Tools for Dependent Simulation Input with Copulas

Johann Christoph Strelen
Rheinische Friedrich–Wilhelms–Universität Bonn
Römerstr. 164
53117 Bonn, Germany
strelen@cs.uni.bonn.de

ABSTRACT

Copulas encompass the entire dependence structure of multivariate distributions, and not only the correlations. Together with the marginal distributions of the vector elements, they define a multivariate distribution which can be used to generate random vectors with this distribution. A toolbox is presented which implements input models with this method, for random vectors and time series. Time series are modeled with some general autoregressive processes. The copulas are estimated from observed samples of random vectors. The MATLAB tool calculates the copula, generates random vectors and time series, and provides statistics and diagrams which indicate validity and accuracy of the input model. It is fast and allows for random vectors with high dimensions, for example 100. For this efficiency an intricate data structure is essential. The generation algorithm is also implemented with Java methods.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; G.3 [**Probability and Statistics**]: Distribution Functions, Multivariate Statistics, Random Number Generation, Statistical Software, Time Series Analysis

General Terms

Algorithms

Keywords

Stochastic Simulation, Workload Modeling, Random Variate Generation, Performance Analysis Tools, Performance Modeling, Stochastic Models

1. INTRODUCTION

Stochastic simulation models are an important means for the quantitative analysis of systems. These models are often not closed but must account for influences from the outside. It is well known that failing to model them carefully, especially the load, may lead to incorrect results and ultimately to wrong decisions based on the simulation results. One reason for defective load models may be the ignoring of dependencies, i.e. using independent random variables instead of random vectors or stochastic processes.

Influence from outside of the model, such as load or failure of system components, can be incorporated into the model using observed traces, or with input models: namely random variables, random vectors, or stochastic processes. Data from traces can be used directly. If input is modeled, then the data produced is a realization of the model.

The use of random variates is well understood and has been commonly performed for a long time, but the use of generated random vectors and stochastic processes are more difficult and less popular, are a feature of current research.

In [7], copulas are proposed for the analysis of observed data and for the generation of dependent random variates and time series. Algorithms for these two aspects are given. This technique accounts for the complete dependence structure of the considered sample, and not only for the correlations as in most other methods found in the literature. The only approximation is a discretization with selectable granularity: The copula density is assumed to be piecewise constant. In many examples the authors have found that these input models are valid and accurate.

The algorithms are quite complex and their implementation as computer programs is challenging. For these reasons, programs have been provided in a toolbox, ready for execution. They are described in this paper: A MATLAB program `pwlCopula` and some Java classes, see [14]. The MATLAB program serves three main purposes: Firstly it analyzes a given sample, namely random vectors or a time series where the elements may be random vectors as well, and calculates empirical marginal distributions and the copula. Secondly, it can generate random vectors and time series. Thirdly, the quality of the input model can be judged.

The use of copulas makes the difficult task of finding a multivariate distribution, more facile by performing two easier tasks. The first step is modeling the marginal distributions, the second consists of estimating the copula. Once this is done, it is a straightforward process to generate random vectors. In this paper the marginal distributions are modeled as empirical distributions, the copula is estimated as a frequency distribution.

The new technique contrasts with other proposed input models. Autoregressive processes (AR) model time series with Gaussian random variables. They are conveniently fitted to measured data with the linear Yule-Walker equations.

ARTA-like models (ARTA [2] for univariate time-series, NORTA [3] for random vectors, VARTA [1] for processes of random vectors) allow for general distributions by means of a Gaussian AR or a multivariate Gaussian random variable as a basis whose ran-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5596
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5596

dom variables are transformed into the desired distributions. The correlations of this basic process are different from the desired correlations. Sometimes this results in infeasible correlation matrices of the basic process [5], the *defective matrix problem*. Fitting given autocorrelations and distributions to an ARTA process is possible but not a trivial task.

TES processes [10] rely on empirical distributions of the random variables. They incorporate lag 1 correlations. The interactive software system TEStool serves the purpose of fitting measured data to a TES process.

AR, ARTA-like, and TES processes as input modeling approaches for random vectors and time-series consider only the correlations and not the whole dependence structure. In contrast, copulas can take account of the entire dependency.

In [4, 6] a similar procedure with copulas is proposed. In contrast to this, the new technique here fits samples of data and therefore considers all parts of what is available in the dependence structure. A very efficient data organization allows for high vector dimension like 100 and fast generation.

In [11] the authors propose some kind of nonlinear non-Gaussian multivariate autoregressive process. The marginal random variables can have any distribution. The dependence structure is quite general, where nonlinear dependencies are possible, but this must be analyzed and modeled explicitly. By contrast, the empirical copula approach of this paper extracts the dependence structure directly from the sample data.

This paper begins in Section 2 with a short presentation of the input model technique [7] for random vectors via copulas: Modeling given samples and generating random vectors efficiently in simulation models.

Section 3 presents a sophisticated data structure which speeds up the analysis and the generation of random vectors and is very space efficient. It is based on hashing and sparse vectors. Section 4 exhibits how the modeling technique applies to time series as well. Section 5 describes how the tools provide for the analysis of sample data and for generating random vectors and time series. It also describes the tools' features for validating a dependent input model.

Section 6 is about validating the technique empirically. Examples which are calculated with the main tool, the MATLAB program `pwlCopula` [14], indicate accurate statistical properties of the new technique. Moreover, simulation results with common independent input and dependent input from the new technique are presented and compared.

2. MODELING SAMPLES AND GENERATION WITH PIECEWISE LINEAR EMPIRICAL COPULAS

A copula is a multivariate distribution function $C(\mathbf{u})$, $\mathbf{u} = (u_1,...,u_D) \in [0,1]^D$, for the random vector $\mathbf{U} = (U_1,...,U_D)$ with univariate uniform margins restricted to the unit D-cube $[0,1]^D$. The key theorem according to Sklar clarifies the relationship of dependence and the copula of a multivariate distribution. Let F denote a D-dimensional distribution function with margins $F_1,...,F_D$. Then a D-copula C exists such that for all real $\mathbf{z} = (z_1,...,z_D)$, $F(\mathbf{z}) = C\Big(F_1(z_1),...,F_D(z_D)\Big)$ holds true. If all the margins are continuous, then the copula is unique; in general, it is determined uniquely on the ranges of the marginal distribution functions. Moreover, with quasi-inverses of the marginal distribution functions like

$$F_d^{-1}(u) = \begin{cases} \inf\{z|\,F_d(z) \geq u\} & u > 0 \\ \sup\{z|\,F_d(z) = 0\} & u = 0 \end{cases}$$

for every $\mathbf{u} = (u_1,...,u_D)$ in the unit D-cube, $C(\mathbf{u}) = F\left(F_1^{-1}(u_1),...,F_D^{-1}(u_D)\right)$ holds true. More details of copulas are given in the book by Nelsen [12].

Multivariate random vectors can be generated using copulas in two steps:

1. Generate dependent random numbers $u_1^{(\text{gen})},...,u_D^{(\text{gen})}$ with the multivariate copula as usual.

2. Transform them into the desired marginal distribution, $z_d^{(\text{gen})} = F_d^{-1}(u_d^{(\text{gen})})$, $d = 1,...,D$, using the inverse transformation technique, see the book of Law and Kelton [9], for example.

For step 1, the conditional distribution functions $C_d(u_d|u_1,...,u_{d-1}) = P\{U_d \leq u_d|U_1 = u_1,...,U_{d-1} = u_{d-1}\}$, $d = 1,...,D$, are essential. They can be expressed with the copula density $c(\mathbf{u})$ and some marginal densities.

For the technique in this paper the empirical copulas $C(\mathbf{u})$ are linear in each variable u_d if the other variables are fixed. This linearity holds true within certain regular non-overlapping sub-cubes of the unit D-cube $[0,1]^D$, but in different sub-cubes the slopes are different, in general. The density $c(\mathbf{u})$ of the copula is constant in each of these sub-cubes. Hence, these copulas are piecewise (multi-)linear.

The sub-cubes are defined with a positive integer *granularity parameter K*, namely with the partition $S_1,...,S_K$ of the interval $[0,1]$, $S_1 = [0,1/K]$, $S_j = \big((j-1)/K, j/K\big]$, $j = 2,...,K$, the sub-cubes are $\mathscr{S}_\mathbf{j} = S_{j_1} \times ... \times S_{j_D}$, $\mathbf{j} = (j_1,...,j_D) \in \{1,...,K\}^D$.

The empirical copulas are estimated from a given sample $\mathbf{z}_i = (z_{1,i},\ldots,z_{D,i})$, $i = 1,...,n$, using empirical distribution functions of the marginal distributions of the sample: For each sample value $z_{d,i}$, the corresponding $u_{d,i}$ in the copula space is calculated, basically with the empirical distribution function, and for each sub-cube the number of $\mathbf{u}$-vectors inside are counted. For details see ([7]). The time complexity for this is $\mathrm{O}(D^2 n \log n)$, where n is the sample size.

With the up-operator $\uparrow$: $[0,1] \to \{1,...,K\}$, $\uparrow u = \max\{1, \lceil uK \rceil\}$, the empirical copula density can be denoted as an array f,

$$c(\mathbf{u}) = f_{\uparrow u_1,...,\uparrow u_D}, \;\; \mathbf{u} \in [0,1]^D, \tag{1}$$

since the values within the sub-cubes are constant.

The needed marginal copula densities can be expressed as

$$c_d(u_1,...,u_d) = f^{(d)}_{\uparrow u_1,...,\uparrow u_d}/K^{D-d}, \;\; d = 1,...,D-1, \tag{2}$$

where

$$f^{(d)}_{j_1,...,j_d} = \sum_{j_{d+1}=1}^{K} \cdots \sum_{j_D=1}^{K} f_\mathbf{j}, \;\; (j_1,...,j_d) \in \{1,...,K\}^d.$$

All $f^{(d)}$-arrays together have the space complexity $O(K^D)$, but they are generally sparse. $f^{(D)}$ is the biggest and has K^D elements, most of which will be zero if $K^D \gg n$. We could thus store the big arrays $f^{(d)}$ in a sparse manner, and the storage demand is of order $O(D^2 n)$ if K^D is not small. Instead, a special hash scheme is used for storing the arrays. It also has the space complexity $O(D^2 n)$, but the access time while generating random vectors is much shorter, as will be seen in the next section.

There are some alternatives for the inverse transformation of a random vector $\mathbf{u}^{(\text{gen})}$ of the copula into a random vector $\mathbf{z}^{(\text{gen})}$. Two of them are based on the estimated empirical distribution functions which are provided in the tools:

1. The empirical distribution functions are used directly. Here one can obtain only the values which occur in the sample. This can be appropriate for integer random variables, in particular.

2. Using a linear interpolation, the flat steps of the step distribution function are replaced by straight lines above them with a

positive slope. This enables the modeler to obtain all real values between those he defines as smallest and largest.

Of course, fitted standard distributions could be used as well.

An unknown reviewer gave an interesting hint. There is a simpler way to generate **u**-vectors. Make a list of the nonempty sub-cubes, in some order, assign each interval of length 1/n to one of the sub-cubes, and then construct an index mapping each subinterval of (0,1) of length 1/n to one of these sub-cubes. Then it suffices to generate a single uniform over (0,1) and map it to the appropriate sub-cube with the index. Finally, if we want a continuous distribution, we just generate one point at random uniformly in the selected sub-cube. This idea works for vectors but not for the time series as in Section 4.

3. THE HASH DATA STRUCTURE

This section is about storing and accessing the array elements $f^{(d)}_{j_1,\dots,j_d}$, the f-*values*, and their cumulative sums

$$s^{(d)}_{j_1,\dots,j_d} = \sum_{j=1}^{j_d} f^{(d)}_{j_1,\dots,j_{d-1},j}, \ \ (j_1,\dots,j_d) \in \{1,\dots,K\}^d, \ d = 2,\dots,D,$$

the s-*values*. The f-values are basically the (marginal) densities of the empirical copula, and the s-values are the cumulative distribution functions.

When the f-values and s-values are stored as arrays, even as sparse arrays, the applicability of the copula method is restricted with respect to the granularity K and the dimension D of the random vectors due to complexity. Something like $K \le 1000$ and $D \le 3$, or $K \le 30$ and $D \le 6$ must be observed with full arrays, and with sparse matrices in MATLAB, $K \le 1000$ and $D \le 6$, or $K \le 30$ and $D \le 12$. The following elaborated data structure which realizes the more favorable space and time complexity is much more efficient and therefore makes greater K and D possible. The tools are made with it and allow, for example $K = 4000$ and $D = 4$, $K = 1000$ and $D = 40$, or $K = 100$ and $D = 100$. Moreover, the algorithm is now much faster. For large values K, 30...300 times shorter CPU times are observed for the set-up phase and 24...90 times faster generation of random vectors, compared with a MATLAB program which relies on sparse matrices.

The data structure has features of hashing and of sparse vectors. It allocates sequentially in sparse vectors the cumulative sums $s^{(d)}_{j_1,\dots,j_d}$ for given d and $(j_1,\dots,j_{d-1})$, in increasing order. This allows for a binary search with logarithmic complexity. Only different $s^{(d)}_{j_1,\dots,j_d}$-elements are stored, together with index j_d and $f^{(d)}_{j_1,\dots,j_d}$. A hash function and collision pointers point to these sparse vectors.

The array elements $f^{(d)}_{j_1,\dots,j_d}$ and $s^{(d)}_{j_1,\dots,j_d}$ are, broadly speaking, accessed as follows:

1. Given d and $(j_1,\dots,j_d)$, an entry is searched in the hash table with the hash key $(d, j_1,\dots,j_{d-1})$ and collision pointers. If no entry is found, the $f^{(d)}$-element and the $s^{(d)}$-element are both 0. Otherwise, two pointers *beg*$(j_1,\dots,j_{d-1})$ and *end*$(j_1,\dots,j_{d-1})$ are found in the entry which point to a linear list with the triples $(j_d, f^{(d)}_{j_1,\dots,j_d}, s^{(d)}_{j_1,\dots,j_d})$.

2. In this list, the triples are sorted according to increasing j_d which includes increasing cumulative sums $s^{(d)}_{j_1,\dots,j_d}$. If there is no triple for the given j_d-value, the array element $f^{(d)}_{j_1,\dots,j_d}$ equals 0.

Since an access to an entry in the data structure consists in a hash access plus in search, the access time has two components: For linear search $O(K)$ steps or for binary search $O(\log K)$ steps, and for the calculation of the hash address and for comparisons $O(D)$ steps since D indices must be considered:

THEOREM 1. *With the hash scheme, the access time for the densities and the cumulated densities is $O(K+D)$ with linear search and $O(\log K + D)$ with binary search.*

Remark: Because of the sparsity which is often high-grade, there are generally only a couple of $f^{(d)}_{j_1,\dots,j_d} > 0$, given d and $(j_1,\dots,j_{d-1})$. Therefore quite often a binary search is not faster than a sequential search. The tools apply sequential search.

More precisely, the data structure is as follows: For each $d = 2$, $\dots, D$, there is an *access hash table* and a *triple table*. Each access hash table is organized as follows (see Table 5 in the appendix): The hash function points to the first section with the entries numbered 1 to m'. If a collision occurs, a collision pointer points to the second section with the entries $m'+1$ to m for the collision chains. Otherwise, the collision pointer is 0. Each collision chain is organized with additional collision pointers, and ends with collision pointer 0. Each entry consists of two pointers pointing to the triple table, *beg* and *end*, the sub-cube indices $j_1,\dots,j_{d-1}$, and a collision pointer p which can be 0.

For table d, the hash function is

$$h: \{1,\dots,K\}^{d-1} \to \{1:m'\}, h(j_1,\dots,j_{d-1}) = \lfloor m'(\sigma\theta - \lfloor \sigma\theta \rfloor)\rfloor + 1$$

$$\text{where } m' = \lceil \frac{n}{2} \rceil, \theta = 16\sqrt{5}, \sigma = \frac{j_1}{K} + \frac{j_2}{K^2} + \dots + \frac{j_{d-1}}{K^{d-1}}. \quad (3)$$

The triple table d (Table 6) consists of entries numbered 1,2,..., no more than n entries. Each entry consists of an index j_d, an f-value $f^{(d)}_{j_1,\dots,j_d}$, and the cumulated value $s^{(d)}_{j_1,\dots,j_d}$. Entries with the same index tuple $(j_1,\dots,j_{d-1})$ are stored one after the other, ordered with increasing indices j_d, say $j_{d,1},\dots,j_{d,end-beg+1}$. *beg*$(j_1,\dots,j_{d-1})$ points to the entry with the lowest $j_d = j_{d,1}$, *end*$(j_1,\dots,j_{d-1})$ to the entry with the highest $j_d = j_{d,end-beg+1}$.

Algorithm 1 is used to look-up f-values:

Algorithm 1. Look-up $f^{(d)}_{j_1,\dots,j_d}$, given dimension d and sub-cube indices $j_1,\dots,j_d$.

```
1. Search the index tuple $(j_1,\dots,j_{d-1})$ in the hash table for di-
   mension $d$: Use the hash address $h(j_1,\dots,j_{d-1})$ and follow
   the pointers in the collision chain.
2. **if** the tuple is not found
   **then** **return** 0
   **else** determine *beg* and *end*
        **if** $j_d$ is found between *beg* and *end* in the triple table
        **then return** the density value
        **else return** 0
        **fi**
   **fi**.
```

For generation of random vectors of the empirical copula, the inverse transformation method is used with the conditional distribution functions $C_d(u_d|u_1,\dots,u_{d-1})$. To this end, s-values must be compared. Algorithm 2 serves this purpose.

Algorithm 2. Search smallest j where $b \le s^{(d)}_{j_1,\dots,j_{d-1},j+1}$, given b, dimension d, and sub-cube indices $j_1,\dots,j_{d-1}$.

```
1. Search the index tuple $(j_1,\dots,j_{d-1})$ in the hash table for di-
   mension $d$: Use the hash address $h(j_1,\dots,j_{d-1})$ and follow the
   pointers in the collision chain.
```

```
2.  **if** the tuple is not found
    **then return** ∞
    **else** determine *beg* and *end*
        **if** b = 0
        **then return** 0
        **else** Search smallest $s^{(d)}_{j_1,...,j_{d-1},j+1} \geq b$ for $1 \leq j+1 \leq K$
              in the triple table (with linear or binary search)
            **if** not found
            **then return** ∞
            **else return** $j$
            **fi**
        **fi**
    **fi**.
```

The data structures quoted above with access hash tables and triple tables are built with a sample where the array elements $f^{(d)}_{j_1,...,j_d}$ are also accessed with a hash table. In the tool, these *set-up hash tables* are designed like those above, but all f-values are in the the hash tables themselves. For a given d and $(j_1,...,j_{d-1})$-tuple, all $f^{(d)}_{j_1,...,j_d} > 0$ are in the same collision chain, namely the hash function is (3). Obviously, in such a chain, there can also be $f^{(d)}_{j'_1,...,j'_{d-1},j'_d}$-values, namely when a different index tuple $(j'_1,...,j'_{d-1})$ has the same hash address.

From the set-up hash tables, the access hash tables and the triple tables can be built with a straight-forward algorithm: For each index tuple $(j'_1,...,j'_{d-1})$ for which there is an f-value > 0 exactly one entry remains in the access hash table, and each f-value together with its s-value is stored in a triple table entry, in increasing order. The f-values and the indices j_d are replaced by the *beg*- and *end*-pointers. This algorithm is omitted here.

4. AUTOREGRESSIVE MODELING TIME SERIES

The technique for random vectors can be used for stationary time series as follows: Consider a time series $\mathbf{t}_i$, $i = 1,...,n+w-1$, of D'-dimensional random vectors, $\mathbf{t}_i = (t_{1,i},....,t_{D',i})$. A moving window with a width of w vectors $\mathbf{t}_i, \mathbf{t}_{i+1},...,\mathbf{t}_{i+w-1}$ is taken as sample vectors $\mathbf{z}_i = (\mathbf{t}_i,...,\mathbf{t}_{i+w-1})$, $i = 1,...,n$, hence $D = wD'$ is the dimensionality of the random vectors $\mathbf{z}_i$. With this sample, the marginal distributions and the copula are estimated as in Section 2. The reader may realize that there are only D' different marginal distributions.

Here $\mathbf{t}_i, \mathbf{t}_{i+1},...$ is assumed to be a general kind of autoregressive process, namely $\mathbf{t}_{i+w-1}$ is modeled as a function of $\mathbf{t}_i, \mathbf{t}_{i+1},...,$ $\mathbf{t}_{i+w-2}$ and a random perturbation. The distribution of $\mathbf{t}_{i+w-1}$ is the conditional distribution function $P\{\mathbf{t}_{i+w-1} \leq \mathbf{t}|$ given a realization of $\mathbf{t}_i$, $\mathbf{t}_{i+1},...,\mathbf{t}_{i+w-2}\}$.

The reader may note that $\mathbf{z}_i, \mathbf{z}_{i+1},...$ is some kind of Markov process, since the distribution of $\mathbf{z}_{i+1}$ is completely defined by $\mathbf{z}_i$ and some conditional distribution, given $\mathbf{z}_i$.

The idea of this is as follows: The dependency between all $\mathbf{t}_i$ may be defined completely between two succeeding vectors, for example $\mathbf{t}_i = \alpha\mathbf{t}_{i-1} + (1-\alpha)\mathbf{x}_i$ where $0 < \alpha < 1$, and the $\mathbf{x}_i$ are independent random vectors. In fact, $\mathbf{t}_i$ and $\mathbf{t}_{i+\Delta}$ are dependent for $\Delta > 1$, but this dependency is sufficiently considered with window width $w = 2$. But the dependency between the $\mathbf{t}_i$ may not be defined completely between two succeeding vectors, for example $\mathbf{t}_i = \alpha\mathbf{t}_{i-1} + \beta\mathbf{t}_{i-2} + (1-\alpha-\beta)\mathbf{x}_i$ where $0 < \alpha, \beta$, $\alpha+\beta < 1$. In this case the window width w must be greater than 2 in order to cover the complete dependency (actually, w must be at least 3 in this example).

Remark: There is one exception to the above. Namely, when the granularity parameter K and the sample size n are equal, the complete dependency is considered automatically even with window width $w = 2$. The reason is as follows: When the random vectors $\mathbf{t}^{(\text{gen})}_i$ are generated under these circumstances, the sub-cubes appear in the same order as in the original sample given.

The generation of a time series $\mathbf{t}^{(\text{gen})}_i, i = 1,2,...,$ is different from the case of random vectors. In each generation step i, the first $(m-1)D'$ elements for the new $\mathbf{z}^{(\text{gen})}_i$ are taken from $\mathbf{z}^{(\text{gen})}_{i-1}$, namely its last $(m-1)D'$ elements. Only the last D' elements are generated newly each time:

$$\mathbf{z}^{(\text{gen})}_{i-1} = \begin{pmatrix} \mathbf{t}^{(\text{gen})}_{i-1} \\ \mathbf{t}^{(\text{gen})}_{i} \\ \cdots \\ \mathbf{t}^{(\text{gen})}_{i+w-2} \end{pmatrix} \begin{matrix} \nearrow \\ \cdots \\ \nearrow \end{matrix} \begin{pmatrix} \mathbf{t}^{(\text{gen})}_{i} \\ \cdots \\ \mathbf{t}^{(\text{gen})}_{i+w-2} \\ \mathbf{t}^{(\text{gen})}_{i+w-1} \end{pmatrix} = \mathbf{z}^{(\text{gen})}_i$$

The according explanation holds true for the generated $\mathbf{u}^{(\text{gen})}_i$ -vectors.

The last D' elements of the generated $\mathbf{z}^{(\text{gen})}_i$ series are the desired generated time series $\mathbf{t}^{(\text{gen})}_i$, $i = 1,2,....$

The first vector $\mathbf{z}^{(\text{gen})}_1$ must be initialized somehow, since no older random vector is available. For this purpose, the whole vector can be generated. But there is an initial transient phase where the generated vectors $\mathbf{t}^{(\text{gen})}_i$, $i = 1,2,...,$ are not generally stationary. The modeler should not use the generated vectors in the beginning.

It must be mentioned that for this generation method of time series, two problems must be solved:
1. All parts $\mathbf{t}_j$ of the vectors $\mathbf{z}_i$ must have the same empirical marginal distributions.
2. For each $\mathbf{z}_i = (\mathbf{t}_i,...,\mathbf{t}_{i+w-1})$, there must be an other vector $\mathbf{z}_j$ with $(\mathbf{t}_{i+1},...,\mathbf{t}_{i+w-1})$ in the lower places like $(\mathbf{t}_{i+1},...,\mathbf{t}_{i+w-1},\mathbf{t}_k)$.
If one of these two conditions is violated it may occur that all $\mathbf{u}^{(\text{gen})}_i = F(\mathbf{z}^{(\text{gen})}_i)$ are in sub-cubes with zero density. Hence, no such vector can be generated, the generation algorithm runs into a dead end.

Both postulations are not immediately true and must be forced explicitly. In the tools, the second problem is omitted as follows: In the sample vectors $\mathbf{z}_i = (\mathbf{t}_i,...,\mathbf{t}_{i+w-1})$, $i = 1,...,n$, each vector $\mathbf{t}_j$ with $j > n$ is replaced with $\mathbf{t}_{j-n}$. Thereby each vector $\mathbf{t}_j$, $j = 1,...,n+w-1$, occurs exactly once in each position of the sample vectors. Hence according marginal distributions are the same, for example elements $z_{i,1}$ and $z_{i,1+D'}$ of all $\mathbf{z}_i$, $i = 1,...,n$, have the same empirical distribution, thus postulation 1 is also fulfilled.

5. THE TOOLS

The method [7] for modeling multivariate distributions and time series seems to work quite well, but programming is complex and has messy details. Therefore it is important to provide tools which can be used easily.

The toolbox consists in an interactive MATLAB program `pwlCopula` and in Java classes, see [14]. Basically, the MATLAB program calculates the copula, provides some statistics and diagrams which serve the purpose to examine the validity and accuracy of this model, and can generate random vectors and time series. The Java classes generate random vectors (methods for time series will be available soon) using a copula and empirical marginal distributions which were calculated with the MATLAB program.

For setting the copula up, the MATLAB program `pwlCopula` takes as input a sample of independent vectors or a time series whose elements can be vectors as well. The copula can be stored in a copula file. During the calculation, the program needs empirical marginal distributions of the sample vectors. They can be stored, too. The user must specify some parameters for the calculations:

- K, integer, determines the accuracy, namely K^D is the number of sub-cubes $\mathscr{S}_{\mathbf{j}}$. This is the granularity of the discretization, the higher, the more accurate. Values between 10 and 4000 were tried.
- $n^{(\text{by } K)}$, integer, defines the sample size $n = n^{(\text{by } K)}K$. Thus K divides n. This is an important condition for the method to be correct, see [7, Theorem 1].
- The name of the file from which the sample is read.
- The window width w for time series.

Using this copula, `pwlCopula` can generate random vectors or a time series which can be strored in a file for later use in a simulation model. For the generation, the user specifies

- The random number stream.
- How many vectors are to be generated.
- The kind of inverse transformation. One method generates only values which occur in the sample. The other, with linear interpolation, also generates intermediate values with linear interpolation of the empirical distribution function.

`pwlCopula` can also generate independent random numbers with the marginal distributions of the vector components of the sample. They can be used to evaluate the benefit of modeling dependency. To this end, simulation results obtained from using sample data as inputs are compared both with simulation results using generated inputs using the independent model, and with simulation results obtained from generated inputs using the dependent model. Thus one can decide which model is valid. This is a highly credible validation method.

`pwlCopula` provides statistics and plot diagrams with the generated random vectors or the time series, and corresponding statistics and diagrams with the given sample. The modeler can compare these in order to obtain insight into validity and accuracy of the copula model.

The statistics deal with the means and the variances in each dimension, and correlations between pairs of dimensions. They are not only calculated for the original sample, but also for the generated vectors. The absolute values of the differences are taken as a measure of accuracy. The difference of means is considered absolute if at least one of the absolute values of the means is less than 10^{-5}. If both values are greater or equal, then the difference of means is considered relative. The difference of two coefficients of variation is considered if both corresponding absolute values of the means are greater than 10^{-5}. If one of them is lower, then the difference of standard deviations is considered. The difference of two correlations is considered if both corresponding standard deviations are greater than 10^{-5}. If one of them is less, then the difference of the covariances is considered. The greatest absolute value of these differences, the *maximum statistical deviation*, is a combined measure of accuracy.

If one replicates the generation process, say r times, the smallest observed maximum statistical deviation and the greatest observed maximum statistical deviation are an (approximate) confidence interval to the confidence level $1-2^{-(r-1)}$, see [13].

Scatter diagrams are provided for visual inspection. In each of them, the value pairs of two different elements of the vectors are plotted as points. Looking at the diagram, one gains insight into the structure of dependency of these two dimensions: There may be regions with no points - obviously the corresponding value pairs do not occur at all, or with rather small probabilities. In the other regions, the points may be variously dense which indicates different probabilities of occurrence in this region.

The modeler can compare corresponding scatter diagrams of the original sample and also of the generated vectors. An indication that the copula model is accurate is when regions without points correspond and when the impression of the frequency is similar.

For time series, correlations are calculated between two vector elements in the same dimension, but at different times $i_1,\ i_2$ with the lag $|i_1 - i_2|$. Again, these correlations are calculated for the original sample and for the generated vectors. The absolute value of their difference is taken as a measure of accuracy. These differences generally increase with a widening lag for lags greater than w, because only dependencies with smaller lags are modeled explicitly by the method in Section 4. Thus it makes no sense to consider only their maximum value, diagrams with differences for different lags are provided instead.

The Java classes are only for the generating of random vectors (and shortly for time series), they implement the same algorithms as the according part of the MATLAB program `pwlCopula`. A copula and empirical marginal distributions which were calculated and stored in a file before with `pwlCopula` must be imported. The Java classes are not interactive, the parameters must be passed to the Java objects via method calls.

6. EMPIRICAL VALIDATION

Many numerical examples indicate validity and good accuracy of the new input model technique with copulas. A couple of them are in [7], and further aspects are discussed in the sequel:

- Nonlinear dependency (Example 1).
- Many dimensions (Example 1).
- What is the influence of the window width (Example 2 and 3)?
- How accurate can long distance dependence be modeled with a small window width (Example 2 and 3)?
- Is it worth modeling dependency, or can one assume independent input (Example 4)?

For all experiments, the CPU times for generation were observed on a Pentium 4 computer of 3 GHz. Sometimes the generation process is replicated 5 times. Then the smallest observed maximum statistical deviation and the greatest observed maximum statistical deviation are an (approximate) confidence interval to the confidence level $1-0.5^4 \approx 0.93$, see [13].

Inverse transformation was done with empirical distribution functions for the integer-valued distributions and with interpolated empirical distribution functions for real-valued distributions.

Example 1 is with random vectors. The sample is artificial, namely realizations from a given distribution where the vector elements are more or less dependent and their dependency is nonlinear. In scatter diagrams, one observes that this nonlinear dependency is covered by the model. High dimensionality is tried, $D = 5,\ 40,\ 100$.

The common distribution is defined as follows: $Z_{1,i} = Y_{1,i}$, $Z_{d,i} = Z_{\lceil d/2\rceil,i}(1 - Z_{\lceil d/2\rceil,i}) + Y_{d,i},\ d = 2,\dots,D,\ i = 1,\dots,n$, where the $Y_{d,i}$ are independent and $U(0,1)$-distributed.

Table 1: Results for Example 1

D	K	n	$n^{(gen)}$	Max.Stat.Diff.	CPU[sec]
5	4000	4000	64000	0.007±0.002	29
40	1000	1000	16000	0.028±0.004	74
100	100	1000	16000	0.031±0.004	210

In Table 1 the model parameters and the results are given. Figure 1 shows scatter diagrams for the vector elements 1 and 2, the left diagram for the sample, the right one for the generated vectors. Their similarity indicate that the dependency structure is modeled accurately.

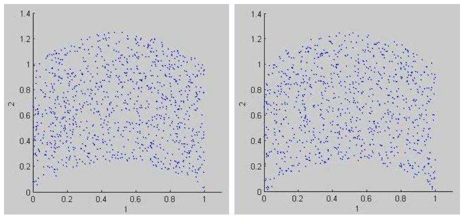

Figure 1. Sample and Generated Points, Dimensions 1 and 2

Example 2 is a time series of scalars, $D' = 1$, also an artificial model. T_i depends explicitly on T_{i-1}, T_{i-2}, and T_{i-3}: $T_i = 0.2T_{i-1} + 0.3T_{i-2} - 0.4T_{i-3} + Y_i$, $i = 4, ..., n0 + n$, where T_1, T_2, T_3, and the Y_i are $U(0,1)$-distributed. The autocorrelations are between -0.3 and 0.3. Modeled window width is w =2 and 4.

The first random variables T_1, T_2, ... are not stationary; this is why the first $n0 = 100$ realizations are skipped, assuming that the stochastic process is then nearly stationary.

Table 2: Results for Example 2

K	1000	1000
w	2	4
n	16000	16000
$n^{(gen)}$	16000	16000
Max.Stat.Diff.	0.016	0.0001
Dev.Autocorr.	0.3	<0.003
CPU[sec]	1.7	14.6

In Table 2 the model parameters and the results are given. For window width $w = 2$, the autocorrelations are modeled poorly for time lags 1,...,200, namely with absolute values of deviations up to nearly 0.3. This is not surprising since the explicit dependence of the time series elements goes back three steps. With width $w = 4$, the maximum statistical deviation is very low and the autocorrelations are also accurate.

Example 3 is a time series. The sample is measured IP-traffic data from [8], inter-arrival times in the first, and packet sizes in the second dimension, $D' = 2$. The marginal distributions are irregular and no suitable standard distributions are available; this is no problem with empirical distributions. The dependence structure is not obvious, however, this is accounted for automatically if the window width is not too small.

In Table 3 the model parameters and the results are given. With window width $w = 2$ in column 1, the results are good, with $w = 4$ the results in column 2 are better, namely the maximum statistical deviations are smaller by one order of magnitude. The deviations of the autocorrelations are even smaller by two orders of magnitude.

The scatter diagrams, figure 3, show a very irregular dependency structure. The comparison of the sample and the generated points indicate good accuracy:

Table 3: Results for Example 3

K	4000	4000	1000	1000
w	2	4	2	4
n	16000	16000	1000	1000
$n^{(gen)}$	16000	16000	16000	16000
Max.Stat.Diff.	0.037 ±0.002	0.001 ±0.004	0.0077 ±0.0009	0.0074 ±0.0008
Dev.Autocorr.	0.4	0.004	0.06	0.06
CPU[sec]	6	13.7	6.2	13.5

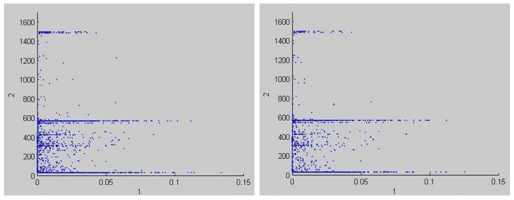

Figure 2. Sample and Generated Points, Dimensions 1 and 2

In the special case where the sample size n and the granularity parameter K are equal in columns 3 and 4, the observed accuracy does not differ significantly for different window widths. This is in line with the remark in Section 5 about the special case where $K = n$.

Example 4 considers the same sample as Example 3, granularity $K = 4000$, window width $w = 4$, but with sample size 60000. Here the benefit of modeling dependency is evaluated by means of three simulations with similar simulation models. They are equal except for different input models:

1. Some performance or reliability measure is evaluated with the sample as input.

2. The same is done for dependent input data which is generated with `pwlCopula`, and the results are compared with those from 1.

3. The same is also done with independent input data which is generated with the empirical marginal distributions, and the results are compared with those from 1.

The simulation model is a queueing system. The service times are commensurate with the packet sizes, service time = packet size / rate. The rate is such that the server utilization is about 53%. The inter arrival times and the packet sizes are taken directly from the sample or from the input model. The performance measures are the number Nbr of packets in the queue, the probability $P26$ that there are less than 26 packets in the waiting queue, and the delay $Delay$ of the packets in the queue.

The simulation with sample data consists in a single run with 60000 arrivals. The simulations with dependently and independently generated arrivals, both consist in five replications with 12000 arrivals each. In this case 95%-confidence intervals are given.

Table 4: Simulation Results for Example 4

Input Model	Nbr	$P26$	$Delay$
Sample	3.97	0.967	0.048
Dependent	4.54 ± 3.57	0.959 ± 0.045	0.048 ± 0.029
Independent	1.25 ± 0.06	1.0 ± 0.00	0.015 ± 0.0004

Table 4 gives an overview of the results: For the dependent input model, the means of the simulated measures are quite near to the simulated means for the simulation with sample data. These means are within the confidence intervals. The model seems to be valid.

For the independent input model, the contrary is true: The means are unacceptably distant, the confidence intervals do not contain the means which were simulated with sample data. This model is invalid.

CONCLUSION

Incorrect simulations may result from poor modeling practices, particularly overlooked dependencies. This is well known, and our Example 4 confirms this once again. There are a couple of modeling techniques for dependent input, but they have several draw-backs: most of them solely consider correlations, or only random vectors with low dimension are feasible, or else they are difficult to apply. The proposed empirical copula approach comprises the complete dependence structure, and is both general and suitable for large dimensions. The proposed tools overcome the method's dependency upon intricate programming. With these ready-to-use programs, input models can be set up, and random variates and time series can be generated immediately without resorting to the complexities of the empirical copula technique.

7. REFERENCES

[1] B. Biller and B. Nelson. Modeling and generating multivariate time-series input processes using a vector autoregressive technique. *ACM Transactions on Modeling and Computer Simulation*, 13:211–237, 2003.

[2] M. Cario and B. Nelson. Autoregressive to anything: Time-series input processes for simulation. *Operations Research Letters*, 19:51–58, 1996.

[3] M. Cario and B. Nelson. Modeling and generating random vectors with arbitrary marginal distributions and correlation matrix. Tech. rep., Northwestern University, Department of Industrial Engineering and Management Sciences, Evanston, Ill, 1997.

[4] S. Ghosh and S. Henderson. Chessboard distributions and random vectors with specified marginals and covariance matrix. *Operations Research*, 50:820–834, 2001.

[5] S. Ghosh and S. Henderson. Properties of the NORTA method in higher dimensions. In E. Yücesan, C. Chen, J. Snowdon, and J. Charnes, editors, *Winter Simulation Conference Proceedings*, pages 263–269. IEEE, Piscataway, N.J., 2002.

[6] S. Ghosh and S. Henderson. Patchwork distributions. Contributed to Festschrift for George Fishman, 2008.

[7] J.Ch.Strelen and F. Nassaj. Analysis and generation of random vectors with copulas. In S.G.Henderson, B.Biller, M.-H.Hsieh, J.Shortle, J.D.Tew, and R.R.Barton, editors, *Proceedings of the 2007 Winter Simulation Conference*, pages 488–496. Omnipress, Washington DC., 2007. http://www.informs-sim.org/wsc07papers/058.pdf.

[8] A. Klemm, C. Lindemann, and M. Lohmann. Traffic modeling of IP networks using the batch Markovian arrival process. In *Proceedings of the 12th Int. Conf. on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, pages 92–110. Springer, London, 2002. LNCS 2324.

[9] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, third edition, 2000.

[10] B. Melamed. The empirical TES methodology: Modeling empirical time series. *J. of Applied Mathematics and Stochastic Analysis*, 10(4):333–353, 1997.

[11] F. Nassaj and J.Ch.Strelen. Dependence input modeling with the help of non-Gaussian AR models and genetic algorithms. In J. Teixera and A.E.Carvalho-Brito, editors, *Modelling and Simulation 2005, Proceedings of the European Simulation and Modelling Conference, Porto, 2005*, pages 146–153. Eurosis-ETI, Ghent, 2005.

[12] R. Nelsen. *An introduction to copulas*. Springer, New York, 1998.

[13] J. C. Strelen. The accuracy of a new confidence interval method. In R. Ingalls, M. Rossetti, J. Smith, and B. Peters, editors, *Proceedings of the 2004 Winter Simulation Conference*, pages 654–662. Omnipress, Washington DC., 2004. www.informs-sim.org/wsc04papers/079.pdf.

[14] J. C. Strelen. Generating random vectors and time series with copulas - the tools. http://web.cs.uni-bonn.de/IV/strelen/Algorithmen, 2008.

Appendix

Table 5: Access Hash Tables

Address	Triple Table Pointers		Sub-Cube Indices			Coll. Ptr.
$1 \rightarrow$	…	…	…	…	…	…
$h(d, j_1, ..., j_{d-1}) \searrow$ $h(d, j'_1, ..., j'_{d-1}) \rightarrow$ $h(d, j''_1, ..., j''_{d-1}) \nearrow$	*beg*	*end*	j_1	…	j_{d-1}	p
$m' \rightarrow$	…	…	…	…	…	…
$m'+1 \rightarrow$ (Overflow Section)	…	…	…	…	…	…
$p \rightarrow$	beg'	end'	j'_1	…	j'_{d-1}	p'
	…	…	…	…	…	…
$p' \rightarrow$	beg''	end''	j''_1	…	j''_{d-1}	0
$m = m' + n \rightarrow$	…	…	…	…	…	…

Table 6: Triple Tables

Address	$j_d =$	f-value	s-value
$1 \rightarrow$	…	…	…
$beg(j_1, ..., j_{d-1}) \rightarrow$ $end(j_1, ..., j_{d-1}) \rightarrow$	$j_{d,1}$ $j_{d,2}$ … $j_{d,end-beg+1}$	$f^{(d)}_{j_1,...,j_d}$ $f^{(d)}_{j_1,...,j_d}$ …	$s^{(d)}_{j_1,...,j_d}$ $s^{(d)}_{j_1,...,j_d}$, …
	…	…	…
$beg' \rightarrow$ $end' \rightarrow$	…	…	…
	…	…	…
$beg'' \rightarrow$ $end'' \rightarrow$	…	…	…
$n \rightarrow$	…	…	…