Multivariate Behavioral Research, 43:355–381, 2008
Copyright © Taylor & Francis Group, LLC
ISSN: 0027-3171 print/1532-7906 online
DOI: 10.1080/00273170802285693

Simulating Multivariate Nonnormal Data Using an Iterative Algorithm

John Ruscio and Walter Kaczetow
The College of New Jersey

Simulating multivariate nonnormal data with specified correlation matrices is difficult. One especially popular method is Vale and Maurelli's (1983) extension of Fleishman's (1978) polynomial transformation technique to multivariate applications. This requires the specification of distributional moments and the calculation of an intermediate correlation matrix such that when data are transformed, the target correlation matrix is reproduced. We present an alternative technique that involves sampling data from specified population distributions and identifying the intermediate correlation matrix through an iterative, trial-and-error process. We provide R program code to implement this technique and show that it can generate data under a wide range of conditions (e.g., with empirical samples, with discrete rather than continuous data, when distributional moments are undefined or outside the boundary conditions of other techniques). This approach could be useful in many contexts, especially Monte Carlo studies of multivariate statistics.

Many research applications require the simulation of artificial data sets from user-defined populations. One crucial step in performing a Monte Carlo study of statistical analyses, for example, is the simulation of appropriate data sets. To evaluate multivariate approaches to data analysis, one must be able to generate multivariate data such that the individual variables are distributed and correlated with one another at the levels specified in the study design. Any programming environment can be used to generate random variables with normal distributions, and the common factor model can be used to reproduce a wide range of

Correspondence concerning this article should be addressed to John Ruscio, Department of Psychology, The College of New Jersey, Ewing, NJ 08628. E-mail: ruscio@tcnj.edu

correlation matrices. Whereas the generation of multivariate normal data is a simple problem, it is more challenging to generate multivariate nonnormal data. For example, applying nonlinear transformations to achieve the desired nonnormal distribution for each variable in a multivariate normal distribution usually alters the correlation matrix.

A number of strategies have been developed to simulate data drawn from specified nonnormal population distributions that reproduce a target correlation matrix more accurately. Perhaps the most popular approach is that of polynomial transformations. Fleishman (1978) devised a technique to generate a univariate score distribution with specified values of the first four moments of the population distribution (mean, variance, skew, and kurtosis). One begins by drawing values at random from a standard normal distribution, and this variable Z is then transformed using the following equation:

$$x = a + bZ + cZ^2 + dZ^3 \tag{1}$$

Fleishman presented a series of equations that are solved simultaneously such that the resulting values of a, b, c, and d yield a population distribution with the desired moments. Vale and Maurelli (1983) extended Fleishman's technique to the generation of multivariate data. In addition to the use of coefficients for each variable that yield the desired moments, their approach calculates the effects that these transformations will have on the relationships between normally distributed variables to create an intermediate correlation matrix. When one generates multivariate normal data that reproduce this intermediate correlation matrix, subsequently applying the polynomial transformation to each variable yields a data set that reproduces the target correlation matrix. The result is a sample of data in which individual variables are distributed and correlated with one another in the desired ways, as though the cases had been drawn at random from a specified multivariate nonnormal population.

The Vale-Maurelli (1983) algorithm is an instantiation of a general approach based on two key procedural elements. First, data are initially drawn from a multivariate normal distribution and then transformed to achieve the desired multivariate nonnormal distribution. Second, an intermediate correlation matrix is calculated to adjust for the influence that the data transformations will have on the correlations among variables. This will be referred to as the TC technique, short for *transform* and *calculate*, to highlight these two aspects of the procedure. The Vale-Maurelli algorithm implements the TC technique using third-order polynomial transformations. This algorithm is fairly simple, executes quickly, and has been used frequently for about a quarter century. More recent advances in polynomial transformation methods include refinements of the way that intermediate correlations are calculated (Headrick & Sawilowsky, 1999) and using a fifth-order polynomial transformation to achieve greater precision in the reproduction of nonnormal distributions by specifying the first six moments rather than only the first four (Headrick, 2002). Other implementations of the TC technique use the Pearson distribution system (Nagahara, 2004) or the generalized lambda distribution (Headrick & Mugdadi, 2006) rather than polynomial transformations.

An alternative to the TC technique is to sample each variable directly from a nonnormal population distribution and identify the intermediate correlation matrix through an iterative, trial-and-error process (Ruscio, Ruscio, & Meron, 2007). Each iteration of this approach begins by generating multivariate normal data that are subsequently replaced with the nonnormal data that were sampled from specified target populations. Even holding constant the rank-ordering of values in each of the original (normal) and desired (nonnormal) univariate distributions, substituting the latter for the former usually alters the correlation matrix. A solution to this problem is to implement the procedure in an iterative manner. After each iteration, discrepancies between the reproduced and target correlation matrices are examined to update the intermediate correlation matrix. A criterion is specified to cease iterations and retain the intermediate correlation matrix that yielded the best reproduction of the target correlation matrix. Thus, whereas the TC technique transforms data to reproduce specified distributional moments and calculates the intermediate correlation matrix, the alternative presented here samples each variable from a specified population distribution and determines the intermediate correlation matrix through an iterative, trial-and-error process. The latter approach will be referred to as the SI technique, short for *sample* and *iterate*. Before comparing and contrasting this with the TC technique, a procedure to implement the SI technique is presented in detail.

HOW THE SI TECHNIQUE WORKS

Following is a step-by-step description of the SI technique as implemented in the `GenData` program presented in the Appendix. `GenData` runs in the R computing environment, which is similar to S and S-PLUS and available as a free download from http://cran.r-project.org/mirrors.html. This version of the `GenData` program generates a data set that reproduces the distributions and correlations observed in a sample of data provided by the user. Terms that are italicized in this description correspond to variable names in the program. In brackets contained within each of the first three steps, an alternative use of the SI technique is described: Rather than providing a sample of data whose characteristics are reproduced, the user specifies the number of variables, the population distribution for each variable, the sample size, and the target correlation matrix. Subsequent steps are identical for both uses of the SI technique.

1. The user provides a data matrix *Supplied.Data* in which rows correspond to cases and columns correspond to variables. The program calculates the sample size N and the number of variables k. [Alternatively, the user can specify the number of variables to generate, the population distribution for each variable, the sample size, and the target correlation matrix.]
2. Generate a univariate distribution of scores for each variable. Each variable's scores are bootstrapped (resampled with replacement) from the corresponding variable's distribution in the supplied data set (Efron & Tibshirani, 1993). [Alternatively, the user can provide a function to sample each variable's scores from a specified population or use values representing uniform quantiles across a specified population; these populations can be identical for all variables or different for each and discrete or continuous in nature.] These score distributions are stored in a matrix *Distributions* of N (sample size) rows by k (number of variables) columns.

At this point in the procedure, score distributions have been sampled independently for each variable. If one were to stop here, the variables would be uncorrelated (save for normal sampling error). The remaining steps are required to reproduce any target correlation matrix that is not an identity matrix.

3. Calculate the correlation matrix *Target.Corr* and store a copy as *Intermediate.Corr*, which will be modified as the procedure iterates. [If the user does not supply a data set, the target correlation matrix cannot be calculated and must be specified instead.]
4. If the user has not specified the number of factors (*N.Factors*) that will be used to reproduce the correlations in the matrix *Target.Corr*, this is determined through a parallel analysis (Montanelli & Humphreys, 1976) implemented as recommended by Reise, Waller, and Comrey (2000). The mean eigenvalues are obtained for 100 random data sets with N cases and k variables. In each random data set, each variable's univariate score distribution is bootstrapped from values stored in *Distributions*; correlations among variables differ from 0 only due to random sampling error. The number of factors equals the number of eigenvalues calculated from *Target.Corr* that exceed the mean eigenvalues for the corresponding factors in the random data.
5. Create a matrix *Shared.Comp* containing random standard normal values that will be partly shared by all variables. This matrix has N rows and *N.Factors* columns. All random standard normal values are drawn from a

population with $\mu = 0, \sigma = 1$ using the "rnorm" function in R (Wichura, 1988).

6. Create a matrix *Unique.Comp* containing random standard normal values that will contribute the unique, or error, component for each variable. This matrix has N rows and k columns.
7. Determine the weights for the shared and unique components that will reproduce *Intermediate.Corr* through a factor analysis of *Intermediate.Corr* using *N.Factors* latent factors. The number of shared loadings to be saved will be $k \times N.Factors$. For each variable i, $Shared.Load_{i,1} \ldots Shared.Load_{i,Factors}$ will be used to weight the shared components in a subsequent step, with the weights for each unique component calculated as

$$Unique.Load_i = \sqrt{1 - \sum_{j=1}^{N.Factors} Shared.Load_{i,j}^2}. \tag{2}$$

8. Calculate each simulated variable ($i = 1$ to k) as a new vector of N values by weighting the shared and unique components for each by the factor loadings calculated in Step 7.
9. Sort the simulated data set by each variable in turn, and replace the normal distributions with the nonnormal distributions stored in *Distributions*. This key step preserves the rank-ordering of cases on each variable but substitutes the desired nonnormal distributions for the normal distributions that were used in earlier steps.
10. Calculate the correlation matrix *Reproduced.Corr* in the simulated data, then compute a matrix of residual correlations: *Residual.Corr* = *Target.Corr* − *Reproduced.Corr*. Particularly on the first iteration of the procedure, these residuals may be substantial because the nonnormal distributions were substituted after correlations were reproduced using the common factor model with normal data.
11. Check whether, after substituting the nonnormal distributions, the current iteration achieves the best correlational reproduction so far by calculating the root mean square residual (RMSR) correlation:

$$RMSR = \sqrt{\frac{\sum r_{residual}^2}{(k[k-1]/2)}}, \tag{3}$$

where each $r_{residual}$ represents a value below the diagonal in *Residual.Corr*. If the RMSR correlation is the lowest so far (as on the 1st iteration it automatically is), store a copy of *Intermediate.Corr* as *Best.Corr*, store the current RMSR correlation as *Best.RMSR*, and begin (or reset) a counter *Trials.Without.Improvement* at 0; this is used to allow a finite number of additional iterations to attempt to achieve a lower RMSR correlation. Empirical results revealed rapidly diminishing returns, and 5 additional iterations is the default value of *Max.Trials*; users can specify an alternative value.

12. If the current iteration has not achieved the best correlational reproduction thus far, add 1 to *Trials.Without.Improvement*. If this now equals *Max.Trials*, proceed to Step 13; otherwise, create a new *Intermediate.Corr* by adding a fraction of *Residual.Corr* to *Best.Corr*, and return to Step 7. The fraction to be added is calculated as $\mathit{Initial.Multiplier}/2^{\mathit{Trials.Without.Improvement}}$, where *Initial.Multiplier* is a user-specified step-size multiplier (by default, this is set to 1.00). Thus, the algorithm allows for *Max.Trials* iterations to attempt to improve the reproduction of correlations. On the first try, the residual correlations are added in their entirety to *Best.Corr* (i.e., the step size multiplier $= 1.00/2^0 = 1.00$, because the counter is reset to 0 when the RMSR correlation is the lowest so far). If this first try fails to improve the reproduction of correlations, a second try is made using a smaller fraction of *Residual.Corr* (i.e., the multiplier $= 1.00/2^1 = .50$). This repeats, if necessary, for additional attempts (i.e., multipliers = .25, .125, .0625, . . . ,). If any of these trials improve correlational reproduction, the counter is reset to allow up to *Max.Trials* starting from a new *Best.Corr*. If the smallest step from *Best.Corr* fails to yield improvement, the iterative algorithm terminates.
13. Construct variables using the factor loadings that generated correlations which, when altered by substituting the nonnormal distributions, best reproduced *Target.Corr*. These loadings are derived from a factor analysis of *Best.Corr*, which was stored for this purpose.
14. Report the results (sample size, number of variables, number of iterations, number of latent factors used, RMSR correlation) and return the sample of nonnormal multivariate data.

In sum, this procedure implements the common factor model with user-specified nonnormal distributions to reproduce a target correlation matrix in an iterative manner. Although early steps in the algorithm generate and work with multivariate normal data, Step 9 substitutes the desired nonnormal score distributions that were sampled for each variable. The algorithm evaluates the extent to which this substitution affects the reproduction of the target correlation matrix and iteratively modifies an intermediate correlation matrix.

To illustrate the process, consider a sample run of the program with $N = 100$, $k = 4$, and variables' population distributions defined as follows: $N(0, 1)$;

$U(0, 1)$; $\chi^2(2)$; $B(10, .25)$. The target correlation matrix was

$$\begin{bmatrix} 1.000 & .700 & .300 & .300 \\ .700 & 1.000 & .300 & .300 \\ .300 & .300 & 1.000 & .700 \\ .300 & .300 & .700 & 1.000 \end{bmatrix}.$$

The program correctly determined that two factors should be used and, after sampling score distributions for each of the variables, identified the intermediate correlation matrix

$$\begin{bmatrix} 1.000 & .794 & .375 & .319 \\ .794 & 1.000 & .386 & .276 \\ .375 & .386 & 1.000 & .834 \\ .319 & .276 & .834 & 1.000 \end{bmatrix}$$

on the 5th iteration. This yielded the lowest RMSR correlation (.006), with the reproduced correlation matrix as follows:

$$\begin{bmatrix} 1.000 & .701 & .294 & .287 \\ .701 & 1.000 & .296 & .297 \\ .294 & .296 & 1.000 & .700 \\ .287 & .297 & .700 & 1.000 \end{bmatrix}.$$

Table 1 shows the intermediate correlation matrix, the reproduced correlation matrix, and the RMSR correlation for each iteration. The substantial decrease

TABLE 1
Correlations From a Run of the GenData Program

Iteration	*Intermediate Correlations*	*Reproduced Correlations*	*RMSR r*
1	.700, .300, .300, .300, .300, .700	.616, .199, .273, .204, .276, .587	.082
2	.784, .401, .327, .396, .324, .813	.702, .316, .328, .305, .344, .694	.023
3	.782, .385, .298, .391, .279, .818	.698, .308, .297, .302, .323, .682	.012
4	.784, .378, .302, .389, .257, .837	.691, .303, .282, .303, .280, .703	.012
5	.794, .375, .319, .386, .276, .834	.701, .294, .287, .296, .297, .700	.006

Note. RMSR = root mean square residual. For this run, $N = 100$ with four variables whose population distributions were defined as follows: (a) $N(0, 1)$; (b) $U(0, 1)$; (c) $\chi^2(2)$; (d) $B(10, .25)$. The target correlation matrix was based on two correlated factors: $r_{12} = r_{34} = .70$, with all other variable pairs correlated at $r = .30$. For both intermediate and reproduced correlation matrices, the off-diagonal correlations are listed in the following order: r_{12}, r_{13}, r_{14}, r_{23}, r_{24}, r_{34}. On the 1st iteration, the intermediate correlations were the target correlations; on subsequent iterations, intermediate correlations were updated based on how accurately the target correlations were reproduced. The program correctly determined that two factors were required and achieved RMSR $r = .006$, on the 5th iteration. Five additional iterations failed to achieve a lower RMSR r.

in the RMSR correlation from the 1st to the 2nd iteration is typical, as the first adjustment to the intermediate correlation matrix tends to compensate well for the attenuation in correlations caused by the substitution of nonnormal for normal distributions. Usually, a subsequent iteration eventually yields an even lower RMSR correlation, but diminishing returns are the norm. It is not uncommon for the RMSR correlation to increase on some iterations as these represent adjustments that were too large; subsequent iterations reduce the step-size multiplier to achieve lower RMSR correlation values.

Because the `GenData` program provided here is a revision and generalization of the implementation of the SI technique developed by Ruscio et al. (2007), there are many similarities. Readers familiar with Ruscio et al.'s (2007) `DimSample` program will recognize the general outline of the SI technique, but three differences are noteworthy. First, whereas the `DimSample` program was designed exclusively to reproduce the characteristics of a supplied data set, the `GenData` program contains instructions to adapt it for drawing samples from user-specified populations that reproduce a user-specified correlation matrix. Second, whereas the `DimSample` program uses the Kaiser (1960) criterion of the number of eigenvalues > 1 to determine the number of latent factors to use, the `GenData` program performs a parallel analysis (see Step 4 in the aforementioned description); alternatively, one can specify the number of factors when using the `GenData` program. Third, the `GenData` program was modified to run more efficiently than `DimSample`. The improvements in speed can be substantial, and they are most notable with large samples.

REPRODUCING CORRELATIONS IN SAMPLES VERSUS POPULATIONS

When using the SI technique, each variable's score distribution is sampled at random from a target population distribution. As a result, across samples the variables' distributions will differ due to normal sampling error. This is not the case for the reproduction of the target correlation matrix. The SI technique strives to reproduce the target correlation matrix as closely as it can for each data set that is generated. Although this may be useful for certain purposes, the correlations observed across multiple samples generated in this way will not vary due to normal sampling error; instead, they will cluster around the target correlations more closely than would be expected by chance. If one would prefer to incorporate normal sampling error in the correlations, the SI technique can be used to do so by creating a single sample that is sufficiently large to be treated as a population and drawing a series of random samples from this population. For example, in a study whose design calls for samples of data with $N = 100$ within a certain condition, one could use the SI technique to generate a sample of N

= 1,000,000 that reproduces the target correlation matrix and then take as many random samples as desired. Creating one large sample to treat as the population would enable a very precise reproduction of the target correlation matrix, and the correlation matrices of samples drawn from this population would vary due to normal sampling error. In practice, the maximal size of the finite population that one might generate will be limited by factors such as the number of variables requested and the computer memory required.

SIMULATING MULTIGROUP DATA SETS

The SI technique also can be used to simulate data sets in which observations belong to multiple groups, which would be useful to study multivariate data-analytic procedures such as cluster analysis, latent class analysis, or taxometric analysis. All that is required is to call the `GenData` program once per group and then merge the resulting data sets to yield the multigroup data. This approach is described in Ruscio et al. (2007) and illustrated in the `TaxSample` program presented in their Appendix. Not only will variables be distributed and correlated at the desired levels within each group, but also they will differentiate groups to the desired extent. By generating and analyzing comparison data sets with different numbers of groups, researchers could develop and test indices to identify the appropriate number of groups in target data sets, which remains a challenging problem for many data-analytic procedures (e.g., McLachlan & Peel, 2001). Ruscio et al. (2007) showed that this approach can be helpful in taxometric analyses.

COMPARING THE TC AND SI TECHNIQUES

Whereas the TC technique, as implemented by Vale and Maurelli (1983), has been used frequently, the SI technique is a relatively new alternative. Moreover, the iterative nature of the SI technique is twofold—one layer of iterations involves updating the intermediate correlation matrix at each step, and the factor analysis of this matrix at each step introduces a second layer of iterations—and requires considerably more processing time than the noniterative TC technique. Nonetheless, there are six potential advantages to the SI technique that can compensate for this increase in processing time.

Handling Distributions With Undefined Moments

Because the TC technique requires that the user specify distributional moments, this approach cannot be used to generate data for distributions with undefined moments. In contrast, the SI technique requires only that a function be available to generate scores from a distribution, not that its moments be defined. The family of g-and-h distributions (Hoaglin, 1985), in which two parameters control the amount of asymmetry (g) and tail weight (h), helps to illustrate this point. To generate g-and-h distributions, one begins by sampling values of Z from a standard normal distribution. If $g = h = 0$, no transformation is performed and the distribution remains normal. If $g \neq 0$ and $h = 0$, the value of g determines the amount of asymmetry that is imparted through the following transformation (Hoaglin, 1985, p. 463):

$$y_g(Z) = \frac{e^{gZ} - 1}{g}.$$

The direction and magnitude of skew are determined by the sign and size of g, respectively. If $g = 0$ and $h \neq 0$, the value of h determines the extent to which tail weight is affected by the following transformation (Hoaglin, 1985, p. 479):

$$y_h(Z) = Ze^{hZ^2/2}.$$

Positive values of h lengthen the tails and negative values of h shorten the tails; the larger the size of h, the greater its influence. Finally, if $g \neq 0$ and $h \neq 0$, both asymmetry and tail weight are affected (Hoaglin, 1985, p. 486):

$$y_{g,h}(Z) = \left(\frac{e^{gZ} - 1}{g}\right)e^{hZ^2/2}.$$

Many of the g-and-h distributions for which scores can be generated have undefined moments. For example, Hoaglin (1985) presented equations for the skew (γ_1) and kurtosis (γ_2) of the g, h, and g-and-h population distributions and noted that the nth moment was defined only for $0 \leq h < 1/n$. Even though one can generate h or g-and-h score distributions for many values of h beyond either end of this range (Figure 1, right graph, contains an example that will be discussed shortly), skew and kurtosis will be undefined. As a result, the TC technique cannot reproduce many of the h and g-and-h distributions. In contrast, any distribution that can be generated—including all of the g-and-h distributions—can be used by the SI technique; R program code for generating g-and-h distributions is available on request.

No Boundary Conditions for Defined Moments

There are many population distributions with defined moments that cannot be used with the TC technique. For example, Headrick and Kowalchuk (2007) showed that for symmetric distributions ($\gamma_1 = 0$), the Fleishman method (and

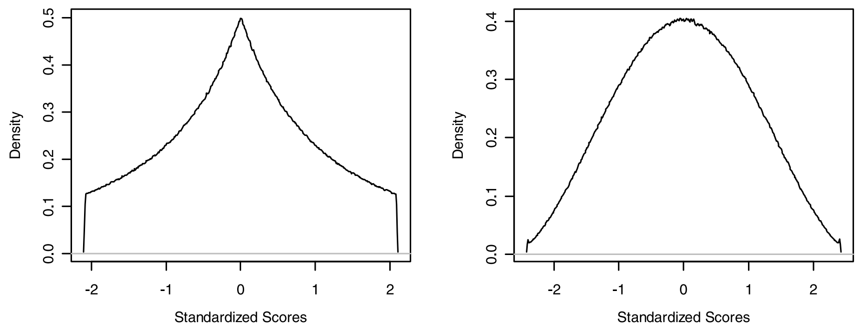

FIGURE 1 Two symmetric distributions ($\gamma_1 = .00$) that are equally light-tailed, relative to a normal distribution ($\gamma_2 = -.65$). The distribution on the left was generated by drawing 10 million values from uniform distributions (ranging from 0 to 1, exclusive) for each of two variables x and y, calculating $z = x/(x + y)$, and standardizing z. The distribution on the right was generated by drawing 10 million values from a g-and-h distribution ($g = 0$, $h = -.07832$) and standardizing these values. Setting aside the fact that the negative kurtosis value lies outside the boundary conditions of the popular Vale-Maurelli technique for simulating multivariate nonnormal data, because the first four moments of these distributions are identical one could not distinguish between them when using this technique.

therefore also the Vale-Maurelli method) is constrained to $0 < \gamma_2 < 43.2$. These bounds narrow when $\gamma_1 \neq 0$, which excludes many distributions (e.g., χ^2, lognormal, a wide range of g-and-h distributions). Implementing the TC technique using fifth-order polynomials (Headrick, 2002), the Pearson system of curves (Nagahara, 2004), or the generalized lambda distribution (Headrick & Mugdadi, 2006) rather than third-order polynomials can extend these bounds considerably, but there remain limits to the moments that can be used. Once again, an advantage of the SI technique is that it can be used whenever a function is available to generate scores from the desired univariate distribution; no limits are placed on the range of moments in the population distributions.

Distinguishing Distributions With Equal Moments

A finite number of distributional moments does not uniquely define a distribution. For example, the first four moments are equal for the two distributions shown in Figure 1, which means that they would be indistinguishable to the Vale-Maurelli procedure, which may be the most widely used implementation of the TC technique. When one provides a set of moments within the boundary conditions of any implementation of the TC technique, it can generate one distribution but not others with equal moments. In contrast, the SI technique does not require the specification of moments. Rather, one can choose among functions to generate scores even when they correspond to population distributions with equal moments. For example, one could generate either the tent-shaped distribution on the left side of Figure 1 or the short-tailed but otherwise fairly normal distribution to its right.

Generating Discrete Distributions

The TC technique begins with continuous data and transforms it in a way that retains its continuity. This means that many types of data commonly encountered in actual research, such as binary or ordered categorical variables, cannot be reproduced using the TC technique. Placing thresholds along the continuous variables simulated using the TC technique to discretize the data usually will alter the correlation matrix. The SI technique does not require continuous population distributions, only that the scale is at least ordinal in nature (otherwise, nonnormal distributions cannot be substituted for normal ones based on rank-ordered scores). Several Monte Carlo studies have used the SI technique to generate data that varied across differing numbers of ordered categories to include this as a factor in the design (e.g., Ruscio, 2007; Ruscio, Haslam, & Ruscio, 2006, Ruscio et al.,2007).

Reproducing Characteristics of a Unique Sample of Empirical Data

Because of the limitations discussed earlier, the TC technique cannot necessarily be used to reproduce the characteristics observed in a unique sample of empirical data. Distributional moments may lie beyond the technique's boundary conditions, observed moments do not uniquely define variables' distributions, and discrete distributions cannot be reproduced. The SI technique was developed originally for precisely this purpose. Ruscio et al. (2007) used a standard bootstrap method to reproduce the observed distributions (Efron & Tibshirani, 1993). This treats the sample as the best estimate of the population, obtaining scores for a simulated data set by sampling at random (with replacement) from the observed distributions. Investigators have taken advantage of this particular application of the SI technique in more than two dozen taxometric investigations (Ruscio et al., 2006, counted 21, and more have been published since that book went to press) where it is becoming standard practice (Ruscio & Marcus, 2007). The version of the `GenData` program in the Appendix uses the SI technique in this way.

Controlling Dimensionality

Whereas the TC technique does not allow users to specify the number of latent factors used to reproduce a correlation matrix, the SI technique does. When dimensionality is itself an important factor to vary in the design of a study or in the generation of comparison data to help interpret results for a unique set of research data, the flexibility of the SI technique represents a substantial feature. For example, suppose that one wished to study the validity with which various criteria identify the number of factors to extract in exploratory factor analyses. Ideally, a data simulation technique would be able to hold distributions and correlation matrices constant across conditions that varied only in the number of latent factors. The SI technique could be used to generate data for such a study, but the TC technique could not.

EMPIRICAL EVALUATION OF THE SI TECHNIQUE

When using the `GenData` program to implement the SI technique, one must either provide the functions that yield score distributions drawn from the desired populations or supply a sample of data to bootstrap scores from the observed distributions. If the distributions in simulated data do not represent random samples from the specified population distributions, this cannot be attributed to the program itself; rather, it would reveal a bias in the supplementary function(s) used for this purpose. Because the `GenData` program does not generate score distributions through transformations and does not alter them as it iterates, its performance should be assessed solely in terms of how well it can reproduce the target correlation matrix *given* the user's choice of population distributions and functions to sample data from them. If the target correlation matrix is reproduced well, this indicates that the iterative, trial-and-error algorithm has determined an appropriate intermediate correlation matrix.

In the series of trials described in the following section, the performance of the SI technique was examined across 22 different distributions: (1)–(12) were g-and-h distributions that included symmetric and asymmetric forms that varied in kurtosis ($\gamma_2 = .00$ to 1.64 for symmetric distributions, $\gamma_2 = 2.51$ to 622.12 for asymmetric distributions); (13) and (14) were the distributions shown in Figure 1, whose first four moments are equivalent; (15) and (16) were χ^2 distributions with 2 *df* and 1 *df*, respectively; (17) was a continuous uniform distribution ranging from 0 to 1 and (18) was a discrete uniform distribution ranging across the whole numbers 1 to 7; (19) and (20) were binomial distributions with size 10 and $p = .50$ and $p = .25$, respectively; (21) and (22) were binary distributions with $p = .50$ and $p = .25$, respectively. Table 2 provides a complete list, along with the values of γ_1 and γ_2 for each population distribution. Figure 2 shows the density for g-and-h distributions (1) through (12). Distributions (13) and (14) were shown in Figure 1, and the remaining distributions are members of the familiar χ^2 and binomial families.

TABLE 2
Skew (γ_1) and Kurtosis (γ_2) for Distributions in the Simulations

Distribution	γ_1	γ_2	*BDI Item Number*[e]	γ_1	γ_2
$g = .00$, $h = .00$ [$N(0, 1)$]	.00	.00	12	1.87	3.16
$g = .00$, $h = .10$	.00[a]	2.51[a]	16	1.33	1.70
$g = .00$, $h = .20$	.00[a]	33.22[a]	18	2.12	4.62
$g = .10$, $h = .00$	.30[a]	.16[a]	19	4.30	21.21
$g = .10$, $h = .10$	.52[a]	3.19[a]	20	2.08	4.86
$g = .10$, $h = .20$	1.30[a]	53.03[a]	21	3.56	14.28
$g = .20$, $h = .00$	.61[a]	.68[a]			
$g = .20$, $h = .10$	1.08[a]	5.50[a]			
$g = .20$, $h = .20$	2.81[a]	152.98[a]			
$g = .30$, $h = .00$	.95[a]	1.64[a]			
$g = .30$, $h = .10$	1.70[a]	10.41[a]			
$g = .30$, $h = .20$	4.84[a]	622.12[a]			
$g = .00$, $h = -.07832$[b]	.00	−.65[c]			
$z = x/(x + y)$[d]	.00	−.65[c]			
$\chi^2(2)$	2.00	6.00			
$\chi^2(1)$	2.83	12.00			
$U(0, 1)$	.00	−1.20			
$U(1, 2, 3, \ldots, 7)$	.00	−1.25			
$B(10, .50)$	.00	−.20			
$B(10, .25)$	.37	−.07			
$B(2, .50)$	.00	−1.00			
$B(2, .25)$	.82	−.33			

Note. BDI = Beck Depression Inventory.
[a]Values were calculated using formulas provided by Hoaglin (1985), with 3 subtracted from γ_2 such that $\gamma_2 = 0$ for a normal distribution. [b]Even though this is not a valid g-and-h pdf, one can generate data using these parameters. [c]$\gamma_2 = -.65$ was calculated in a sample of 10 million cases drawn from this population. [d]x and y were drawn from uniform distributions (ranging from 0 to 1, exclusive), and subsequent to its calculation z was standardized. [e]γ_1 and γ_2 for each item were calculated in the sample of $N = 2{,}293$ used by Ruscio, Ruscio, and Keane (2004).

For each of these population distributions, the influence of three data characteristics that users can specify was examined: sample size (N), the number of variables (k), and the number of latent factors (*N.Factors*). Within each condition studied, 1,000 data sets were created using `GenData` and the median RMSR correlation is reported because the values tended to be positively skewed. For all results reported later, the standard error of the M RMSR correlation within each condition was $\leq .002$ and usually $< .001$. To facilitate the interpretation of results for readers more familiar with the goodness of fit index (GFI; McDonald, 1999) as a measure of the accuracy with which a correlation matrix is reproduced, Figure 3 plots the relationship between the RMSR correlation and the GFI for conditions typical of the simulations presented later ($k = 4$, with two correlated

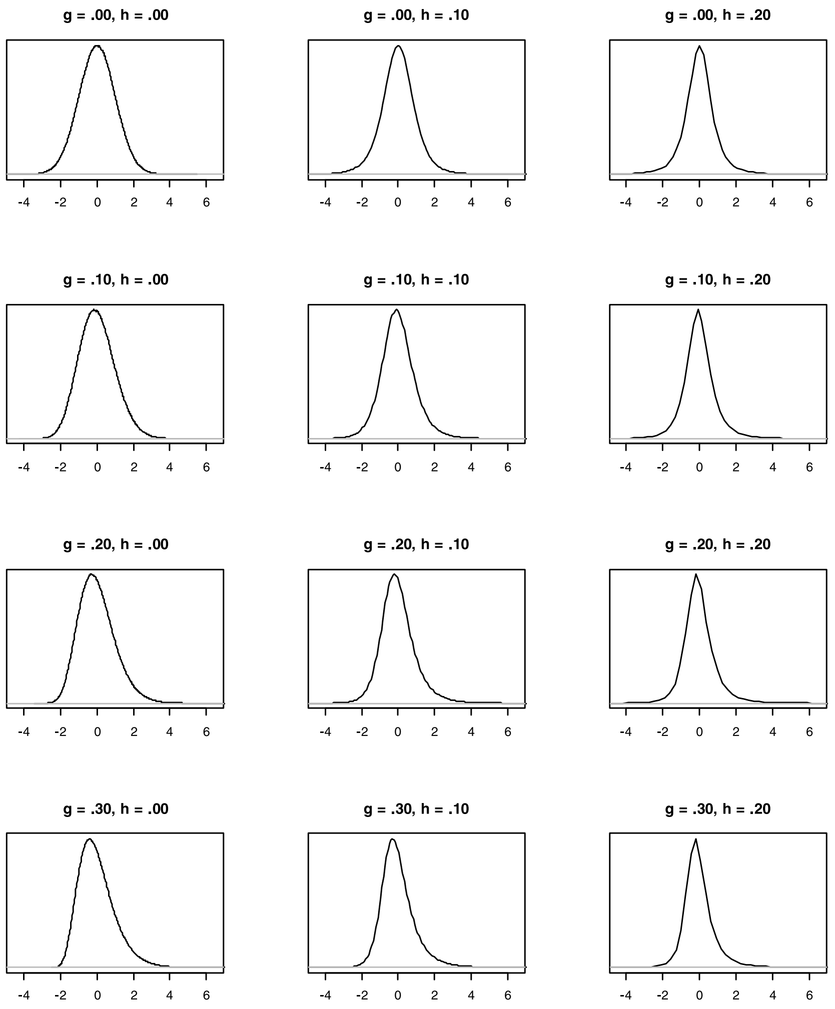

FIGURE 2 *g*-and-*h* distributions used in the simulations, illustrated by drawing 10 million values from each population distribution ($\mu = 0$, $\sigma = 1$) and plotting their density using common *x* axis scaling.

factors where $r_{12} = r_{34} = .70$ and all other variable pairs are correlated $r = .30$). RMSR correlations up to .058 yielded GFIs $\geq .95$, and RMSR correlations up to approximately .077 yielded GFIs $\geq .90$. This should be understood as a rule of thumb for the correspondence between RMSR correlations and GFIs, given simplifying assumptions that are approximately true for the conditions studied here.

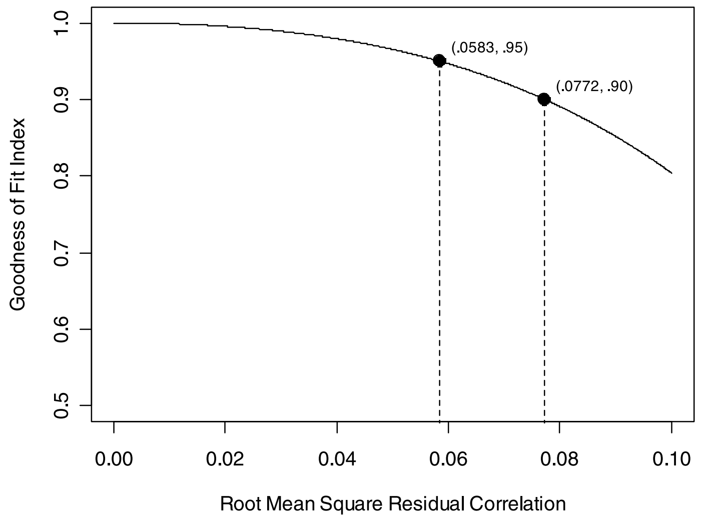

FIGURE 3 Goodness of fit index (GFI) calculated for varying root mean square residual (RMSR) correlations in 4 × 4 matrices. The target correlation matrix was based on two correlated factors: $r_{12} = r_{34} = .70$, all other variable pairs correlated at $r = .30$. The reproduced correlation matrix was constructed by adding the RMSR correlation to one half of the off-diagonal correlations (including one half of the .70 values and one half of the .30 values) and subtracting it from the other half.

In the first series of simulations, sample size varied ($N = 20$, 40, 100, and 10,000), $k = 4$, and the target correlation matrix was constructed on the basis of two correlated factors ($r_{12} = r_{34} = .70$, and $r = .30$ for all other variable pairs). For all distributions studied under these conditions, there was a sharp improvement in performance between $N = 20$ and $N = 40$ (see Table 3). When $N = 20$, RMSR correlations were unacceptably large for all distributions (.194 to .217; $M = .200$, $SD = .005$).[1] When $N = 40$, these values were considerably lower (.021 to .064; $M = .034$, $SD = .012$); as plotted in Figure 3, a RMSR correlation of .034 would correspond to a GFI of .986. When $N = 100$, RMSR correlations were lower still (.006 to .028; $M = .012$, $SD = .006$). These results demonstrate the importance of checking that correlations are reproduced

[1]Either of two strategies may enable investigators to reproduce correlations better in small samples. First, one could add an outer loop to the `GenData` program that sets a threshold RMSR correlation, rerunning the program with a new seed if this criterion is not met. Of course, there is no guarantee that a specified level of precision can be met. Second, one could reproduce each variable's distribution using uniform quantiles along a specified population, which might afford a better reproduction of correlations than with randomly sampled scores; see the Appendix for further notes and illustrative program code to implement this approach.

TABLE 3
Root Mean Square Residual (RMSR) Correlation for Simulated Data With Varying Sample Sizes

Distribution	*N = 20*	*N = 40*	*N = 100*	*N = 10,000*
$g = .00$, $h = .00$ [$N(0, 1)$]	.195	.023	.006	.001
$g = .00$, $h = .10$	.198	.027	.008	.001
$g = .00$, $h = .20$	.199	.033	.011	.001
$g = .10$, $h = .00$	.198	.027	.009	.001
$g = .10$, $h = .10$	.196	.029	.008	.001
$g = .10$, $h = .20$	.195	.034	.012	.001
$g = .20$, $h = .00$	.197	.024	.008	.001
$g = .20$, $h = .10$	.202	.029	.009	.001
$g = .20$, $h = .20$	.200	.040	.015	.001
$g = .30$, $h = .00$	.199	.025	.007	.001
$g = .30$, $h = .10$	.200	.032	.010	.001
$g = .30$, $h = .20$	.201	.041	.017	.001
$g = .00$, $h = -.07832$	.201	.021	.006	.001
$z = x/(x + y)$	.198	.022	.006	.001
$\chi^2(2)$	.209	.042	.013	.001
$\chi^2(1)$	.217	.060	.020	.001
$U(0, 1)$	.201	.021	.006	.001
$U(1, 2, 3, \ldots, 7)$	.199	.031	.012	.001
$B(10, .50)$	.196	.033	.013	.001
$B(10, .25)$	.194	.038	.015	.001
$B(2, .50)$	.198	.051	.023	.001
$B(2, .25)$	.200	.064	.028	.001
M	.200	.034	.012	.001

Note. Within each condition, 1,000 samples of data were generated with $k = 4$ and a target correlation matrix based on two correlated factors: $r_{12} = r_{34} = .70$, all other variable pairs correlated at $r = .30$. The median value of the RMSR correlation is reported; $SE \leq .002$ for all conditions.

sufficiently well when generating small samples of data. If the SI technique is used to create modest to large data sets, or a single data set to be treated as a population from which samples are then drawn, correlations can be reproduced very well. Even when the size of a finite population was set to just 10,000 cases, RMSR correlations were .001 for all distributions studied. In practice, one would likely use an even larger N for this purpose, which would afford even more precise correlational reproduction; an N of only 10,000 was used here so that 1,000 replications could be completed for each distribution in a timely fashion.[2]

[2]In a test of the time required to generate data, finite populations in which eight variables were distributed as $\chi^2(2)$ and shared loadings on two correlated factors were created, and a series of

The results shown in Table 3 also reveal consistent differences in RMSR correlations across population distributions. Values increased with departures from normality, particularly when both skew and kurtosis diverged from 0, as well as for discrete distributions, especially the binary populations. Nonetheless, variation across distributions was much smaller than variation across sample sizes.

In the second series of simulations, the number of variables varied ($k = 2$, 4, 8, 16, 32), $N = 100$, and the target correlation matrix was constructed on the basis of two correlated factors (the first half of the variables correlated with one another at $r = .70$, as did the second half of the variables; all other variable pairs correlated at $r = .30$); for each value of k, 1,000 data sets were created using `GenData`. RMSR correlations increased with the number of variables, but even at the modest sample size of $N = 100$ they remained low for most distributions even with as many as 32 variables (see Table 4). From $k = 2$ through $k = 16$, all values were $\leq .044$. At $k = 32$, the values remained acceptable for most distributions but varied more widely (.040 to .089; $M = .055$, $SD = .014$). In the same way that was observed across sample sizes, population distributions yielded larger RMSR correlations when they deviated from normality or were discrete. Even for the condition that yielded the largest value—$\chi^2(1)$ distributions with $k = 32$, RMSR correlation = .089—correlations could be reproduced very well if larger data sets were created to treat as populations from which samples would be drawn. In 100 such data sets generated with $N = 10{,}000$, RMSR correlations ranged from .004 to .005.

In the third series of simulations, the number of factors varied (*N.Factors* = 1, 2, 3, 4), $N = 100$, and $k = 12$. To construct the target correlation matrix for each condition, each of *N.Factors* equal-size subsets of the k variables correlated $r = .70$ with one another, and all other variable pairs correlated $r = .30$. The RMSR correlations were comparable and small for 1 factor ($M = .033$, $SD = .006$) and 2 factors ($M = .030$, $SD = .005$), and increasing but still fairly small for 3 factors ($M = .040$, $SD = .007$) and 4 factors ($M = .058$, $SD = .006$); see Table 5 for complete results. Differences in values across population

random samples was drawn from each population. On a PC-compatible laptop running R version 2.4.1 with a 2.16 GHz processor, the `GenData` program created finite populations of 10,000 cases in 2–3 s. This time increased to approximately 30 s for finite populations of 100,000 cases and approximately 6.5 min for finite populations of 1,000,000 cases. At each size, drawing 1,000 random samples of $N = 100$ for subsequent analysis required a total time of less than 1 s. The TC technique can run considerably faster, but to generate an equivalent number of samples from the same population it must be run repeatedly. For comparison with the aforementioned times, a program that implements the TC technique using third-order polynomial transformations was run 1,000 times with $N = 100$. Each such trial took approximately 8 min. Thus, for simulation studies with large numbers of replications per data condition, the comparatively slow speed of the SI technique (as implemented in the `GenData` program) for generating a finite population is offset by the speed with which large numbers of random samples can be drawn for analysis.

TABLE 4
Root Mean Square Residual (RMSR) Correlation for Simulated Data With Varying Numbers of Variables

Distribution	*k* = *2*	*k* = *4*	*k* = *8*	*k* = *16*	*k* = *32*
$g = .00$, $h = .00$ [$N(0, 1)$]	.000	.006	.021	.027	.040
$g = .00$, $h = .10$	.000	.008	.022	.028	.044
$g = .00$, $h = .20$	.000	.011	.026	.033	.063
$g = .10$, $h = .00$	.000	.009	.022	.029	.047
$g = .10$, $h = .10$	.000	.008	.022	.029	.047
$g = .10$, $h = .20$	.001	.012	.027	.035	.065
$g = .20$, $h = .00$	.000	.008	.022	.028	.043
$g = .20$, $h = .10$	.000	.009	.024	.030	.053
$g = .20$, $h = .20$	.001	.015	.029	.038	.073
$g = .30$, $h = .00$	.000	.007	.023	.029	.049
$g = .30$, $h = .10$	.001	.010	.025	.034	.061
$g = .30$, $h = .20$	.001	.017	.034	.043	.083
$g = .00$, $h = -.07832$	.000	.006	.021	.027	.041
$z = x/(x + y)$	.000	.006	.021	.027	.040
$\chi^2(2)$	.001	.013	.029	.036	.070
$\chi^2(1)$	.001	.020	.035	.044	.089
$U(0, 1)$	.000	.006	.021	.028	.043
$U(1, 2, 3, \ldots, 7)$	.003	.012	.023	.028	.047
$B(10, .50)$	.004	.013	.022	.027	.045
$B(10, .25)$	.005	.015	.023	.028	.047
$B(2, .50)$	.009	.023	.027	.031	.059
$B(2, .25)$	.012	.028	.031	.035	.070
M	.002	.012	.025	.032	.055

Note. Within each condition, 1,000 samples of data were generated with $N = 100$ and a target correlation matrix based on two correlated factors: the first half of the variables correlated with one another at $r = .70$, as did the second half of the variables; all other variable pairs correlated at $r = .30$. The median value of the RMSR correlation is reported; $SE \leq .002$ for all conditions.

distributions were similar to those observed earlier. Even for the condition that yielded the largest value—$\chi^2(1)$ distributions with 4 factors, RMSR correlation = .076—correlations could be reproduced very well if larger data sets were created to treat as populations from which samples would be drawn. In 100 data sets generated with $N = 10{,}000$, RMSR correlations ranged from .002 to .015 ($M = .006$, $SD = .002$).

The results of these three series of simulations were used to subject the SI technique to additional tests under especially demanding conditions. Data sets were generated with $k = 32$ variables and a target correlation matrix constructed using 4 factors. For each of the three population distributions that posed the greatest challenge to the accurate reproduction of target correlation matrices—

TABLE 5
Root Mean Square Residual (RMSR) Correlation for Simulated Data With Varying Numbers of Factors

Distribution	*1 Factor*	*2 Factors*	*3 Factors*	*4 Factors*
$g = .00$, $h = .00$ [$N(0, 1)$]	.027	.025	.034	.051
$g = .00$, $h = .10$	.029	.026	.036	.053
$g = .00$, $h = .20$	.037	.031	.041	.059
$g = .10$, $h = .00$	.030	.027	.037	.054
$g = .10$, $h = .10$	.030	.027	.036	.055
$g = .10$, $h = .20$	.038	.033	.044	.059
$g = .20$, $h = .00$	.029	.026	.033	.053
$g = .20$, $h = .10$	.032	.029	.037	.057
$g = .20$, $h = .20$	.042	.036	.048	.062
$g = .30$, $h = .00$	.031	.027	.038	.055
$g = .30$, $h = .10$	.036	.032	.041	.060
$g = .30$, $h = .20$	.047	.041	.054	.069
$g = .00$, $h = -.07832$	.028	.025	.036	.052
$z = x/(x + y)$	.028	.025	.034	.051
$\chi^2(2)$	.039	.034	.046	.065
$\chi^2(1)$	.047	.041	.056	.076
$U(0, 1)$	.029	.026	.033	.052
$U(1, 2, 3, \ldots, 7)$	.029	.027	.037	.055
$B(10, .50)$	.027	.026	.037	.055
$B(10, .25)$	.028	.027	.039	.056
$B(2, .50)$	.031	.030	.045	.060
$B(2, .25)$	.035	.034	.047	.066
M	.033	.030	.040	.058

Note. Within each condition, 1,000 samples of data were generated with $N = 100$, $k = 12$, and a target correlation matrix based on correlated factors. Equal numbers of variables loaded onto each factor such that within a factor, variables correlated with one another at $r = .70$; all other variable pairs correlated at $r = .30$. The median value of the RMSR correlation is reported; $SE = .002$ for all conditions.

$\chi^2(1)$, $B(2, .25)$, and $g = .30$, $h = .20$–100 samples of data were generated with $N = 10{,}000$. In each of these three configurations of data conditions, the program performed well: For $\chi^2(1)$ distributions, RMSR correlations ranged from .004 to .015 ($M = .008$, $SD = .002$); for $B(2, .25)$ distributions, RMSR correlations ranged from .004 to .010 ($M = .007$, $SD = .001$); and for $g = .30$, $h = .20$ distributions, RMSR correlations ranged from .006 to .024 ($M = .011$, $SD = .005$). Two more tests were similar to these three except that the variables' population distributions spanned a wide range of positive and negative skew. First, 32 variables were sampled from binary distributions with p ranging from .10 to .90 in equal increments. Second, 32 variables were sampled from g-and-h distributions with g ranging from $-.30$ to $+.30$ in equal increments (to vary skew) and $h = .20$ for each (to maintain high kurtosis values). These represent the most demanding conditions studied, and the program reproduced target correlation matrices well. In 100 samples with $N = 10{,}000$ for each condition, the RMSR correlations ranged from .013 to .015 ($M = .014$, $SD = .001$) for the binary data and from .004 to .020 ($M = .008$, $SD = .004$) for the g-and-h data.

Finally, the ability of the SI technique to reproduce the correlations observed in an actual data set was tested. Data were selected to pose substantial challenges to the `GenData` program: Scores were distributed asymmetrically across a small number of ordered categories. The data consist of 2,293 college students' responses to six items from the Beck Depression Inventory (BDI; Beck, Ward, Mendelson, Mock, & Erbaugh, 1961) that others had used to represent a relatively severe form of depression known as Involuntary Defeat Syndrome. There are only four response options for each BDI item (scored as 0, 1, 2, or 3), and within this sample the responses had nontrivial skew and kurtosis (see Table 2). The 15 correlations among these 6 variables ranged from $r = .05$ to $r = .30$ ($M = .17$, $SD = .07$). Ruscio, Ruscio, and Keane (2004) submitted these data to taxometric analysis, and here we examine the reproduction of the observed correlation matrix using distributions bootstrapped from those observed in the actual data.

Whether matching the sample size of the original data set or using a larger sample size in order to create a sample to treat as a population from which samples could be drawn, the target correlation matrix was reproduced with good precision. In 1,000 data sets generated with $N = 2{,}293$, the RMSR correlations ranged from .032 to .067 ($M = .050$, $SD = .006$). In 1,000 data sets generated with $N = 10{,}000$, the RMSR correlations ranged from .022 to .038 ($M = .032$, $SD = .003$). Even though these values are larger than for the other conditions studied here, they are still sufficiently small for all or most research purposes. In each of these 2,000 samples of simulated data, the GFI was at least .95. For a more detailed discussion of the use of the SI technique to reproduce the characteristics of a unique sample of empirical data, as well as extensive tests of its performance in doing so, see Ruscio (2007), Ruscio and Marcus (2007), or Ruscio et al. (2006, 2007).

DISCUSSION

The TC technique for generating multivariate nonnormal data operates by calculating an intermediate correlation matrix such that a target correlation matrix is reproduced as multivariate normal data are transformed to simulate data drawn population distributions with specified moments. Because this approach executes quickly, it is not surprising that it has become popular since the Vale-Maurelli (1983) implementation was introduced. In contrast, the SI technique operates by sampling data from specified population distributions and determining an intermediate correlation matrix through an iterative, trial-and-error process. This is more computationally intensive than the TC technique, but it offers a number of potential advantages that can offset this burden. Because the SI technique samples scores rather than transforms data, there are no boundary conditions on the moments of population distributions one can use, and some or all of these moments may be undefined. One can choose to sample from a particular distribution even when others have the same moments, and one can sample from discrete distributions—provided that scores can be rank-ordered. The SI technique can be used to generate data from specified populations or to reproduce the distributions observed in a unique sample of empirical data. In any of these procedural variants, one can control the number of latent factors to use in reproducing the target correlation matrix. Finally, one can use the SI technique to generate one or more data sets such that each reproduces the target correlation matrix or to create one large data set that will be treated as a population from which samples will be drawn. Both of these methods incorporate normal sampling error in sampling scores from population distributions, and the latter method also incorporates normal sampling error in the reproduction of the correlation matrix when one draws samples from the large data set that is treated as a population.

Simulations demonstrated that the SI technique, as implemented in the `GenData` program, reproduces target correlation matrices well across a wide range of population distributions, including continuous and discrete distributions that deviated substantially from normality; at different sample sizes, numbers of variables, and numbers of latent factors; and when providing a unique sample of empirical data. Even when especially challenging data conditions were specified (e.g., many variables whose distributions deviated substantially from normality and that loaded onto multiple factors), the program was able to reproduce correlations with good precision when even as few as 10,000 cases were generated to represent a finite population from which samples could then be drawn. Results suggest that investigators can construct fairly small samples (e.g., with $N = 40$ under the conditions of our first series of simulations), each of which reproduces a target correlation matrix, even when population distributions deviate considerably from normality. The SI technique offers researchers exceptional flexibility and control in simulating data, and the `GenData` program appears to be an implementation of the technique that allows researchers to realize its many potential advantages. These tools may be useful in many situations that require the generation of multivariate nonnormal data.

ACKNOWLEDGMENTS

We thank Mati Meron for thoughtful comments on an earlier draft of this article as well as an anonymous reviewer who helped us to improve the efficiency and readability of the program code in a number of ways.

REFERENCES

Beck, A. T., Ward, C. H., Mendelson, M., Mock, J., & Erbaugh, J. (1961). An inventory for measuring depression. *Archives of General Psychiatry, 4,* 53–63.

Efron, B., & Tibshirani, R. J. (1993). *An introduction to the bootstrap*. Boca Raton, FL: CRC Press.

Fleishman, A. I. (1978). A method for simulating non-normal distributions. *Psychometrika, 43,* 521–532.

Headrick, T. C. (2002). Fast fifth-order polynomial transforms for generating univariate and multivariate nonnormal distributions. *Computational Statistics and Data Analysis, 40,* 685–711.

Headrick, T. C., & Kowalchuk, R. K. (2007). The power method transformation: Its probability density function, distribution function, and its further use for fitting data. *Journal of Statistical Computation and Simulation, 77,* 229–249.

Headrick, T. C., & Mugdadi, A. (2006). On simulating multivariate nonnormal distributions from the generalized lambda distribution. *Computational Statistics and Data Analysis, 50,* 3343–3353.

Headrick, T. C., & Sawilowsky, S. S. (1999). Simulating correlated multivariate nonnormal distributions: Extending the Fleishman power method. *Psychometrika, 64,* 25–35.

Hoaglin, D. C. (1985). Summarizing shape numerically: The *g*-and-*h* distributions. In D. C. Hoaglin, F. Mosteller, & J. W. Tukey (Eds.), *Exploring data, tables, trends, and shapes* (pp. 461-511). New York: Wiley.

Kaiser, H. F. (1960). The application of electronic computers to factor analysis. *Educational and Psychological Measurement, 20,* 141–151.

McDonald, R. P. (1999). *Test theory: A unified treatment*. Mahwah, NJ: Erlbaum.

McLachlan, G. J., & Peel, D. (2001). *Finite mixture models*. New York: Wiley.

Montanelli, R. G., Jr., & Humphreys, L. G. (1976). Latent roots of random data correlation matrices with squared multiple correlations on the diagonal: A Monte Carlo study. *Psychometrika, 41,* 341–347.

Nagahara, Y. (2004). A method of simulating multivariate nonnormal distributions by the Pearson distribution system and estimation. *Computational Statistics and Data Analysis, 47,* 1–29.

Reise, S. P., Waller, N. G., & Comrey, A. L. (2000). Factor analysis and scale revision. *Psychological Assessment, 12,* 287–297.

Ruscio, J. (2007). Taxometric analysis: An empirically-grounded approach to implementing the method. *Criminal Justice and Behavior, 34,* 1588–1622.

Ruscio, J., Haslam, N., & Ruscio, A. M. (2006). *Introduction to the taxometric method: A practical guide*. Mahwah, NJ: Erlbaum.

Ruscio, J., & Marcus, D. K. (2007). Detecting small taxa using simulated comparison data: A reanalysis of Beach, Amir, and Bau's (2005) data. *Psychological Assessment, 19,* 241–246.

Ruscio, J., & Ruscio, A. M., & Keane, T. M. (2004). Using taxometric analysis to distinguish a small latent taxon from a latent dimension with positively skewed indicators: The case of Involuntary Defeat Syndrome. *Journal of Abnormal Psychology, 113,* 145–154.

Ruscio, J., Ruscio, A. M., & Meron, M. (2007). Applying the bootstrap to taxometric analysis: Generating empirical sampling distributions to help interpret results. *Multivariate Behavioral Research, 42,* 349–386.
Vale, D. C., & Maurelli, V. A. (1983). Simulating multivariate nonnormal distributions. *Psychometrika, 48,* 465–471.
Wichura, M. J. (1988). Algorithm AS 241: The percentage points of the normal distribution. *Applied Statistics, 37,* 477–484.

APPENDIX

R Program Code for GenData

The GenData program code contains comments that connect to steps described in the text. In this implementation, the user supplies a data set and the program reproduces the variables' score distributions and correlation matrix. Alternative ways to implement the program are described in comments in the second and third blocks of the code. The supplementary Factor.Analysis function used by GenData is provided as well. Both programs can be downloaded at http://www.taxometricmethod.com

This code should reside in a text file with a .R extension for recognition by the "File / Source R Code . . . " menu item in R. Once processed in this way, the user treats it as a macro by calling the function and storing its output in a new data object. For example, if one had a data object named x, the following command in R would call the GenData program and assign the resulting simulated data set to an object named y:

$$\texttt{y} <- \texttt{GenData(Supplied.Data} = \texttt{x)}$$

This example leaves all options at the program defaults, but a user can specify alternative values of N.Factors, Max.Trials, Initial.Multiplier, and/or seed (see the description of the program in the text for details).

```
################################################################################################################
GenData <- function(Supplied.Data, N.Factors = 0, Max.Trials = 5, Initial.Multiplier = 1, seed = 0)
{

# Initialize variables and (if applicable) set random number seed (step 1) -------------------------------------

   N <- dim(Supplied.Data)[1]                       # Number of cases
   k <- dim(Supplied.Data)[2]                       # Number of variables
   Data <- matrix(0, nrow = N, ncol = k)            # Matrix to store the simulated data
   Distributions <- matrix(0, nrow = N, ncol = k)   # Matrix to store each variable's score distribution
   Iteration <- 0                                   # Iteration counter
   Best.RMSR <- 1                                   # Lowest RMSR correlation
   Trials.Without.Improvement <- 0                  # Trial counter
   if (seed != 0) set.seed(seed)                    # If user specified a nonzero seed, set it

# Generate distribution for each variable (step 2) -------------------------------------------------------------

   for (i in 1:k)
      Distributions[,i] <- sort(sample(Supplied.Data[,i], N, replace = TRUE))
```

```
#     This implementation of GenData bootstraps each variable's score distribution from a supplied data set.
#     Users should modify this block of the program, as needed, to generate the desired distribution(s).
#
#     For example, to sample from chi-square distributions with 2 df, replace the 2nd line in this block with:
#        Distributions[,i] <- sort(rchisq(N, df = 2))
#
#     Or, one can drop the loop and use a series of commands that samples variables from specified populations:
#        Distributions[,1] <- sort(rnorm(N, 0, 1))          # Standard normal distribution
#        Distributions[,2] <- sort(runif(N, 0, 1))          # Uniform distribution ranging from 0 - 1
#        Distributions[,3] <- sort(rlnorm(N, 0, 1))         # Log-normal distribution, log scale M = 0, SD = 1
#        Distributions[,4] <- sort(rexp(N, rate = 1))       # Exponential distribution with rate = 1
#        Distributions[,5] <- sort(rpois(N, lambda = 4))    # Poisson distribution with lambda = 4
#        Distributions[,6] <- sort(rbinom(N, 10, .25)       # Binominal distribution, size = 10 and p = .25
#        Distributions[,7] <- sort(rbinom(N, 2, .25)        # Binary distribution with p = .25
#
#     All of the commands shown above draw random samples from specified population distributions.  As an
#     alternative, one can reproduce distributions without sampling error.  For example, working with a
#     supplied data set, one can replace the 2nd line in this block with:
#        Distributions[,i] <- Supplied.Data[,i]

#     Alternatively, idealized distributions can be reproduced.  For example, uniform quantiles can be
#     created and used to generate data from common distributions:
#        Uniform.Quantiles <- seq(from = 0, to = 1, length = (N + 2))[2:(N + 1)]   # quantiles 0, 1 dropped
#        Distributions[,1] <- qnorm(Uniform.Quantiles, 0, 1)      # Standard normal distribution
#        Distributions[,2] <- qunif(Uniform.Quantiles, 0, 1)      # Uniform distribution ranging from 0 to 1
#        Distributions[,3] <- qchisq(Uniform.Quantiles, df = 2)   # Chi-square distribution with 2 df
#
#     Note that when score distributions are generated from specified populations rather than bootstrapped from
#     a supplied data set, the user must provide the target correlation matrix (see the next block).  This is
#     true regardless of whether the distributions incorporate sampling error.

# Calculate and store a copy of the target correlation matrix (step 3) -----------------------------------------

   Target.Corr <- cor(Supplied.Data)
   Intermediate.Corr <- Target.Corr

#     This implementation of GenData calculates the target correlation matrix from a supplied data set.
#     Alternatively, the user can modify the program to generate data with user-defined sample size, number of
#     variables, and target correlation matrix by redefining the function as follows:
#        GenData <- function(N, k, Target.Corr, N.Factors = 0, Max.Trials = 5, Initial.Multiplier = 1, seed = 0)
#     In this case, one would also remove the program lines that calculate N, k, and Target.Corr.
#     To generate data in which variables are uncorrelated, one would remove the ŞsortŤ function from step 2
#        and terminate the program before step 3 begins by returning the Distributions object as the data set.

# If number of latent factors was not specified, determine it through parallel analysis (step 4) ---------------

   if (N.Factors == 0)
   {
      Eigenvalues.Observed <- eigen(Intermediate.Corr)$values
      Eigenvalues.Random <- matrix(0, nrow = 100, ncol = k)
      Random.Data <- matrix(0, nrow = N, ncol = k)
      for (i in 1:100)
      {
         for (j in 1:k)
            Random.Data[,j] <- sample(Distributions[,j], size = N, replace = TRUE)
         Eigenvalues.Random[i,] <- eigen(cor(Random.Data))$values
      }
      Eigenvalues.Random <- apply(Eigenvalues.Random, 2, mean) # calculate mean eigenvalue for each factor
      N.Factors <- max(1, sum(Eigenvalues.Observed > Eigenvalues.Random))
   }

# Generate random normal data for shared and unique components, initialize factor loadings (steps 5, 6) --------

   Shared.Comp <- matrix(rnorm(N * N.Factors, 0, 1), nrow = N, ncol = N.Factors)
   Unique.Comp <- matrix(rnorm(N * k, 0, 1), nrow = N, ncol = k)
   Shared.Load <- matrix(0, nrow = k, ncol = N.Factors)
   Unique.Load <- matrix(0, nrow = k, ncol = 1)

# Begin loop that ends when specified number of iterations pass without improvement in RMSR correlation --------

   while (Trials.Without.Improvement < Max.Trials)
   {
      Iteration <- Iteration + 1

# Calculate factor loadings and apply to reproduce desired correlations (steps 7, 8) ---------------------------
```

```
      Fact.Anal <- Factor.Analysis(Intermediate.Corr, Corr.Matrix = TRUE, N.Factors = N.Factors)
      if (N.Factors == 1) Shared.Load[,1] <- Fact.Anal$loadings
      else Shared.Load <- Fact.Anal$loadings
      Shared.Load[Shared.Load > 1] <- 1
      Shared.Load[Shared.Load < -1] <- -1
      if (Shared.Load[1,1] < 0) Shared.Load <- Shared.Load * -1
      Shared.Load.sq <- Shared.Load * Shared.Load
      for (i in 1:k)
         if (sum(Shared.Load.sq[i,]) < 1) Unique.Load[i,1] <- (1 - sum(Shared.Load.sq[i,]))
         else Unique.Load[i,1] <- 0
      Unique.Load <- sqrt(Unique.Load)
      for (i in 1:k)
         Data[,i] <- (Shared.Comp %*% t(Shared.Load))[,i] + Unique.Comp[,i] * Unique.Load[i,1]
      # the %*% operator = matrix multiplication, and the t() function = transpose (both used again in step 13)

# Replace normal with nonnormal distributions (step 9) --------------------------------------------------------

      for (i in 1:k)
      {
         Data <- Data[sort.list(Data[,i]),]
         Data[,i] <- Distributions[,i]
      }


# Calculate RMSR correlation, compare to lowest value, take appropriate action (steps 10, 11, 12) -------------

      Reproduced.Corr <- cor(Data)
      Residual.Corr <- Target.Corr - Reproduced.Corr
      RMSR <- sqrt(sum(Residual.Corr[lower.tri(Residual.Corr)] * Residual.Corr[lower.tri(Residual.Corr)]) /
              (.5 * (k * k - k)))
      if (RMSR < Best.RMSR)
      {
         Best.RMSR <- RMSR
         Best.Corr <- Intermediate.Corr
         Best.Res <- Residual.Corr
         Intermediate.Corr <- Intermediate.Corr + Initial.Multiplier * Residual.Corr
         Trials.Without.Improvement <- 0
      }
      else
      {
         Trials.Without.Improvement <- Trials.Without.Improvement + 1
         Current.Multiplier <- Initial.Multiplier * .5 ^ Trials.Without.Improvement
         Intermediate.Corr <- Best.Corr + Current.Multiplier * Best.Res
      }
   }  # end of the while loop

# Construct the data set with the lowest RMSR correlation (step 13) -------------------------------------------

   Fact.Anal <- Factor.Analysis(Best.Corr, Corr.Matrix = TRUE, N.Factors = N.Factors)
   if (N.Factors == 1) Shared.Load[,1] <- Fact.Anal$loadings
   else Shared.Load <- Fact.Anal$loadings
   Shared.Load[Shared.Load > 1] <- 1
   Shared.Load[Shared.Load < -1] <- -1
   if (Shared.Load[1,1] < 0) Shared.Load <- Shared.Load * -1
   Shared.Load.sq <- Shared.Load * Shared.Load
   for (i in 1:k)
      if (sum(Shared.Load.sq[i,]) < 1) Unique.Load[i,1] <- (1 - sum(Shared.Load.sq[i,]))
      else Unique.Load[i,1] <- 0
   Unique.Load <- sqrt(Unique.Load)
   for (i in 1:k)
      Data[,i] <- (Shared.Comp %*% t(Shared.Load))[,i] + Unique.Comp[,i] * Unique.Load[i,1]
   Data <- apply(Data, 2, scale) # standardizes each variable in the matrix
   for (i in 1:k)
   {
      Data <- Data[sort.list(Data[,i]),]
      Data[,i] <- Distributions[,i]
   }

# Report the results and return the simulated data set (step 14) ----------------------------------------------

   Iteration <- Iteration - Max.Trials
   cat("\nN =",N,", k =",k,",",Iteration,"Iterations,",N.Factors,"Factors, RMSR r =",round(Best.RMSR,3),"\n")
   return(Data)
}
```

```
################################################################################################################
Factor.Analysis <- function(Data, Corr.Matrix = FALSE, Max.Iter = 50, N.Factors = 0)
{
   Data <- as.matrix(Data)
   k <- dim(Data)[2]
   if (N.Factors == 0) N.Factors <- k
   if (!Corr.Matrix) Cor.Matrix <- cor(Data)
   else Cor.Matrix <- Data
   Criterion <- .001
   Old.H2 <- rep(99, k)
   H2 <- rep(0, k)
   Change <- 1
   Iter <- 0
   Factor.Loadings <- matrix(nrow = k, ncol = N.Factors)
   while ((Change >= Criterion) & (Iter < Max.Iter))
   {
      Iter <- Iter + 1
      Eig <- eigen(Cor.Matrix)
      L <- sqrt(Eig$values[1:N.Factors])
      for (i in 1:N.Factors)
         Factor.Loadings[,i] <- Eig$vectors[,i] * L[i]
      for (i in 1:k)
         H2[i] <- sum(Factor.Loadings[i,] * Factor.Loadings[i,])
      Change <- max(abs(Old.H2 - H2))
      Old.H2 <- H2
      diag(Cor.Matrix) <- H2
   }
   if (N.Factors == k) N.Factors <- sum(Eig$values > 1)
   return(list(loadings = Factor.Loadings[,1:N.Factors], factors = N.Factors))
}
```