], \text{xind}[1{:}m]$); $v_2 \leftarrow$ SORTINNER($y[(m+1){:}L], \text{xind}[(m+1){:}L]$)
        $v \leftarrow$ MERGEINNER($v_1\$y, v_2\$y, v_1\$\text{xind}, v_2\$\text{xind}$)
        return $v$    ▷ Returns a list containing sorted $y$ and index of original $x$ vector (xind)
    end if
end function

function MERGEINNER($y_1, y_2, \text{xind}_1, \text{xind}_2$)    ▷ The function for merging $y$ and incrementing counters
    $L_1 \leftarrow \text{length}(y_1)$; $L_2 \leftarrow \text{length}(y_2)$; $L \leftarrow L_1 + L_2$
    initialize $i \leftarrow 1, j \leftarrow 1, k \leftarrow 1, b \leftarrow 0$
    initialize $y, \text{xind}$ of length $L$
    while $i \leq L_1$ and $j \leq L_2$ do
        if $y_1[i] \leq y_2[j]$ then
            $y[k] \leftarrow y_1[i]$; $\text{xind}[k] \leftarrow \text{xind}_1[i]$
            if $w[\text{xind}_1[i]] = 1$ then $b = b + 1$ end if; $i \leftarrow i + 1$    ▷ Two $x, y$ elements in increasing order; left
                                                                                          element eligible for counting
        else
            $y[k] \leftarrow y_2[j]$; $\text{xind}[k] \leftarrow \text{xind}_2[j]$
            if $w[\text{xind}_2[j]] = 0$ then $z[\text{xind}_2[j]] = z[\text{xind}_2[j]] + b$ end if; $j \leftarrow j + 1$
        end if
        $k \leftarrow k + 1$
    end while
    if $i \leq L_1$ then
        fill in the rest of $y, \text{xind}$ with remaining content of the “left" vectors, i.e., $y_1, \text{xind}_1$
    end if
    if $j \leq L_2$ then
        fill in the rest of $y, \text{xind}$ with remaining content of the “right" vectors, i.e., $y_2, \text{xind}_2$
        increment the associated counters ($z$) whose corresponding value of $w$ is 0 by $b$
    end if
    return list($y = y, \text{xind} = \text{xind}$)
end function

Increment every element of $z$ by 1, and then divide by $N$ to obtain the empirical distribution function at the $N$ observations;
the $r$th element of $z$ corresponds to the $r$th observation $(Y_{r1}, Y_{r2}, Y_{r3})$.
```



Counts for each digit

	1	2	3	4	5	6	7	8
	8	4	3	6	5	1	2	7
	2	3	8	4	6	5	1	7
Total	0	0	0	1	1	0	0	5
ecdf at (Y_{i1}, Y_{i2}, Y_{i3})	$1/8$	$1/8$	$1/8$	$2/8$	$2/8$	$1/8$	$1/8$	$6/8$

Fig. 3 Illustration of the modified merge sort for trivariate empirical cdf with $N = 8$ elements. Elements in grey are those coming from the “left” vector in the sorting of the second variables (left-hand side of the figure). Circled elements are those coming from the “left” vector in the sorting of the third variables (right-hand side of the figure). The grey colouring is retained going from the left-hand side to the right-hand side, and starts afresh with a new merge on the left. The counter for each white square element on the right of the figure is incremented by the number of grey circle elements preceding it. The smaller digit at the top right corner of each box is the counter *for that merge operation*. The row for “total” indicates the number of smaller elements to the left of each one; the trivariate empirical distribution function at (Y_{i1}, Y_{i2}, Y_{i3}) is obtained by adding 1 to the “total” and then dividing the number by N.

4 Simulation studies

We conduct simulation studies to compare the efficiency between the naive method and the proposed algorithms, and to demonstrate the importance of using a random pivot in the modified quicksort. All simulations are run on an Intel Core i5-2450M CPU (2.5 GHz) with 6 GB of RAM.

In the first simulation, bivariate samples of sizes $10^2, 10^{7/3}, \ldots, 10^{20/3}, 10^7$ are simulated from the Gumbel copula (Gumbel (1960)):

$$C(u_1, u_2; \delta) = \exp\left\{-\left[\sum_{i=1}^{2}(-\log u_j)^\delta\right]^{1/\delta}\right\}, \quad u_1, u_2 \in [0,1]; \quad \delta \geq 1, \tag{5}$$

with parameter $\delta = 2$ which corresponds to a Kendall’s τ of 0.5. The samples have their first variables sorted using an $O(N \log_2 N)$ algorithm and the corresponding second variables are used as input for finding their sequential ranks. This part of the procedure is timed for each of the three methods: (a) Naive, (b) modified merge sort, and (c) modified quicksort with random pivot, all implemented in Fortran 90 and linked to in R. To reduce sampling variability, the whole procedure is repeated

5 times, and the average time is taken for each sample size. The naive method is only run with a maximum sample size of $10^{16/3} = 215443$ due to its slow speed compared to the other methods.

A similar simulation up to $N = 10^6$ is performed on trivariate observations generated from the exchangeable Gumbel copula with pairwise Kendall's τ of 0.5; the cdf is given by (5) but with variables u_1, u_2, u_3 and the summation from 1 to 3 instead. The computation times for the naive method and the modified merge sort in Section 3 are compared. All empirical cdf's are inspected to ensure results from different methods agree.

The results of the two simulation studies are shown in Figure 4. In each case, the naive method has complexity $O(N^2)$ and is much slower than the modified methods when N is large ($N > 5000$). For the bivariate case, the modified merge sort and quicksort yield similar performance; both methods are efficient for very large sample sizes ($N = 10^6$ or 10^7). In addition, we perform a simulation on independent bivariate and trivariate distributions; the results are similar and are thus omitted.

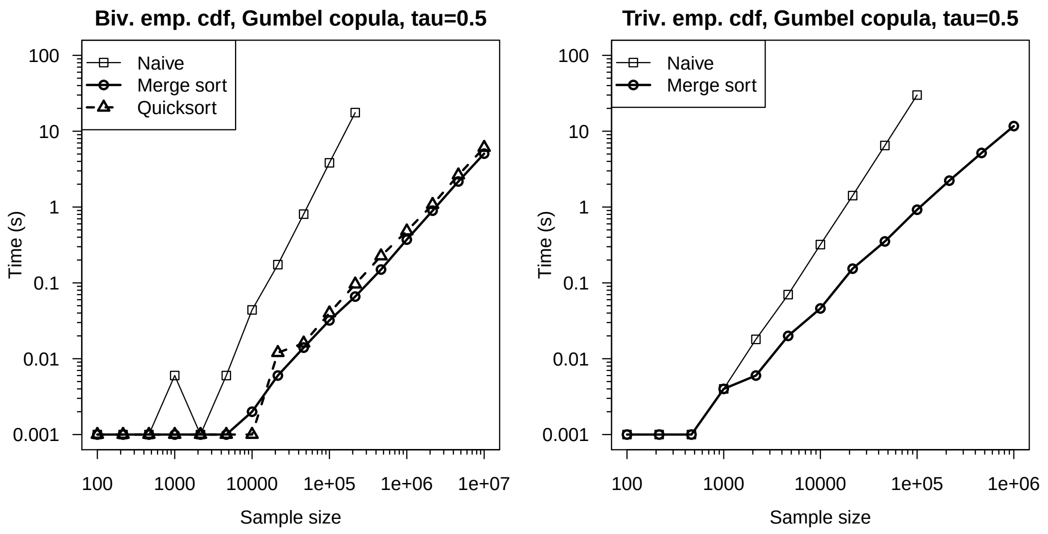


Fig. 4 Time (in seconds) needed to compute the bivariate (left) and trivariate (right) empirical cdf at the observed values using the naive method with $O(N^2)$ complexity and proposed methods with $O(N \log_2^{d-1} N)$ complexity for dimension d.

A third simulation is conducted to demonstrate the importance of using a random pivot for the modified quicksort algorithm. We simulate bivariate samples of various sizes from the Gumbel copula with Kendall's τ equal to 0.5, 0.9 and 0.97, and obtain the empirical cdf using the modified quicksort algorithm with the pivot being (a) randomly chosen with equal probability, and (b) the last element of the sequence being sorted. A comparison of the computation times based on 5 replicates for each sample size is displayed in Figure 5; with a random pivot, the computational efficiency is more or less the same for all three strengths of dependence. However, when the last element is always chosen as the pivot, the time used increases when observations are more concordant, i.e., when there are longer sorted subsequences. This also has implications on the memory usage, as the number of nested recursive calls of the sorting function depends on the number of iterations. To avoid potential time and memory issues, we thus recommend using a random pivot regardless of the structure of the data.

Finally, to demonstrate the versatility of the proposed algorithms, a simulation is performed using bivariate copulas with (a) negative quadrant dependence and (b) permutation asymmetry (i.e., there exists $(u_1, u_2) \in [0,1]^2$ such that $C(u_1, u_2) \neq C(u_2, u_1)$). For (a), we use the t copula:

$$C(u_1, u_2; \rho, \nu) = T_{2,\nu}\left(T_{1,\nu}^{-1}(u_1), T_{1,\nu}^{-1}(u_2); \rho\right), \quad u_1, u_2 \in [0,1]; \quad \rho \in [-1,1]; \quad \nu > 0,$$

where $T_{d,\nu}$ is the cdf of the d-variate t distribution with ν degrees of freedom. We choose $\nu = 5$ and three values of ρ so that the copula has Kendall's τ equal to -0.5, -0.9 and -0.97. For (b), we use the asymmetric Gumbel copula (Tawn (1988)):

$$C(u_1, u_2; \delta, p_1, p_2) = \exp\left\{-\left[(1-p_1)x_1 + (1-p_2)x_2 + (p_1^\delta x_1^\delta + p_2^\delta x_2^\delta)^{1/\delta}\right]\right\}, \quad u_1, u_2 \in [0,1]; \quad p_1, p_2 \in [0,1]; \quad \delta > 1,$$

where $x_i = -\log u_i$, $i = 1, 2$. Various degrees of permutation asymmetry can be obtained by adjusting the values of p_1 and p_2. We fix $p_2 = 1$ and choose $(\delta, p_1) = (1.3, 0.9)$, $(1.42, 0.6)$ and $(2.2, 0.3)$. These three copulas have similar overall dependence strengths but different degrees of permutation asymmetry, with the set $(\delta, p_1) = (2.2, 0.3)$ being the most asymmetric; see Figure 6 for the contour plots of these copula densities[2].

[2] The densities are coupled with standard normal margins for a better illustration of the permutation asymmetry.

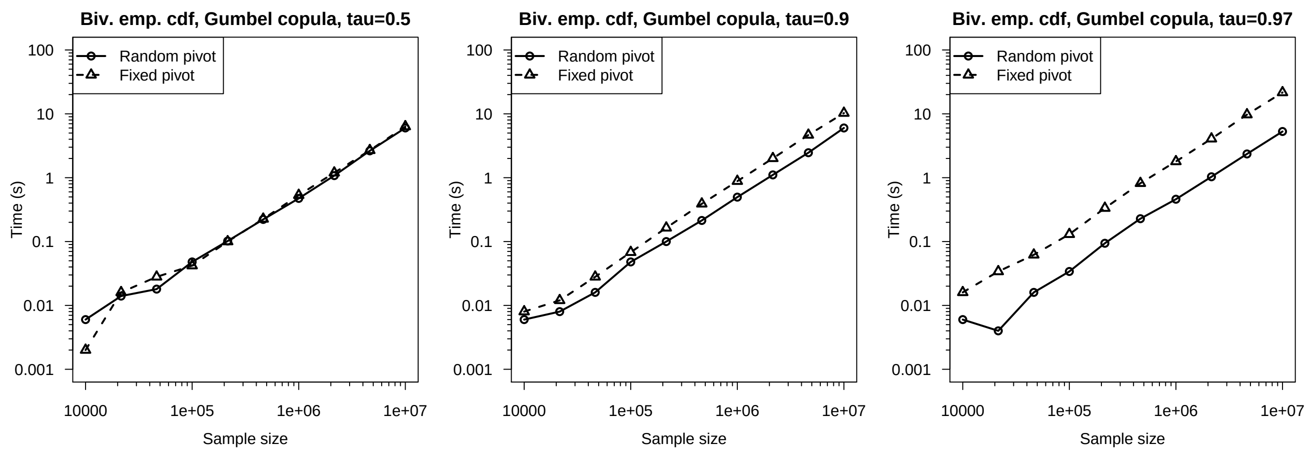


Fig. 5 Time (in seconds) needed to compute the bivariate empirical cdf at the observed values using quicksort with a random pivot (solid lines) and the last element in the subsequence as the pivot (dashed lines), for observations from the Gumbel copula with Kendall's $\tau = 0.5$, 0.9 and 0.97.

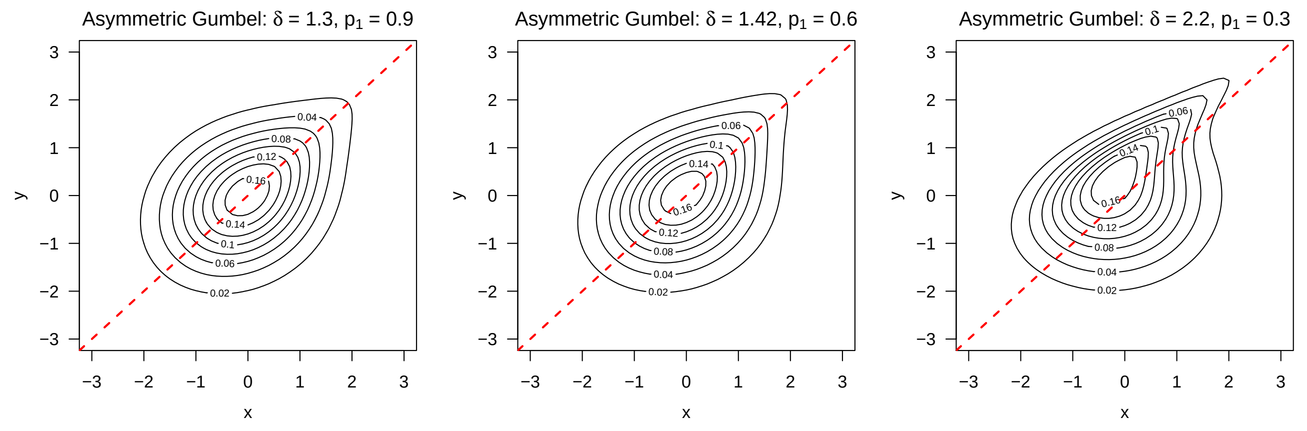


Fig. 6 Contour plots of three asymmetric Gumbel copula densities with standard normal margins for three different sets of parameter values. The contours should be symmetric along the diagonal dashed line $x = y$ if the copula has permutation symmetry.

For each sample simulated from these copulas, we carry out the modified merge sort and quicksort (with random pivot) algorithms. Based on 5 replicates for each copula and sample size combination, the average computation times are obtained and listed in Table 1. We can see that the proposed algorithms have robust performance with respect to different degrees of dependence strengths (both positive and negative) and permutation asymmetry. The computational times also appear to grow at a rate very close to $O(N \log_2 N)$.

Table 1 Time (in seconds) needed to compute the bivariate empirical cdf at the observed values using the proposed methods, for observations from t_5 copulas with parameters $\rho = -0.707$, -0.988 and -0.999 (corresponding to Kendall's τ of -0.5, -0.9 and -0.97), and asymmetric Gumbel copulas with the three sets of parameters given in Figure 6.

Copula	t_5			Asymmetric Gumbel		
Parameter	−0.707	−0.988	−0.999	(1.3,0.9)	(1.42,0.6)	(2.2,0.3)
Sample size	Average time (in s) using modified merge sort					
10^5	0.03	0.03	0.03	0.03	0.03	0.04
10^6	0.36	0.31	0.29	0.37	0.37	0.37
10^7	4.81	4.30	3.72	4.94	4.92	4.93
Sample size	Average time (in s) using modified quicksort					
10^5	0.04	0.05	0.04	0.04	0.05	0.05
10^6	0.47	0.45	0.45	0.49	0.47	0.49
10^7	5.79	5.78	5.28	5.95	5.91	5.79

5 Summary and concluding remarks

We proposed algorithms based on the merge sort and quicksort to compute bivariate empirical cdf's at the observed values efficiently. These methods leverage the $O(N \log_2 N)$ efficiency of these algorithms and scale well to large sample sizes, which are often needed for Monte Carlo evaluation of integrals. The modified merge sort can be generalized to higher dimensions with $O(N \log_2^{d-1} N)$ complexity, where d is the dimension. For our applications in diagnostics for low-dimensional margins of multivariate distributions (such as the one mentioned in the introduction), we are mainly interested in two- and three-dimensional integrals of the form (1).

Through simulation studies, we confirmed that the proposed algorithms can handle large samples (in the order of 10^6 or 10^7), and that the modified merge sort and quicksort algorithms have similar efficiency for bivariate data. We also demonstrated the importance of using a random pivot for the modified quicksort; the one with a fixed pivot is not robust against certain structures of the data that may be common in statistical analysis.

Acknowledgements

This research has been supported by UBC's Four Year Doctoral Fellowship and NSERC Discovery Grant 8698. We would like to thank the editors and the anonymous referee for the comments that lead to a better presentation of the paper.

References

Bedford T, Cooke RM (2001) Probability density decomposition for conditionally dependent random variables modeled by vines. Annals of Mathematics and Artificial Intelligence 32:245–268

Brechmann EC, Czado C, Aas K (2012) Truncated regular vines in high dimensions with application to financial data. Canadian Journal of Statistics 40:68–85

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to Algorithms, 3rd edn. MIT Press, Cambridge, MA

Gumbel EJ (1960) Distributions des valeurs extrêmes en plusieurs dimensions. Publications de l'Institut de Statistique de l'Université de Paris 9:171–173

Hoeffding W (1948) A class of statistics with asymptotically normal distribution. Annals of Mathematical Statistics 19:293–325

Knuth DE (1998) The Art of Computer Programming: Sorting and Searching, vol 3, 2nd edn. Addison-Wesley, Reading, MA

Krupskii P, Joe H (2013) Factor copula models for multivariate data. Journal of Multivariate Analysis 120:85–101

Tawn JA (1988) Bivariate extreme value theory: models and estimation. Biometrika 75:397–415