Journal of Uncertain Systems
Vol.2, No.3, pp.192-201, 2008
Online at: www.jus.org.uk

Generating Correlation Matrices for Normal Random Vectors in NORTA Algorithm Using Artificial Neural Networks

Seyed Taghi Akhavan Niaki, Babak Abbasi*
Department of Industrial Engineering, Sharif University of Technology, Tehran, Iran

Received 11 December 2007; Accepted 1 March 2008

Abstract

Generating multivariate random vectors is a crucial part of the input analysis involved in discrete-event stochastic simulation modeling of multivariate systems. The NORmal-To-Anything (NORTA) algorithm, in which generating the correlation matrices of normal random vectors is the most important task, is one of the most efficient methods in this area. In this algorithm, we need to solve the so-called correlation-matching problem in which some complicated equations that are defined to obtain the correlation matrix of normal random variables need to be solved. Many researchers have tried to solve these equations by three general approaches of (1) solving nonlinear equations analytically, (2) solving equations numerically, and (3) solving equations by simulation. This paper suggests the use of artificial neural networks, called Perceptron, to solve the corresponding problem. Using three simulation experiments, the applicability of the proposed methodology is described and the results obtained from the proposed method to the ones from solving the equations numerically are compared. The results of the simulation experiments show that the proposed method works well.

Keywords: random vectors generation, correlation matrix, neural networks, multivariate distribution, NORTA

1 Introduction and Literature Review

In many practical settings, there is a basic need to capture the dependence between random variables that serve as primitive inputs to stochastic models. For example, due to special characteristics of a particular product, the processing times at a series of k machining stations may be dependent. Similarly, the service times for a single customer at the order desk, cashier and loading dock of a store may be dependent. In financial engineering, Das *et al.* claim that the risk profile of credit portfolios cannot be understood if we ignore the correlations [6]. Further applications have been reported in cost analysis [21], market size and selling price in capital return models [15], correlated attributes in decision analysis [5], decision and risk analysis [4], and generation of test problems [13]. In all of these situations, if we apply a simulation study to analyze the systems, we are in need of an efficient method to generate samples from a joint distribution of some correlated random variables.

There are numerous methods available in the literature to represent and generate random vectors with dependent components and marginal distributions from a common family. In general, we can classify these methods in three categories: (1) Analytical approaches that employ conditional distributions, (2) Numerical procedures using the acceptance/rejection methods, and (3) Simulation approaches that apply the partially specified property in transformation procedures.

In the first category Johnson [16] generates from marginal distribution and generate from the conditional distribution for k=2, …, n. This approach may be difficult to apply because conditional distributions usually are not easy to derive except in special cases. In addition, in this approach, we assume that the joint distribution of the random vectors is known, the case does not often happen in practice. In other analytical approaches Moonan [23] proposes the generation of normal random vectors based on linear transformation of a set of independent standard normal random variables, Ronning [26], Schmeiser and Lal [27] and Lewis [18] propose generation models for the nonnegative correlated multivariate gamma, bivariate gamma, and negatively correlated multivariate gamma distributions, respectively. Moreover, Johnson [16] developed a multivariate lognormal generation method, Parrish

* Corresponding author. Email: B.Abbasi@Gmail.com (B. Abbasi)

[25] proposes a multivariate Pearson generation method, and Johnson [16] and Stanfield [31] develop a Multivariate Johnson generation procedure.

In the second category, Johnson [16] and Devroye [7] use joint density functions to generate random vectors by the acceptance/rejection approach. In these methods, we select one of the joint probability density functions that dominate the original joint probability density function. Then we generate a random vector based on the selected joint density function. Finally, this random vector is accepted or rejected. Gilks and Wild [11], Hormann [14], and Leydold [19] suggest a transformed density rejection method to construct the dominating density function.

Most of the proposed methods of random vector generation in the above two categories often have constraints on the size of the random vector and many of them are applicable only for bivariate distributions. Moreover, they are only utilizable to generate random vectors whose variables have a common distribution. For example, the method proposed by Johnson [16] generates random vectors of dependent variables with a common probability distribution.

In the third category, the analyst is required to give up specifying a full joint probability distribution for the random vector to be sampled in favor of an appropriate set of marginal probability distributions and a correlation matrix (the partial specification). The NORmal-To-Anything (NORTA) transformation detailed by Cairo and Nelson [2], which is based on work by Marida [22] and Li and Hammond [20], is an example of the research in this category. Moreover, Song and Hsiao [29] and Song *et al.* [30] apply a similar concept to generate time series random variables.

For those applications, where the partially specified approaches are acceptable, the payoffs can be large. The NORTA transformation demonstrates that we can obtain samples from the partially specified distribution by transforming the elements of a sample from a multivariate standard normal distribution according to the appropriate desired marginal distribution, where the correlations of the elements of the deriving normally distributed random vector are set to generate the desired correlations in the transformed random vector. Setting the correlation matrix appropriately amounts to solving a number of one-dimensional root finding problems corresponding to each desired pair-wise correlation value, each of which can be solved by bisection. Solving some complex nonlinear system of equations is the most important problem in partially specified approaches.

In NORTA algorithm, we first generate a *k*-dimensional standard normal random vector and then we transform it into a vector of uniform random vectors. At the end by the inverse transformation technique, we transform back the uniform random vector into a random vector with target marginals. The initialization step in the NORTA algorithm requires finding the correlation matrix of the normal random vector such that it guarantees the last random vectors generated have specific correlation matrix (see [16, 23, and 17]). To reach this goal a number of one-dimensional simultaneous equations must be solved, which usually is analytically hard to do. Moreover, there are two complications in NORTA algorithm. The first is that a desired pair-wise correlation may not be feasible; that is, there may not exist a pair-wise correlation for the driving multivariate normal random vector such that the corresponding transformed elements have the desired correlation value. The second, that is more serious, is that even if all desired correlation values are feasible, the full correlation matrix for the driving multivariate normal random vector is not positive definite. This becomes an issue where the dimensionality of the random vector increases. Ghosh and Henderson [9 and 10] present the behavior of the NORTA method in higher dimensional cases and propose a procedure to modify NORTA in these cases. In addition, they studied the ability of the NORTA procedure to match any feasible correlation matrix for a given set of marginals. Stanhope [32] develops a more general framework for partially specified random vector generation and proposes several alternatives to NORTA algorithm.

In this paper, we focus on the NORTA algorithm and propose a model to generate the correlation matrices of normal random vectors that are required in NORTA. We apply a Perceptron Neural Network (PNN) to estimate the correlation matrices of normal random vectors, ignoring the analytically complicated equations in NORTA algorithm.

In Section 2, we briefly explain the NORTA algorithm and the approaches to solve some complicated equations. The concept of neural networks, its training phase, and the back propagation algorithm all come in Section 3. Section 4 contains the proposed methodology to generate the correlation matrices of multi-normal random vectors used in the NORTA algorithm. To better describe and understand the proposed method, in Section 5 we present three numerical examples. Then in Section 6, we compare the results obtained from the proposed method to the ones from the research by Cairo and Nelson [2]. The conclusion and recommendations for future research come in Section 7.

2 The NORTA Algorithm

The goal of the NORTA algorithm is to generate a *k*-dimensional random vector $\mathbf{X} = [x_1, x_2, ..., x_k]^T$ with the following properties: $x_i \sim F_{X_i}, i = 1, 2, ..., k$, where F_{X_i} is an arbitrary cumulative distribution function (cdf) and $\text{Corr}[\mathbf{X}] = \boldsymbol{\Sigma}_{\mathbf{X}}$, where $\boldsymbol{\Sigma}_{\mathbf{X}}$ is given.

We generate the vector X by a transformation of a k-dimensional standard multivariate normal (MVN) vector $\mathbf{Z}=(z_1,z_2,...,z_k)^T$ with correlation matrix $\mathbf{\Sigma_Z}$. Specifically, the NORTA vector $\mathbf{X}$ is:

$$\mathbf{X}=\left(F_{x_1}^{-1}[\Phi(z_1)],F_{x_2}^{-1}[\Phi(z_2)],\cdots,F_{x_k}^{-1}[\Phi(z_k)]\right)^T \tag{1}$$

where Φ is the univariate standard normal cdf and $F_x^{-1}(u)\equiv\inf\{x:F_x(x)\geq u\}$ denotes the inverse cdf. We note that since the exact value of $F_x^{-1}(\Phi(z))$ for each distribution (especially for discrete distributions) may not be known, the infimum value of x that satisfies $F_x(x)\geq\Phi(z)$ is selected for x.

The transformation $F_{x_i}^{-1}[\Phi(.)]$ ensures that x_i has the desired marginal distribution F_{x_i}. Therefore, the general problem is to select the correlation matrix $\mathbf{\Sigma_Z}$ that gives the desired correlation matrix $\mathbf{\Sigma_X}$. In fact, the NORTA transformation can be viewed as a two-step process. In the first step, a multivariate normal vector $\mathbf{Z}$ is transformed into a multivariate uniform vector $\mathbf{U}$. The second step involves transforming the multivariate uniform vector $\mathbf{U}$ into the desired vector $\mathbf{X}$. Then, the joint distribution of $\mathbf{U}$ is known as copula and any joint distribution has a representation as a transformation of a copula [28].

2.1 Relationship Between $\mathbf{\Sigma_Z}$ and $\mathbf{\Sigma_X}$

Each element of $\mathbf{\Sigma_Z}$ such as $\rho_z(i,j),i\neq j,$ shows the correlation between z_i and z_j. Similarly, $\rho_x(i,j),i\neq j,$ denotes the correlation between x_i and x_j in $\mathbf{\Sigma_X}$. That is,

$$\rho_x(i,j)=Corr[x_i,x_j]=Corr\left\{F_{x_i}^{-1}[\Phi(z_i)],F_{x_j}^{-1}[\Phi(z_j)]\right\},\quad i\neq j. \tag{2}$$

Since

$$Corr(x_i,x_j)=\frac{E(x_ix_j)-E(x_i)E(x_j)}{\sqrt{Var(x_i)Var(x_j)}}, \tag{3}$$

$E(x_i)$, $E(x_j)$, $Var(x_i)$ and $Var(x_j)$ are fixed by F_{x_i} and F_{x_j}, and (z_i,z_j) has a standard bivariate normal distribution with correlation $Corr(z_i,z_j)=\rho_z(i,j)$, we have

$$E(x_ix_j)=E\left\{F_{x_i}^{-1}[\Phi(z_i)]F_{x_j}^{-1}[\Phi(z_j)]\right\}=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}F_{x_i}^{-1}[\Phi(z_i)]F_{x_j}^{-1}[\Phi(z_j)]\varphi_{\rho_z(i,j)}(z_i,z_j)dz_idz_j \tag{4}$$

where $\varphi_{\rho_z(i,j)}(z_i,z_j)$ is the standard bivariate normal probability density function (pdf) with correlation $\rho_z(i,j)$.

From equation (4) we see that the correlation between x_i and x_j is a function of the correlation between only z_i and z_j. We denote this function by $c_{ij}[\rho_z(i,j)]$. In other words, $\rho_x(i,j)=c_{ij}\left[\rho_z(i,j)\right]$.

In order to generate a k-dimensional random vector by NORTA algorithm we need to solve equation (4) for each pair of the variables. Hence, we need to solve $k(k-1)/2$ complicated equations that for many marginal distributions are usually unsolvable by analytical methods. Cario and Nelson [2] presented some theorems and propositions that describe the property of $\mathbf{\Sigma_Z}$ and are helpful in solving equation (4).

In short, to generate $\mathbf{\Sigma_Z}$ there are three basic approaches as follow:

1) The analytical approach that works for some special cases such as uniform random vectors is difficult to apply because the conditional distributions are not easy to obtain in most cases. Chen [3] and Hull [15] applied conditional distribution (assumed known) to solve equation (4) for multivariate normal distributions.

2) In the numerical approach, one employs numerical root finding methods to solve $k(k-1)/2$ equations. In this approach, the double integral function values of the form (4) are evaluated by numerical integration methods. Li and Hammond [20] and Cario and Nelson [2] used Newton's method and Yen [33] applied the efficient Gaussian-Quadrature integration and Newton's methods. In these methods, the computational time increase quadratically with k. Furthermore, the equispaced integration methods may be inefficient, and the Gaussian quadrature methods may be inaccurate (Chen [3]).

3) In the simulation approach, for any set of the root candidates, first the NORTA algorithm is applied to generate m random vectors. Then, the correlations of the generated observations are estimated and checked to reach the required correlation matrix. Chen [3] employed this approach to solve the $k(k-1)/2$ equations in (4) by treating

it as a stochastic root-finding problem, solving equations using only the estimates of the function values. Yen [33] mentioned that the disadvantage of this approach is that the computation time is usually longer than the numerical approach.

To avoid the problems involved in solving equations (4), in the following section, we use the concept of function fitting to generate Σ_Z using artificial neural networks.

3 Multilayer Perceptrons Neural Networks (PNN)

This is perhaps the most popular network architecture in use today, which is discussed at length in most neural network textbooks (e.g., Bishop, [1]). In this type of network, we arrange the units in a layered feed forward topology, where the units each perform a biased weighted sum of their inputs and pass this activation level through a transfer function to produce their output. The network thus has a simple interpretation as a form of input-output model, with the weights and thresholds (biases) as the free parameters of the model. Figure 1 shows the topology of PNN with one hidden layer.

Such networks can model functions of almost arbitrary complexity, with the number of layers, and the number of units in each layer, determining the function's complexity. Important issues in Multilayer Perceptrons (MLP) design include specification of the number of hidden layers and the number of units in these layers (see [1, 12]). The number of input and output units is defined by the problem (there may be some uncertainty about precisely which inputs to use). The number of hidden units to use is far from clear. A good starting point is to use one hidden layer, and the number of units in hidden layer is traded.

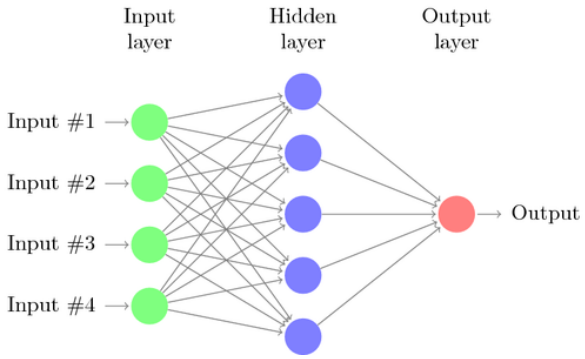

Figure 1: Topology of PNN with a hidden layer

3.1 Training Multilayer Perceptrons

Once we select the number of layers and the number of units in each layer, the network's weights and thresholds must be set to minimize the prediction error made by the network.

This is the role of the training algorithms. In order to minimize the error, we use the historical cases that the user has gathered to adjust the weights and thresholds automatically. This process is equivalent to fitting the model represented by the network to the training data available. The error of a particular configuration of the network can be determined by running all the training cases through the network, comparing the actual output generated with the desired (target) outputs. Then, by an error function, we combine the differences together to get the network error. The most common error functions used in the literature are the sum of squared error (SSE), in which we square and sum together the individual errors of output units in each case.

3.2 The Back Propagation Algorithm

The best-known example of a neural network-training algorithm is the back propagation algorithm (see [24, 12 and 8]). Modern second-order algorithms such as conjugate gradient descent and Levenberg-Marquardt (see [1]) are substantially faster, e.g., a faster order of magnitude, for many problems, but back propagation still has advantages in

some circumstances, and is the easiest algorithm to understand. We will now introduce and use this algorithm. There are also heuristic modifications of back propagation, which work well for some problem domains.

In back propagation, the gradient vector of the error surface is calculated. This vector points along the line of steepest descent from the current point, so we know that if we move along it a "short" distance, we will decrease the error. A sequence of such moves, slowing as we near the bottom, will eventually find a minimum of some sort. The difficult part is to decide how large the steps should be.

Large steps may converge more quickly, but may also overstep the solution or go off in the wrong direction (if the error surface is very eccentric). A classic example of this in neural network training is where the algorithm progresses very slowly along a steep, narrow, valley, bouncing from one side across to the other. In contrast, although very small steps may go in the correct direction, they also require a large number of iterations. In practice, the step size is proportional to the slope (so that the algorithms settle down in a minimum) and to a special constant: the learning rate. The correct setting for the learning rate is application-dependent, and is typically chosen by experiment; it may also be time varying, getting smaller as the algorithm progresses.

The algorithm therefore progresses iteratively, through a number of epochs. On each epoch, we submit the training cases in turn to the network and target actual outputs; then, compare and calculate the error. This error, together with the error surface gradient, is used to adjust the weights, and then the process repeats. The initial network configuration is random and training stops when a given number of epochs elapse, or when the error reaches an acceptable level, or when the error stops improving.

4 Using ANN to Generate Σ_Z

Generating $\rho_z(i, j)$ for each pair of random variables in $\mathbf{\Sigma_Z}$ matrix is very troublesome. Our suggestion is to employ an ANN that must be first trained with marginal distributions of the random variables and then be used to generate the $\rho_z(i, j)$ values between each pairs of the variables. In fact, we need to employ one network for each pair of the variables that have different marginal distributions. In other words, on the one hand, if all of the variables posses a common distribution, to generate $\mathbf{\Sigma_Z}$ one trained ANN is adequate. On the other hand, in a three dimensional random vector for example, if every pair of the random variables have different marginal probability distribution functions, we need to employ three networks.

For one pair of the random variables assume x_i and x_j have c.d.f. $F_i(x_i)$ and $F_j(x_j)$ respectively, and the correlation between them is $\rho_x(i, j)$.

To design a proper ANN, we use the Perceptron neural network. This network is one of the most powerful ANN in function fitting. To apply PNN we take the following steps: (1) Establish the network architecture; (2) Provide the training data; (3) Train the network; (4) Apply the train network. These steps are discussed in the following subsections.

4.1 Establishing the Network Architecture

We design a PNN with one neuron in its input layer to represent the variables of whose we need to generate random variates. The number of neurons in the output layer is one, which models the random variates of the variables generated by NORTA.

4.2 Data Training

To generate data for training purposes, we run NORTA in backward order. In other words, first we generate random variates from the joint probability distribution with known correlation between the random variables using NORTA. Since the training data sets should cover all domains of the input and the output variables, we select 100 uniformly distributed random values between -1 and 1 as the correlations between the variables ($\rho_z(i, j)$). Then, using the method of moments we estimate the correlation between the generated varaites ($\hat{\rho}_x$). These correlation values are set as the input of the network. The network should be trained such that it returns output values close to the uniformly

generated correlations between -1 and 1.

The following training-data-generation algorithm shows the steps involved to provide 100 pairs of input-output data sets required to train the PNN network.

For s=1:100 $\rho_z(s)$ = *Uniformly random variates between [-1,1];*

Follow general steps of NORTA algorithm for $\rho_z(i,j) = \rho_z(s)$ *1000 times.*

Based on the generated data, compute $\hat{\rho}_x(s)$ *(an estimate of correlation between* x_i, x_j *.)*

Input = [Input $\hat{\rho}_x(s)$ *] Output=[Output* $\rho_z(s)$ *]*

End for

4.3 Training the Network

After generating the training data sets, we design PNN by changing the number of neurons in the hidden layer; trying to minimize SSE by back propagation algorithm. We stop the algorithm while SSE reaches to a desired small value. Then, we apply the trained PNN to generate $\rho_z(i,j)$ from $\rho_x(i,j)$. Figure 2 depicts the proposed algorithm to generate random vectors using both NORTA algorithm and the neural network approach.

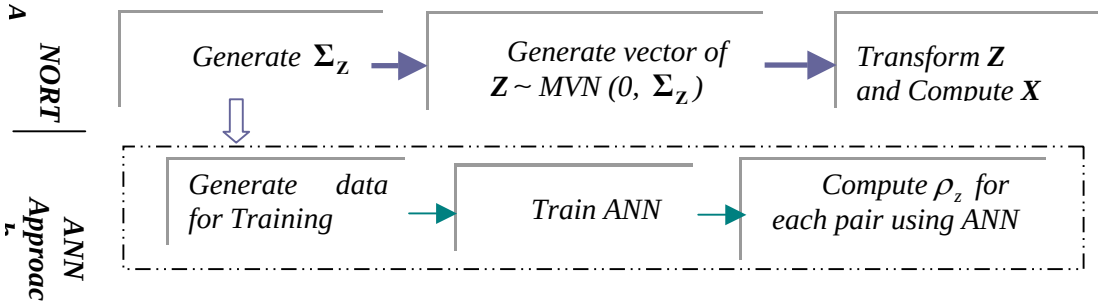

Figure 2: NORTA and PNN model to generate random vectors

5 Performance Evaluation

In order to better understand the method and demonstrate its efficiency we state three numerical examples.

Example 1: As a simple case assume we want to generate a bivariate binomial distribution with parameters $(n_1 = 20, p_1 = 0.2)$ and $(n_2 = 30, p_2 = 0.15)$ respectively, with the correlation of $\rho_x(1,2) = 0.25$, i.e.

$$X \sim MBinomial([20;30],[0.2;0.15];0.25).$$

We coded this algorithm along with the NORTA algorithm and the PNN network all in MATLAB 7. In the first step of the proposed method, we generate input data sets for training purposes. For each of the 100 random numbers generated from a Uniform distribution between -1 and 1 as the correlation values (ρ_z), we generate 1000 bivariate binomial random vectors with parameters of $([n_1 = 20; n_2 = 30],[p_1 = 0.2; p_2 = 0.15])$ by NORTA algorithm. Then using the generated data and applying the moments method we estimate 100 coefficient of correlations of the variables, $\hat{\rho}_x(1,2)$ and set them to be the inputs of the PNN network. Next, we design a PNN network with 11 neurons in its hidden layer by trial and error. In the training phase of this network, we set the target values of the network to be the corresponding uniformly distributed random values. The designed network reaches to SSE=0.0003 after 2525 epochs. Figure 3 shows the last 25 epochs of the training process in which the value of $\hat{\rho}_x(1,2)$ becomes 0.2429. Now, we are ready to apply the trained network to generate $\rho_Z(1,2)$ for the NORTA algorithm.

In order to evaluate the performance of the trained network we generate 5000 random vectors using NORTA algorithm with $\rho_Z(1,2) = 0.2429$ and compute $\rho_x(1,2)$. The result of this process is $\hat{\rho}_x(1,2) = 0.2466$ and $\boldsymbol{\mu} = [3.9742, 4.5]$ which is very close to their intended values of $\rho_x(1,2) = 0.25$ and $[n_1 p_1 = 4.0, n_2 p_2 = 4.5]$. Figure 4 shows the joint probability distribution of the data generated for this example. In addition, Figure 5 shows the marginal distributions of the generated data. From Figure 5 we see that the shapes of the marginal distributions of the generated data are close to binomial distribution.

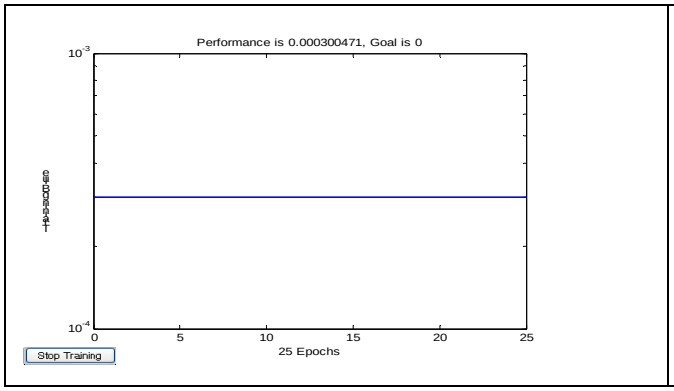

Figure 3: The last 25 epochs of the training process

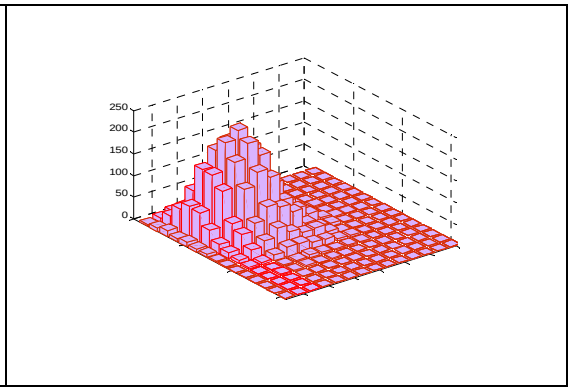

Figure 4: The joint probability distribution of the data generated in Example 1

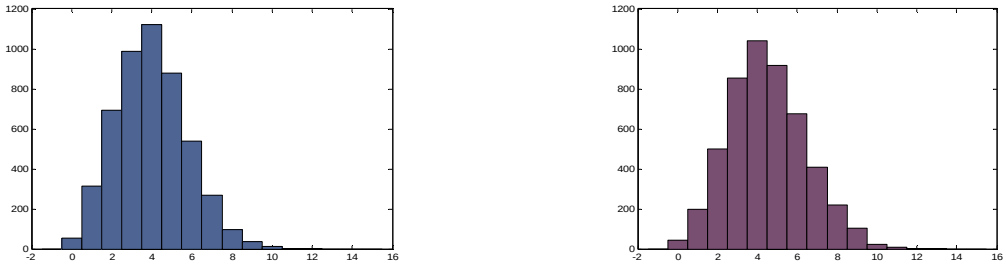

Figure 5: The marginal distributions of the data generated in Example 1

Example 2: Assume we want to generate a three dimensional random vector with the variables having marginal Poisson distribution with parameters $\lambda_1 = 2, \lambda_2 = 3, \lambda_3 = 1$ and the correlation matrix of

$$\boldsymbol{\Sigma}_{\mathbf{X}} = \begin{pmatrix} 1 & 0.5 & -.2 \\ 0.5 & 1 & -0.5 \\ -0.2 & -0.5 & 1 \end{pmatrix}.$$

Since the parameters of the distributions have different values, we need to employ three PNN networks to estimate the coefficient of correlations between (Z_1, Z_2), (Z_1, Z_3), and (Z_2, Z_3) in $\boldsymbol{\Sigma}_{\mathbf{Z}}$. In order to do this, we use the training-data-generating algorithm and design PNN networks with nine neurons in their hidden layer for each case. In the training phase, the first network reaches the value of SSE=0.0003 after 5525 epochs. These figures for the second and the third networks are 0.0006 in 6050 epochs and 0.0005 in 8025 epochs, respectively. The results of the training phase show a correlation matrix of

$$\hat{\boldsymbol{\Sigma}}_{\mathbf{Z}} = \begin{pmatrix} 1 & 0.5333 & -0.2281 \\ 0.5333 & 1 & -0.5943 \\ -0.2281 & -0.5943 & 1 \end{pmatrix}.$$

In order to evaluate the performance of the trained networks, we generate 5000 random vectors by NORTA algorithm using the estimated $\hat{\boldsymbol{\Sigma}}_{\mathbf{Z}}$ and generate $\hat{\boldsymbol{\Sigma}}_{\mathbf{X}}$ as

$$\hat{\boldsymbol{\Sigma}}_{\mathbf{X}} = \begin{pmatrix} 1 & 0.5099 & -0.2119 \\ 0.5099 & 1 & -0.5266 \\ -0.2119 & -0.5266 & 1 \end{pmatrix}.$$

Then, by the moments method we estimate the parameter vector as $\hat{\lambda} = [2.0024, 3.0142, 0.9882]$, which is very close to its intended one. Figure 6 shows the marginal distribution of the generated data. From Figure 6 we see that the shapes of the marginal distributions of the generated data are close to the shape of Poisson distributions.

Example 3: As a third numerical example, consider a mixed random vector containing discrete random variable $X_1 \sim U(1,10)$ and continuous random variable $X_2 \sim Exponential(10)$ in which their coefficient of correlation is -0.5. In this case, solving equation (4) in NORTA algorithm even with numerical methods becomes very cumbersome because we need to use both the integral and the summation operators. However, in the proposed method we can design a PNN network with nine neurons in its hidden layer by trial and error. Then, we generate the training data sets by the training-data-generating algorithm. To do this, for each of the 100 random numbers generated from a uniform

distribution between -1 and 1 as the correlation values, we generate 1000 random vectors with the given parameters by NORTA algorithm. Then applying the moments method we estimate 100 coefficient of correlations of the variables, $\hat{\rho}_x(1,2)$, using the generated data and then set them to be the inputs of the PNN network. In the training phase of this network, we set the target values of the network to be the corresponding uniformly distributed random values. The designed network reaches to SSE=0.0005 after 6525 epochs. At this stage, when we input -0.5 as the coefficient of correlation value to the trained network, the output becomes -0.5573.

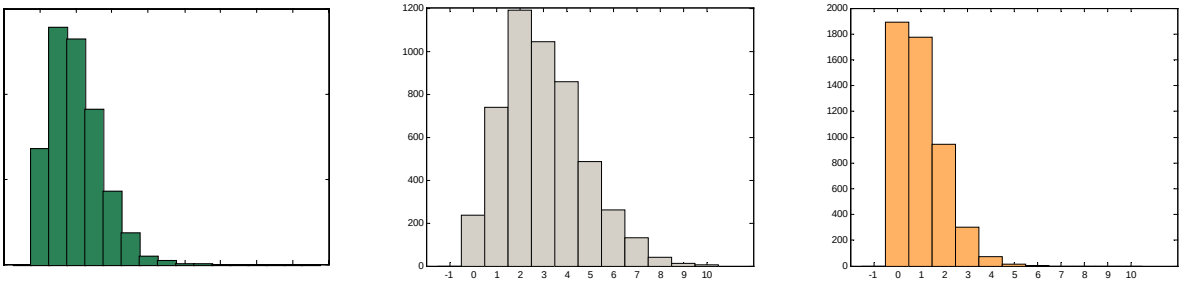

Figure 6: The marginal distributions of the generated data in Example 2

In order to evaluate the performance of the trained network we generate 5000 random vectors using NORTA algorithm with $\rho_Z(1,2) = -0.5573$ and compute $\rho_x(1,2)$. The result of this process is $\hat{\rho}_x(1,2) = -0.4968$ which shows a difference of 0.0032 from the target value. This difference is due to errors such as the error in estimating the coefficient of correlation and the error in generating normal random vectors in the first step of the NORTA algorithm. In order to reduce this difference, in the training phase of the network, we may use 5000 data sets instead of 100 to estimate the coefficient of correlation. Although this will certainly reduce the difference, it will result in higher amounts of running time. For 5000 generated random vectors of this example, the SSE of the PNN network becomes 0.00009 after 7100 epochs in which the estimated coefficient of correlation is -0.5719 for the given input of -0.5. Then $\hat{\rho}_x(1,2)$ becomes 0.5024 for the 5000 testing data sets, which is very close to its intended value of -0.5.

6 A Comparison Study

Cario and Nelson's [2] presented three numerical examples in which they obtained the correlation matrices needed for NORTA algorithm by numerically solving the equation (4) using Newton's Method. We compare the estimated correlation matrices from the proposed method to the ones generated in their examples. In order to do this we generate the correlation matrices of normal random vectors using both methods (PNN and Newton). Then, we use these matrices to generate 5000 random vectors by the NORTA algorithm. Finally, we estimate the correlation matrices and compare the results. In their first numerical example, all of the random variables of a four-dimensional random vector come from a Gamma distribution with parameters of $\alpha = 14.4$ and $\beta = 0.03424$, in which the correlation matrix is given in the third column of Table 1. In the second example, we want to generate three-dimensional random vectors in which the variables all share the same Binomial distribution with parameters as $n = 3$ and $p = 0.5$ and correlation matrix given in the third column of Table 1. The third example is the same as Example 3 presented in 5.3.

In Table 1, the second and the third columns show the marginal probability distributions of the variables in the random vectors and the correlation matrices of the random vectors, respectively. The fourth column shows the generated correlation matrices using the Newton's method for the NORTA algorithm. Column 5 shows the total number of simultaneous equations which we need to solve in the Newton's method. We note that in a given correlation matrix, for every coefficient of correlation, which is different from the other ones, we need to solve one equation in equation (4) numerically. The sixth column shows the generated correlation matrices of the proposed method for the NORTA algorithm. Column 7 shows the number of required PNN networks to estimate the correlation matrices. Note that we need to employ one network for each pair of variables that have different marginal distributions. For example, we need to train only one network in the third example because there is only one marginal probability distribution in this example. In column 8, we have the estimated correlation matrix of 5000 random vectors generated by the NORTA algorithm using the Newton's method. In column 9, the estimated correlation matrix of 5000 generated random vectors by the NORTA algorithm using the proposed method is shown. Finally, column 10 shows the sum of squared differences (SSE) between the required figures in a given correlation matrices and the ones generated by the Newton's and the proposed method.

Table 1: The results obtained from a comparison study

N0.	Marginal Probability Distributions	Σ_x	Σ_z (by Newton's method)	# of equations that must be solved	Σ_z (by proposed method)	# of PNNs that must be trained
1	$x_i \sim \Gamma(14.4, 0.03424)$ $i = 1,2,3,4$	$\begin{pmatrix} 1 & 0.7 & 0.5 & -0.9 \\ & 1 & 0.7 & -0.6 \\ & & 1 & -0.3 \\ & & & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.7040 & 0.5040 & -0.9200 \\ & 1 & 0.7040 & -0.6160 \\ & & 1 & -0.3040 \\ & & & 1 \end{pmatrix}$	5	$\begin{pmatrix} 1 & 0.7008 & 0.5052 & -0.9289 \\ & 1 & 0.7008 & -0.6133 \\ & & 1 & -0.3016 \\ & & & 1 \end{pmatrix}$	1
2	$x_i \sim Binomial(3, 0.5)$ $i = 1,2,3$	$\begin{pmatrix} 1 & 0.2 & -0.8 \\ & 1 & 0.2 \\ & & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.2288 & -0.8960 \\ & 1 & 0.2288 \\ & & 1 \end{pmatrix}$	2	$\begin{pmatrix} 1 & 0.2304 & -0.8981 \\ & 1 & 0.2304 \\ & & 1 \end{pmatrix}$	1
3	$x_1 \sim Discrete\ U(1,10)$ $x_2 \sim Exp.(10)$	$\begin{pmatrix} 1 & -0.5 \\ & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & -0.5760 \\ & 1 \end{pmatrix}$	1	$\begin{pmatrix} 1 & -0.5719 \\ & 1 \end{pmatrix}$	1

Table 1: Continued

No.	$\hat{\Sigma}_x$ of 5000 data sets generated using Σ_z in Newton's method	$\hat{\Sigma}_x$ of 5000 data generated using Σ_z in the proposed method	SSE	
			Newton's method	Proposed method
1	$\begin{pmatrix} 1 & 0.7087 & 0.5175 & -0.8902 \\ & 1 & 0.7072 & -0.6070 \\ & & 1 & -0.3151 \\ & & & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.7005 & 0.5160 & -0.9008 \\ & 1 & 0.7013 & -0.6015 \\ & & 1 & -0.3118 \\ & & & 1 \end{pmatrix}$	0.000807	0.000400
2	$\begin{pmatrix} 1 & 0.2116 & -0.7959 \\ & 1 & 0.1964 \\ & & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.2061 & -0.8017 \\ & 1 & 0.1930 \\ & & 1 \end{pmatrix}$	0.000164	0.000080
3	$\begin{pmatrix} 1 & -0.5037 \\ & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & -0.5024 \\ & 1 \end{pmatrix}$	0.000013	0.000005

The results of Table 1 show that not only the proposed method works better than the Newton's method, but also there is much less efforts involved.

7 Conclusion and Recommendations for Future Research

In this paper, we proposed a new method to generate random vectors with arbitrary marginal distributions and correlation matrices. We focused on the NORTA algorithm and employed artificial neural network estimation of the correlation matrices of normal random vectors, ignoring the analytically complicated equations in NORTA algorithm. We can apply the new method to any type of marginal distribution, and it is very simple to code for any random vectors even if the vectors contain variables of both discrete and continuous types with different marginal distributions. The results of a comparison study and the applications of the proposed method on three numerical examples are encouraging.

For the future research we recommend extending the use of neural network estimation methodology to the cases in which one desires to generate multivariate random vectors with a given auto-correlation matrix directly.

References

[1] Bishop, C., *Neural Networks for Pattern Recognition*, Oxford University Press, U.K, 1995.
[2] Cairo, M.C., and B.L. Nelson, Modeling and Generating Random Vectors with Arbitrary Marginal Distributions and Correlation Matrix, *Tech. Rep.*, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL, 1997.
[3] Chen, H., Initialization for NORTA: Generation of random vectors with specified marginal and correlations, *INFORMS Journal on Computing*, vol.13, pp.312-331, 2001.
[4] Clemen, R.T., and T. Reilly, Correlations and copulas for decision and risk analysis, *Management Science*, vol.45, pp.208–224, 1999.
[5] Corner, J.I., and C.W. Kirkwood, The magnitude of errors in proximal multiattribute decision analysis with probabilistically dependent attributes, *Management Science*, vol.42, pp.1033-1042, 1996.
[6] Das, S.R., G. Fong, and G. Geng, Impact of correlated default risk on credit portfolios, *Journal of Fixed Income*, vol.11, pp.9-19, 2001.
[7] Devroye, L., *Non-uniform Random Variate Generation*, Springer-Verlag, New York, 1986.
[8] Fausett, L., *Fundamentals of Neural Networks*, Prentice Hall, New York, 1994.
[9] Ghosh, S., and S.G. Henderson, Chessboard distributions and random vectors with specified marginals and covariance matrix, *Operations Research*, vol.50, pp.820-834, 2002.
[10] Ghosh, S., and S.G. Henderson, Behavior of the NORTA method for correlated random vector generation as the dimension increases, *ACM Transactions on Modeling and Computer Simulation*, vol.13, pp.1-19, 2003.
[11] Gilks, W.R., and P. Wild, Adaptive rejection sampling for Gibbs sampling, *Applied Statistics*, vol.41, pp.337-348, 1992.
[12] Haykin, S., *Neural Networks: A Comprehensive Foundation*, Macmillan Publishing, New York, 1994.
[13] Hill, R.R., and C.H. Reilly, The effects of coefficient correlation structure in two dimensional knapsack problems on solution procedure performance, *Management Science*, vol.46, pp.302-317, 2000.
[14] Hörmann, W., A technique for sampling from T-concave distributions, *ACM Transactions on Mathematical Software*, vol.231, pp.182-193, 1999.
[15] Hull, J.C., Dealing with dependence in risk simulations, *Operational Research Quarterly*, vol.28, pp.201-213, 1997.
[16] Johnson, M.E., *Multivariate Statistical Simulation*, Wiley, New York, 1987.
[17] Law, A.M., and W.D. Kelton, *Simulation Modeling and Analysis*, 3rd ed., McGraw-Hill, New York, 2000.
[18] Lewis, P.A.W., Generating negatively correlated gamma variables using the beta-gamma transformation, *Proceedings of the 1983 Winter Simulation Conference, Institute of Electrical and Electronics Engineers*, Piscataway, NJ, U.S.A., pp.175-176, 1983.
[19] Leydold, J., A rejection technique for sampling from log concave multivariate distributions, *ACM Transactions on Modeling and Computer Simulation*, vol.8, pp.254-280, 1998.
[20] Li, S.T., and J.L. Hammond, Generation of pseudorandom numbers with specified univariate distributions and correlation coefficients, *IEEE Trans. System. Man. and Cybernetics*, vol.5, pp.557-561, 1975.
[21] Lurie, P.M., and M.S. Goldberg, An approximate method for sampling correlated random variables from partially-specified distributions, *Management Science*, vol.44, pp.203-218, 1998.
[22] Marida, K.V., A translation family of bivariate distributions and Fr´echet's bounds, *Sankhya*, vol.32, pp.119-122, 1970.
[23] Moonan, W.J., Linear transformation to a set of stochastically dependent normal variables, *Journal of American Statistical Association*, vol.52, pp.247-252, 1957.
[24] Patterson, D., *Artificial Neural Networks*, Prentice Hall, Singapore, 1996.
[25] R.S. Parrish, Generating random deviates from multivariate Pearson distributions, *Computational Statistics & Data Analysis*, vol.9, pp.283-295, 1990.
[26] Ronning, G., A simple scheme for generating multivariate gamma distributions with nonnegative covariance matrix, *Technometrics*, vol.19, pp.179-183, 1997.
[27] Schmeiser, B., and R. Lal, Bivariate gamma random vectors, *Operations Research*, vol.30, pp.355-374, 1982.
[28] Schmeizer, B., Thirty years of copula, Beyond the Copulas in *G.* Dall'Aglio, S. Kotz, and G. Salinetti eds.: Advances in Probability Distributions with Given Marginals, Boston, Kluwer, pp.13-50.
[29] Song, W.T., and L. Hsiao, Generating of auto correlated random variables with a specified marginal distributions, *Proceedings of the 1993 Winter Simulation Conference, Institute of Electrical and Electronics Engineers*, Piscataway, NJ, pp.370-377, 1993.
[30] Song, W.T., L. Hsiao, and Y. Chen, Generating pseudo-random time series with specified marginal distributions, *European Journal of Operational Research*, vol.94, pp.194-202, 1996.
[31] Stanfield, P.M., J.R. Wilson, *et al.*, Multivariate input modeling with Johnson distributions, *Proceedings of the 1996 Winter Simulation Conference, Institute of Electrical and Electronics Engineers*, Piscataway, NJ, pp.1457-1464, 1996.
[32] Stanhope, S., Case studies in multivariate-to-anything transforms for partially specified random vector generation, *Insurance Mathematics and Economics*, vol.37, pp.68-79, 2005.
[33] Yen, J.C., NORTA initialization for random vector generation by numerical methods, Masters of Industrial Engineering Thesis, Chung Yaun Chiristian University, Taiwan, 2001.