Where do Multivariate Normal Samples Come from?

Paul E. Johnson <pauljohn@ku.edu>
Center for Research Methods and Data Analysis
University of Kansas
Lawrence, Kansas 66045

March 8, 2017

This note is about details of simulation of draws from a multivariate normal distribution. It reviews the mathematical formulation of the problem, some matrix terminology, and compares software implementations. It explores a basic question that most social scientists never consider, "Where do multivariate normal samples come from?" The essay compares Stata (StataCorp, 2015b) functions, `corr2data` and `drawnorm`, and the `mvrnorm` function in the MASS package (Venables & Ripley, 2002) for R (R Core Team, 2015). A number of matrix algebra details are written out in detail.[1]

In an essay named, "New Estimates for Propensity Score Analysis Monte Carlo Simulations", we discuss a replication of Monte Carlo simulation estimates reported in Guo & Fraser (2015). Some differences between simulations prepared in R (R Core Team, 2015) and Stata 14 (StataCorp, 2015b), as reported by Guo and Fraser, are considered.

This essay is a record of the technical side of the project. In the process of discovering the source of the differences, a good deal of effort was invested developing an understanding of the technical foundations of the competing methods of creating simulated multivariate normal data. Students in research methodology can benefit from an inspection of this material in four ways. First, an abstract algorithm for simulating draws from a multivariate normal (MVN) distribution is laid out, step by step. Second, competing methods of decomposing a matrix $\mathbf{X}$ and the associated cross product matrix $\mathbf{X}^T\mathbf{X}$ are considered. Third, coding differences between R and Stata implementations of the MVN are considered. Finally, the different characteristics of "empirically standardized" MVN data are investigated. The Stata documentation cautions readers against the usage of their function called `corr2data`. We consider the implications of mistakenly using this function when one ought to be using a function to draw MVN samples instead.

The first sections offer a formal definition of the MVN distribution and its parameters along with some mathematical details. A 5 step procedure for generating MVN samples is described. The software implementations in Stata and R are compared. The final section focuses on Stata's `corr2data` function, which may have been used accidentally in simulations that should instead have used drawnorm.

This note presumes a basic training in statistics, random variables[2], and elementary matrix algebra as it is used in regression analysis. Most of the other details are given a basic description when they arise.

1 Normal and Multivariate Normal Distributions

Basic surveys of the normal distribution, $N(\mu, \sigma^2)$, and the multivariate normal distribution, $MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, are available on my Website[3].

The probability density function for the one variable model is written

$$\text{Univariate}: f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \text{ or } \frac{1}{(2\pi)^{1/2}\sigma} e^{-\frac{1}{2}(x-\mu)\sigma^{-1}(x-\mu)}. \tag{1}$$

A draw from that probability process is referred to as $x \sim N(\mu, \sigma^2)$. The parameter μ determines the center point of the distribution's values, while σ^2 is the dispersion. The density function has the property that the mostly likely outcome is also the expected value, which happens to equal the parameter μ. For that reason, the parameter μ is sometimes simply referred to as the expected value, or the mean. The dispersion parameter is often referred to as the variance. I try to avoid the name "population" to refer to this process; that causes more confusion than clarity for readers. Instead, refer to this a data generating process, and μ and σ are the "true" or "parametric" values. Estimates from samples are distinguished with hats, $\hat{\mu}$ and $\hat{\sigma}$.

We write $\mathbf{x} \sim MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to refer to a column vector that is drawn from the multivariate normal distribution, $MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma})$[4]

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \sim MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = MVN\left(\begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & \sigma_{12} & & \sigma_{1p} \\ \sigma_{12} & \sigma_2^2 & & \sigma_{2p} \\ & & \ddots & \\ \sigma_{1p} & \sigma_{2p} & & \sigma_p^2 \end{bmatrix} \right). \tag{2}$$

The probability density function (PDF) for the MVN is quite similar to the formula for the one dimensional model.

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} e^{\frac{-1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}. \tag{3}$$

The probability of a given outcome depends on the parameters,

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & & \sigma_{1p} \\ \sigma_{12} & \sigma_2^2 & & \sigma_{2p} \\ & & \ddots & \\ \sigma_{1p} & \sigma_{2p} & & \sigma_p^2 \end{bmatrix}. \tag{4}$$

[2] "Distribution Overview: Probability by the Seat of the Pants", http://pj.freefaculty.org/guides/stat/Distributions/DistributionOverview/DistributionReview.pdf

[3] "Normal Distribution", http://pj.freefaculty.org/guides/stat/Distributions/DistributionWriteups/Normal/Normal-01.pdf.
"The Multivariate Normal Distribution", http://pj.freefaculty.org/guides/stat/Distributions/DistributionWriteups/NormalMultivariate/NormalMultivariate.pdf

[4] To save space on a written page, we often refer the transpose, $\mathbf{x}^T$, which is a row vector, or, equivalently $\mathbf{x} = (x_1, x_2, \ldots x_p)^T$.

The matrix $\mathbf{\Sigma}$ is also known in the literature as the "variance-covariance matrix" or the "covariance matrix".

The similarity of MVN to the one-dimensional normal is more apparent if we write the one variable model's density as

$$f(x) = \frac{1}{(2\pi)^{1/2}\sigma}\, e^{-\frac{1}{2}(x-\mu)\sigma^{-1}(x-\mu)}. \tag{5}$$

If $p = 1$, of course, the two density functions are the same (both one dimensional).

2 Understanding the Variance Matrix $\mathbf{\Sigma}$

Most students do not have trouble appreciating the fact that the vector of means for the individual elements, $(\mu_1, \ldots, \mu_p)^T$, provides the center points (modes, means) of the individual components of the random draw. They do, however, have difficulty understanding the variance matrix $\mathbf{\Sigma}$.

One source of difficulty is a notational incongruity. In the literature, we refer to variance as σ^2 (sigma squared) and the standard deviation as σ (sigma). In contrast, in a multivariate normal model, we refer to the variance matrix simply as $\mathbf{\Sigma}$ (bold-faced upper-case sigma), whereas it would seem more natural to refer to it as $\mathbf{\Sigma}^2$.

A nice way to begin study the variance matrix is to set all of the non-diagonal elements to zero ($\sigma_{ij} = 0$),

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & & 0 \\ 0 & \sigma_2^2 & & 0 \\ & & \ddots & \\ 0 & 0 & & \sigma_p^2 \end{bmatrix} \tag{6}$$

As one can see in (6), the elements on the main diagonal (σ_i^2) are variance parameters. Notice the incongrous notation: *Sigma* (on the left) is equal to a diagonal of *sigma*s squared.

A draw from MVN with that $\mathbf{\Sigma}$ matrix would provide uncorrelated elements. We can see that the square root of each element in the diagonal ($\sqrt{\sigma_i^2} = \sigma_i$) looks an awful lot like the standard deviation of an individual variable. If we took draws from MVN with this variance, we would essentially have separate columns. We could look at the j'th element of $\mathbf{x}$ in isolation and the value would be distributed with standard deviation σ_j.

Lets collect the standard deviations in a vector $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_p)^T$. If we place the standard deviation values along the diagonal, as in the following, we have a matrix that might be thought of as a square root of $\mathbf{\Sigma}$:

$$\mathbf{\Sigma}^{1/2} = \begin{bmatrix} \sigma_1 & 0 & & 0 \\ 0 & \sigma_2 & & 0 \\ & & \ddots & \\ 0 & 0 & & \sigma_p \end{bmatrix} = diag(\boldsymbol{\sigma}) \tag{7}$$

Note that it is a matrix square root, in the sense that

$$\mathbf{\Sigma} = \mathbf{\Sigma}^{1/2}\mathbf{\Sigma}^{1/2}. \tag{8}$$

This will turn out to be a key idea in the generation of multivariate normal random. We are able to state conditions under which a square root of the variance $\mathbf{\Sigma}$ exists.

2.1 About the off-diagonal elements, σ_{ij}.

Generally, the off-diagonal elements in $\mathbf{\Sigma}$ are not equal to 0. Those off diagonal elements, σ_{ij} in (4), are commonly called covariance parameters. A positive value of σ_{ij} means that elements x_i and x_j "go together", in the sense that if x_i is high, then x_j is also likely to be high. A negative value means that when x_i is high, then x_j is low.

There are p^2 elements in $\mathbf{\Sigma}$ (it is a $p \times p$ matrix). We are *not free to set all of them however we like*. There are logical and mathematical restrictions on the values that must be respected. Obviously, the main diagonal elements σ_i^2 must be positive (they are variances). In addition, as we see in the next sections, $\mathbf{\Sigma}$ is symmetric and positive definite.

2.1.1 $\mathbf{\Sigma}$ is symmetric.

The values of $\mathbf{\Sigma}$ above and below the main diagonal must be the same, $\sigma_{ij} = \sigma_{ji}$. Thus, $\mathbf{\Sigma} = \mathbf{\Sigma}^T$. As soon as we define variance, the proof that $\mathbf{\Sigma}$ is symmetric will fall out without any effort.

To define variance, it is necessary to understand the concept of expected value. In a one variable probability model, the expected value is $E[x] = \mu$ and the variance is defined as the expected value of the squared deviation of observed scores around the expected value, $E[(x - E[x])^2] = E[(x-\mu)^2] = \sigma^2$. Expected value is a probability weighted sum of possible outcomes for a variable. In the normally distributed variables, μ is the average and σ^2 is the diversity of scores likely to be observed.

The multivariate model uses vector multiplication to define variance. Replace the one variable expression $(x - E[x])$ with the multivariate $(\mathbf{x} - E[\mathbf{x}])$. I'll be explicit:

$$\begin{aligned}
\mathbf{\Sigma} &= E[(\mathbf{x} - E[\mathbf{x}])((\mathbf{x} - E[\mathbf{x}])^T] \qquad (9)\\
&= E\left[\begin{bmatrix} x_1 - E[x_1] \\ x_2 - E[x_2] \\ \vdots \\ x_p - E[x_p] \end{bmatrix} \begin{bmatrix} x_1 - E[x_1], & x_2 - E[x_2], & \ldots, & x_p - E[x_p] \end{bmatrix}\right] \\
&= E\begin{bmatrix} (x_1 - E[x_1])^2 & (x_1 - E[x_1])(x_2 - E[x_2]) & (x_1 - E[x_1])(x_3 - E[x_3]) & \cdots & (x_1 - E[x_1])(x_p - E[x_p]) \\ (x_1 - E[x_1])(x_2 - E[x_2]) & (x_2 - E[x_2])^2 & (x_2 - E[x_2])(x_3 - E[x_3]) & & \vdots \\ (x_1 - E[x_1])(x_3 - E[x_3]) & (x_2 - E[x_2])(x_3 - E[x_3]) & \ddots & & \vdots \\ \ddots & & & \ddots & \vdots \\ (x_1 - E[x_1])(x_p - E[x_p]) & & & & (x_p - E[x_p])^2 \end{bmatrix}.
\end{aligned}$$

The expected value of a matrix is the expected value of each individual element. Before writing that out, lets simplify by replacing $E[x_i]$ with μ_i:

$$= \begin{bmatrix} E[(x_1 - \mu_1)^2] & E[(x_1 - \mu_1)(x_2 - \mu_2)] & E[(x_1 - \mu_1)(x_p - \mu_p)] \\ E[(x_1 - \mu_1)(x_2 - \mu_2)] & E[(x_2 - \mu_2)^2] & \\ & \ddots & \\ E[(x_1 - \mu_1)(x_p - \mu_p)] & & E[(x_p - \mu_p)^2] \end{bmatrix} \qquad (10)$$

From the construction of $\mathbf{\Sigma}$, it should be apparent that $\mathbf{\Sigma}$ is a symmetric matrix[5]: $E[(x_i - \mu_i)(x_j - \mu_j)] = E[(x_j - \mu_j)(x_i - \mu_i)]$.

The impact of symmetry is that it reduces the number of "unrestricted elements" in $\mathbf{\Sigma}$. There are $(n-1)n/2$ elements above the main diagonal. Once they are specified, they must be mirrored below.

[5]In elementary mathematics, we know that $a \cdot b = b \cdot a$.

2.1.2 $\boldsymbol{\Sigma}$ is positive definite

This section is about the restrictions that flow from the idea that the last part of the exponent in equation (3),

$$(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}), \tag{11}$$

must be positive. We explore the idea that $\boldsymbol{\Sigma}^{-1}$, and hence $\boldsymbol{\Sigma}$, must be "positive definite". This discussion is a little bit esoteric, but this concept/terminology pervades the literature on multivariate random numbers and there is simply no way to avoid it.

Here's one intuition. The distance between two points cannot be negative (Remember Pythagoras: $a^2+b^2=c^2$). The difference $(\mathbf{x}-\boldsymbol{\mu})$ is the distance between $\mathbf{x}$ and $\boldsymbol{\mu}$. The squared distance from $\mathbf{x}$ to $\boldsymbol{\mu}$ is

$$(\mathbf{x}-\boldsymbol{\mu})^T(\mathbf{x}-\boldsymbol{\mu}) = (x_1-\mu_1)^2+(x_2-\mu_2)^2+\cdots+(x_p-\mu_p)^2, \tag{12}$$

an application of the Pythagorean theorem. Squared values are always positive. As long as the two points are indeed at different positions, the distance between them has to be greater than zero. This is not something to be derived. It is a property to be assumed:

$$(\mathbf{x}-\boldsymbol{\mu})^T(\mathbf{x}-\boldsymbol{\mu}) > 0 \, if \, \mathbf{x}\neq\boldsymbol{\mu}. \tag{13}$$

In (11) we have a weighted distance matrix, where the inverse of $\boldsymbol{\Sigma}$ appears between $(\mathbf{x}-\boldsymbol{\mu})^T$ and $(\mathbf{x}-\boldsymbol{\mu})$. The weighted distance, well, has to be positive, unless the two points we are comparing are at the exactly same position. So we restate the "distance must be positive" idea:

$$(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) > 0. \tag{14}$$

The expression in (14) has a formal name: $\boldsymbol{\Sigma}^{-1}$ is "positive definite". A matrix, such as $\boldsymbol{\Sigma}^{-1}$, is positive definite if, for any non-zero vector $\mathbf{z}$,

$$\mathbf{z}^T\boldsymbol{\Sigma}^{-1}\mathbf{z} > 0. \tag{15}$$

If the inequality allows "equal to", $\geq$, then $\boldsymbol{\Sigma}^{-1}$ is said to be positive semi-definite. This generally indicates that one column can be reproduced as a weighted sum of the other columns; such a column adds no information.

Admittedly, this seems esoteric because we are working hard to justify the simple idea that "something squared is a positive value". Not only does it seem esoteric, it seems unhelpful. We want information about restrictions on $\boldsymbol{\Sigma}$, but we are giving restrictions on $\boldsymbol{\Sigma}^{-1}$. However, the effort is not wasted.

Here is an important fact: if $\boldsymbol{\Sigma}$ is positive definite, then $\boldsymbol{\Sigma}^{-1}$ is positive definite. This is fairly easy to prove. Suppose $\mathbf{x}\boldsymbol{\Sigma}\mathbf{x} > 0$ and $\boldsymbol{\Sigma}$ is symmetric and invertible. Let $\mathbf{y}=\boldsymbol{\Sigma}\mathbf{x}$. Note that $\mathbf{y}^T=\mathbf{x}^T\boldsymbol{\Sigma}^T$. Thus $\mathbf{y}^T\boldsymbol{\Sigma}^{-1}\mathbf{y}=\mathbf{x}^T\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}\mathbf{x}=\mathbf{x}^T\boldsymbol{\Sigma}\mathbf{x}$. The signs of the first and third elements must be the same, so if $\boldsymbol{\Sigma}$ is positive definite, then so is $\boldsymbol{\Sigma}^{-1}$.

2.2 Checking if a Matrix is Positive Definite: Eigenvalues and Eigenvectors

In many linear algebra books (for example, Golub and Van Loan, 1996, p. 141-2), there will be a list of equivalent properties that link positive definiteness of a matrix to a number of other qualities. These other qualities give us ideas about how to check whether a matrix is positive definite, or how we might manufacture a positive definite matrix..

Theorem 1. *(Positive Definite Matrix Properties) The following are equivalent:*

1. $\mathbf{x}^T\mathbf{\Sigma}\mathbf{x} > 0$ for all $\mathbf{x} \neq 0$.

2. If a matrix $\mathbf{S}$ has full column rank (the p columns are linearly independent), then the product $\mathbf{S}^T\mathbf{S}$ is a positive definite matrix.[6]

3. A positive definite matrix $\mathbf{\Sigma}$ can be decomposed into a product of matrices. The Cholesky root and the eigenvalue decompositions are explored below.

4. The eigenvalues of $\mathbf{\Sigma}$ are positive.

5. The determinants of principal submatrices (square submatrices beginning with row and column 1) are positive.

In property 4, we note that when $\mathbf{\Sigma}$ is positive definite, then the eigenvalues are all positive.

Eigenvalues are discussed in depth in a first course on linear algebra. This note cannot replace a thorough study of the material, but it might help students remember what they learned, or motivate them to study some more. Here is a nutshell definition of eigenvalue.

The eigenvalues (λ_j) and eigenvectors ($\mathbf{v}_j$) of a matrix are defined as the solutions of this equation

$$\mathbf{\Sigma}\mathbf{v}_j = \lambda_j\mathbf{v}_j \tag{16}$$

Heuristically, this says that the scaling effect that $\mathbf{\Sigma}$ exerts on a vector $\mathbf{v}_j$ can be summarized by a proportional rescaling $\lambda_j\mathbf{v}_j$. This is the sense in which the matrix $\mathbf{\Sigma}$ is characterized by λ_j and $\mathbf{v}_j$.

Eigenvalues are calculated as follows. Rearrange the definition as follows:

$$\begin{aligned} \mathbf{\Sigma}\mathbf{v}_j &= \lambda_j\mathbf{v}_j \\ \mathbf{\Sigma}\mathbf{v}_j - \lambda_j\mathbf{v}_j &= 0 \\ \mathbf{\Sigma}\mathbf{v}_j - \lambda_j\mathbf{I}\mathbf{v}_j &= 0 \\ (\mathbf{\Sigma} - \lambda_j\mathbf{I})\ \mathbf{v}_j &= 0 \end{aligned} \tag{17}$$

We want to ignore the trivial solution where $\mathbf{v}_j = 0$. Thus, when $\mathbf{v}_j \neq 0$, it must be that $(\mathbf{\Sigma} - \boldsymbol{\lambda}_\mathbf{j}\mathbf{I}) = 0$. There may be several solutions, the eigenvalues are always generally unique. A theorem in linear algebra states that the eigenvalues can be found as the solutions of the so-called characteristic equation, $det(\mathbf{\Sigma} - \lambda_j\mathbf{I}) = 0$, where *det* represents the determinant. When $\mathbf{\Sigma}$ is a well specified $p \times p$ variance matrix, then there will be p separate solutions of the characteristic equation, and hence p eigenvalues. After the eigenvalues are found, then the corresponding vectors are calculated.

In most software packages, the eigenvalues are stored in order of descending magnitude in $\boldsymbol{\lambda}^T = (\lambda_1, \lambda_2, \ldots, \lambda_p)$ and the eigenvectors are scaled so that $\mathbf{v}_j^T\mathbf{v}_j = 1$. (The scaling is done by calculating $\mathbf{v}_j^T\mathbf{v}_j$ for the un-scaled eigenvector and then replacing $\mathbf{v}_j$ with $\frac{1}{\mathbf{v}_j^T\mathbf{v}_j}\mathbf{v}_j$).

We use the symbol $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_p]$ to refer to a $p \times p$ matrix with columns of the scaled eigenvectors. The columns of $\mathbf{V}$ are all linearly independent from one another. That is to say, not only are they scaled so that their lengths are 1, but also $\mathbf{v}_j^T\mathbf{v}_k = 0$ for all $j \neq k$. Thus

$$\mathbf{V}^T\mathbf{V} = \mathbf{I} \tag{18}$$

[6]The number of columns in $\mathbf{S}$ determines the size of $\mathbf{S}^T\mathbf{S}$. If $\mathbf{S}$ is $n \times p$, then $\mathbf{S}^T\mathbf{S}$ is square, $p \times p$, and it is symmetric. The number of rows in $\mathbf{S}$ is not relevant to the final size of $\mathbf{S}^T\mathbf{S}$.

3 Rescaling Normal Random Variables

3.1 One Dimension: σ is a Scaling Parameter

Until recent software enhancements, it was very common that statistical packages (or spreadsheets) would offer to draw from a standard normal distribution $N(0,1)$, but not from a normal with other values for the expected value and variance. A draw from $N(0,1)$ can be re-scaled to match the desired probability model.

If we have a draw x from $N(0,1)$ but we wish we had a draw from $N(\mu,\sigma^2)$, where $\sigma^2 > 0$, we re-scale x:

$$y = \mu + \sigma \cdot x. \tag{19}$$

Note that x is multiplied by the standard deviation, σ, not the variance, σ^2. The parameter σ is a scaling factor, while the μ plays the role of a location parameter.

This pre-supposes that one has a high quality method to draw a simulated $N(0,1)$, of course, but that is fairly well worked out at this late date.

There are two technical claims worth emphasizing.

1. Given a random variable x with $E[x] = 0$ and $Var[x] = 1$, a weighted sum $y = \mu + \sigma x$ has $E[y] = \mu$ and $Var[y] = \sigma^2$. This claim is true for all random variables, whether x is normal or not.

2. y is normally distributed, $N(\mu,\sigma^2)$. The proof that this requires us to do some math (either a "change of variables" in the probability density function or a comparison of the moment generating functions).

3.2 The Multivariate Version of Re-scaling

The multivariate version of the rescaling exercise is as follows. This follows the argument in Scheuer and Stoller (1962, p. 278). Suppose $\mathbf{x}$ is a vector of p values drawn from an $MVN(0,\mathbf{I})$ process.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \sim MVN\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ & & & \ddots & \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \right) \tag{20}$$

This is no different from saying that each of the elements in $\mathbf{x}$ is drawn independently, $x_i \sim N(0,1)$.

We want to apply a transformation so that the result is

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix} \sim MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = MVN\left(\begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & \sigma_{12} & & \sigma_{1p} \\ \sigma_{12} & \sigma_2^2 & & \sigma_{2p} \\ & & \ddots & \\ \sigma_{1p} & \sigma_{2p} & & \sigma_p^2 \end{bmatrix} \right). \tag{21}$$

The correct transformation looks quite a bit like (19):

$$\mathbf{y} = \boldsymbol{\mu} + \mathbf{S}\mathbf{x}. \tag{22}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{bmatrix} + \begin{bmatrix} s_{11} & s_{12} & & s_{1p} \\ s_{21} & s_{22} & & s_{2p} \\ & & \ddots & \\ s_{p1} & s_{p2} & & s_{pp} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}.$$

where $\mathbf{S}$ is a square scaling matrix that plays the same role that σ played in the one variable model.

Clearly, the expected value of our candidate draw $\mathbf{y}$ is correct,

$$E[\mathbf{y}] = E[\boldsymbol{\mu} + \mathbf{S}\mathbf{x}] = \boldsymbol{\mu} + \mathbf{S}E[\mathbf{x}] = \boldsymbol{\mu}. \tag{23}$$

And the variance matrix of $\mathbf{y}$ is

$$Var[\mathbf{y}] = \mathbf{S}Var(x)\mathbf{S}^T. \tag{24}$$

Because $Var(\mathbf{x}) = \mathbf{I}$ and because $\mathbf{S}^T\mathbf{S}$ is symmetric,

$$Var[\mathbf{y}] = \mathbf{S}\mathbf{S}^T = \mathbf{S}^T\mathbf{S}. \tag{25}$$

As long as the scaling matrix $\mathbf{S}$ is chosen *very carefully*, so that $\mathbf{S}^T\mathbf{S} = \boldsymbol{\Sigma}$, then a big part of the work is finished. We have some encouragement in positive definite matrix property 2, which indicates that $\mathbf{S}^T\mathbf{S}$ is positive definite. We just don't know yet if $\mathbf{S}^T\mathbf{S}$ equals $\boldsymbol{\Sigma}$. That problem is discussed in the next section.

The next step is to establish the fact that $\mathbf{y}$ is multivariate normally distributed, $\mathbf{y} \sim MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. According to Greene (2008, p. 1015), "Any linear function of a vector of joint normally distributed variables is also normally distributed... Thus,

$$If\ \mathbf{x} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}),\ then\ \mathbf{A}\mathbf{x} + \mathbf{b} \sim N(\mathbf{A}\boldsymbol{\mu} + \mathbf{b},\ \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T)". \tag{26}$$

It is said that the normal distribution is *closed under addition*: if we put in a draw from a multivariate normal, and re-scale by adding or multiplying by real valued matrices, we get out a re-scaled, but still multivariate normal, random variable. Devroye (1986, p. 565-6) gives an argument for this based on the moment generating function. He adds, "Unfortunately, the only symmetric stable distribution with finite variance is the normal distribution.... Thus, the property that the normal distribution is closed under the operation 'linear combination' is what makes it so attractive to the user. If the user specifies non-normal marginals, the covariance structure is much more difficult to enforce" (Devroye 1986, p. 565).

3.3 Square Root of a Matrix

The number 9 has two possible square roots, 3 and -3. The square root of a number is not unique. In light of that, it should not come as a shock to learn that the square root of a matrix is not unique. It may come as a shock, however, to learn matrix square roots are grossly different from one another and there are many square roots.

Method 1: Cholesky decomposition

If all of the estimated eigenvalues are greater than zero, the Cholesky decomposition of $\boldsymbol{\Sigma}$ can be calculated. The Cholesky algorithm finds a upper triangular matrix $\mathbf{R}$ with this interesting property:

$$\boldsymbol{\Sigma} = \mathbf{R}^{\mathbf{T}} \times \mathbf{R} = \begin{bmatrix} r_{11} & 0 & 0 & 0 & 0 \\ r_{12} & r_{22} & 0 & 0 & 0 \\ r_{13} & r_{23} & r_{33} & 0 & 0 \\ & & & \ddots & 0 \\ r_{1p} & r_{1p} & r_{3p} & & r_{pp} \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1p} \\ & r_{22} & r_{23} & & r_{2p} \\ & & r_{33} & & \\ & & & \ddots & \\ & & & & r_{pp} \end{bmatrix}. \tag{27}$$

This is one way to get a "matrix square root." The lower triangular part, $\mathbf{R}^T$, can play the role of $\mathbf{S}$ in the rescaling equation (22).

How do we know that $\mathbf{R}^T$ can pass for $\mathbf{S}$? The definition in (27) indicates that $\mathbf{R}^T\mathbf{R}$ equals the variance matrix,$\mathbf{\Sigma}$.

Method 2: Eigen decomposition

When $\mathbf{\Sigma}$ is positive semi-definite (there is an eigenvalue equal to 0), the Cholesky decomposition is not possible. In that case, an eigen decomposition of $\mathbf{\Sigma}$ can be used. Recall the eigenvalue vector is $\boldsymbol{\lambda}^T = (\lambda_1, \lambda_2, \ldots, \lambda_p)$ and the scaled eigenvectors are the columns of $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_p]$.

We have not emphasized until now that the eigenvalue matrix $\mathbf{V}$ is an *orthonormal* matrix. That means the vectors are orthogonal to one another, $\mathbf{v}_i^T\mathbf{v}_j = 0$ (that's the "ortho" part), and they are re-scaled so their norms are are unity: $\mathbf{v}_j^T\mathbf{v}_j = 1$ (that's the "normal" part). Hence

Fact 2. *If* $\mathbf{V}$ *is an orthonormal matrix, then*

1. $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, and
2. $\mathbf{V}^{-1} = \mathbf{V}^T$.

The eigenvalues and eigenvectors can be used to write out a decomposition of $\mathbf{\Sigma}$. (This is also referred to as the spectral decomposition of $\mathbf{\Sigma}$). Begin with the definition in (16):

$$\begin{aligned} \mathbf{\Sigma}\mathbf{V} &= \mathbf{V}diag(\boldsymbol{\lambda}) \\ \mathbf{\Sigma}\mathbf{V}\mathbf{V}^T &= \mathbf{V}diag(\boldsymbol{\lambda})\mathbf{V}^T \\ \mathbf{\Sigma} &= \mathbf{V}diag(\boldsymbol{\lambda})\mathbf{V}^T. \end{aligned} \tag{28}$$

The function *diag* places a vector's values along the diagonal:

$$diag(\boldsymbol{\lambda}) = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_p \end{bmatrix}. \tag{29}$$

Since $\lambda_j \geq 0$, a real-valued square root of each eigenvalue exists, and we can write this as a product using $diag(\boldsymbol{\lambda})^{1/2}$:

$$diag(\boldsymbol{\lambda}) = \begin{bmatrix} \sqrt{\lambda_1} & 0 & 0 & 0 \\ 0 & \sqrt{\lambda_2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sqrt{\lambda_p} \end{bmatrix} \begin{bmatrix} \sqrt{\lambda_1} & 0 & 0 & 0 \\ 0 & \sqrt{\lambda_2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sqrt{\lambda_p} \end{bmatrix} = diag(\boldsymbol{\lambda})^{1/2} diag(\boldsymbol{\lambda})^{1/2} \tag{30}$$

This allows us to revise (28) into a format that helps us to see that we have square root of $\mathbf{\Sigma}$:

$$\begin{aligned} \mathbf{\Sigma} &= \mathbf{V}diag(\boldsymbol{\lambda})^{1/2}\ diag(\boldsymbol{\lambda})^{1/2}\mathbf{V}^T. \\ & \quad \mathbf{V}diag(\boldsymbol{\lambda})^{1/2}(\mathbf{V}diag(\boldsymbol{\lambda})^{1/2})^T \end{aligned} \tag{31}$$

The scaling matrix will be

$$\mathbf{S} = \mathbf{V}diag(\boldsymbol{\lambda})^{1/2} \tag{32}$$

because $\mathbf{S}\mathbf{S}^T = \mathbf{S}^T\mathbf{S} = \mathbf{\Sigma}$.

Is there any substantive interpretation for $\mathbf{S}$? It is clear that the square roots of the eigenvalues are being used to re-scale the columns of the eigenvector matrix $\mathbf{V}$.

$$\mathbf{S} = \begin{bmatrix} v_{11} & v_{12} & & v_{1p} \\ v_{21} & v_{22} & & v_{2p} \\ & & \ddots & \\ & & & v_{pp} \end{bmatrix} \begin{bmatrix} \sqrt{\lambda_1} & 0 & 0 & 0 \\ 0 & \sqrt{\lambda_2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sqrt{\lambda_p} \end{bmatrix} = \begin{bmatrix} \sqrt{\lambda_1}v_{11} & \sqrt{\lambda_2}v_{12} & & \sqrt{\lambda_p}v_{1p} \\ \sqrt{\lambda_1}v_{21} & \sqrt{\lambda_2}v_{22} & & \sqrt{\lambda_p}v_{2p} \\ & & \ddots & \\ & & & \sqrt{\lambda_p}v_{pp} \end{bmatrix} \tag{33}$$

3.4 Decompositions of X, rather than Σ

A different decomposition problem arises if the user has a "raw data matrix" $\mathbf{X}$. The usual statistics textbook recommends formula involving a cross product matrix $\mathbf{X}^T\mathbf{X}$, such as the ordinary least squares (OLS) regression vector, $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. Statistics students then travels into a field like numerical linear algebra, where the first thing they learn is "you were not taught proper formulas for digital calculation" (see Woods, 2006, for example). Following the advice in Golub & Van Loan (1996), we avoid forming $\mathbf{X}^T\mathbf{X}$ explicitly and do not try to "solve" $\mathbf{X}^T\mathbf{X}$ by explicit matrix inversion. No reasonable regression software tries to invert $(\mathbf{X}^T\mathbf{X})$ any more.

There are much more accurate ways to calculate theoretically important quantities like the inverse of $(\mathbf{X}^T\mathbf{X})^{-1}$. This is done with matrix factorization. In R, for example, there are two functions for principal components analysis. The function `princomp` uses the older style of less stable linear algebra based on the eigen decomposition of the centered variance matrix while the newer `prcomp` is carried out after decomposing the data matrix. The R help page for the `prcomp` mentions this difference, somewhat obliquely, "The calculation is done by a singular value decomposition of the (centered and possibly scaled) data matrix, not by using '`eigen`' on the covariance matrix. This is generally the preferred method for numerical accuracy" (R Core Team, 2015).

There are several different ways that can decompose a data matrix. The two most widely mentioned are the QR decomposition and the singular value decomposition (SVD). The QR and the SVD are different from the Cholesky and eigen decompositions because the latter approaches require the input data must be a square matrix, whereas QR and SVD can be applied to an $n \times p$ matrix.

Method 3. QR decomposition

The theoretical quantity $(\mathbf{X}^T\mathbf{X})$ can be calculated in a much more numerically accurate way as $(\mathbf{R}^T\mathbf{R})$, where $\mathbf{R}$ is an upper triangular matrix.

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1p} \\ 0 & r_{22} & & r_{2p} \\ 0 & 0 & \ddots & \\ 0 & 0 & 0 & r_{pp} \end{bmatrix} \tag{34}$$

The $\mathbf{R}$ matrix is one result of the QR decomposition. The fact that $\mathbf{R}$ is triangular leads to a number of simpler calculations. In particular, if we do need to calculate $\mathbf{R}^{-1}$, it is a comparatively fast, stable exercise.

The "thin" version of the $\mathbf{QR}$ decomposition is

$$\mathbf{X} = \mathbf{Q}\mathbf{R}. \tag{35}$$

The matrix $\mathbf{Q}$ is $n \times p$ orthogonal columns

$$\mathbf{Q} = \begin{bmatrix} Q & \begin{matrix} p\,columns \\ orthogonal \end{matrix} \\ n\,rows & \\ & \end{bmatrix}. \tag{36}$$

Orthogonality implies $\mathbf{Q}^{-1}\mathbf{Q} = \mathbf{Q}^T\mathbf{Q} = \mathbf{I}$, and $\mathbf{R}$ is the upper triangular matrix in (34).

The $\mathbf{R}$ produced by QR is, theoretically, equivalent to the Cholesky decomposition of $(\mathbf{X}^T\mathbf{X})$, with the possible exception that the diagonal elements in some of the rows in $\mathbf{R}$ from $\mathbf{QR}$ are not positive and signs of those rows need to be reversed. To avoid explicitly forming $(\mathbf{X}^T\mathbf{X})$, we replace $\mathbf{X}$ with $\mathbf{QR}$:

$$\mathbf{X}^T\mathbf{X} = (\mathbf{QR})^T\mathbf{QR} = \mathbf{R}^T\mathbf{Q}^T\mathbf{QR} = \mathbf{R}^T\mathbf{R}. \tag{37}$$

If a calculation calls for $(\mathbf{X}^T\mathbf{X})^{-1}$, then, we can replace that with $(\mathbf{R}^T\mathbf{R})^{-1}$. However, we would not explicitly calculate $(\mathbf{R}^T\mathbf{R})$ and invert that product. Instead, we note, theoretically

$$(\mathbf{R}^T\mathbf{R})^{-1} = \mathbf{R}^{-1}\mathbf{R}^{-1^T} \tag{38}$$

It is necessary to calculate $\mathbf{R}^{-1}$, but that is a simpler, more stable calculation because the lower left side of $\mathbf{R}$ is full of 0's. The inverse of an upper triangular matrix will also be upper triangular, so the benefits of this simplification continue.

Currently, I believe that most software implementations of the Cholesky root of $(\mathbf{X}^T\mathbf{X})$ will not form $(\mathbf{X}^T\mathbf{X})$ and then decompose it. They will instead conduct the decomposition of $\mathbf{X}$ itself, and then return the triangular $\mathbf{R}$ as the solution.

Method 4. Singular Value Decomposition

The QR decomposition is the predominant method of calculating regression estimates because it is fast and very stable numerically. An alternative decomposition, the singular value decomposition, is probably even better in terms of numerical stability (avoiding roundoff error, etc), but it is also more costly to compute. There has been discussion from time-to-time suggesting that the SVN will eventually become the predominant method, but, so far, it is not. However, in a close case, where a matrix is perhaps nearly singular, it may be that the SVD can calculate results that would be simply impossible with the other approaches.

The thin singular value decomposition (SVD) of $\mathbf{X}$ is a product of 3 matrices.

$$\mathbf{X} = \mathbf{UDV}^T. \tag{39}$$

$$\begin{bmatrix} \mathbf{U} & \begin{matrix} p\,columns \\ orthogonal \end{matrix} \\ n\,rows & \\ & \end{bmatrix} \begin{bmatrix} \delta_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \delta_p \end{bmatrix} \begin{bmatrix} \mathbf{V}^T & p\,columns \\ p\,rows & \end{bmatrix}. \tag{40}$$

The columns of $\mathbf{U}$ and $\mathbf{V}$ are orthogonal. That affords simplifications such as $\mathbf{U}^T\mathbf{U} = I$ and $\mathbf{V}^T = \mathbf{V}^{-1}$. The matrix $\mathbf{D}$ is a $p \times p$ diagonal matrix of the so-called "singular values", δ_i.

$$\mathbf{D} = diag(\delta_1, \delta_2, \ldots, \delta_p) = \begin{bmatrix} \delta_1 & 0 & & 0 \\ 0 & \delta_2 & & 0 \\ & & \ddots & \\ 0 & 0 & & \delta_p \end{bmatrix} \tag{41}$$

To see the simplifying benefit of the SVD, replace $\mathbf{X}$ with $\mathbf{UDV}^T$.

$$\mathbf{X}^T\mathbf{X} = (\mathbf{UDV}^T)^T\mathbf{UDV}^T = \mathbf{VDU}^T\mathbf{UDV}^T = (\mathbf{DV}^T)^T\mathbf{DV}^T = (\mathbf{VD})(\mathbf{VD})^T \tag{42}$$

The square root of $\mathbf{X}^T\mathbf{X}$ is thus seen to be $\mathbf{VD}$(or$(\mathbf{DV}^T)^T$, depending on how you want to group things. So the SVD based candidate for a square root of $\mathbf{X}^T\mathbf{X}$ is

$$\mathbf{S} = \mathbf{V}diag(\boldsymbol{\delta}) \tag{43}$$

The SVD approach is similar in personality to the eigenvalue decomposition of $\mathbf{X}^T\mathbf{X}$. If numerical linear algebra were "perfectly accurate", then the eigen method in equation (32), $\mathbf{V}diag(\boldsymbol{\lambda})^{1/2}$, would be identical to the SVD solution $\mathbf{V}diag(\boldsymbol{\delta})$. Consequently, we see that, on a theoretical level, the singular values are the squares of the eigen values.

4 Software Implementations that Draw from $MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

The multivariate normal generator distributed with R is `mvrnorm` in the recommended package MASS (Venables & Ripley, 2002). The Stata function `drawnorm` is part of the base package.

The important parameters specified by the user are 1) the number of draws required (n), 2) the population mean vector ($\boldsymbol{\mu}$) that has p elements, 3) the variance matrix ($\boldsymbol{\Sigma}$) which is $p \times p$, and 4) a tolerance parameter (tol) which is used to decide if the variance matrix is positive definite. The desired result is an $n \times p$ matrix in which each row is a draw from $MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. I concentrate on the case in which the user specifies a covariance matrix (rather than a correlation matrix).

The code for these functions is displayed in Appendices 1 and 2. Although the coding language differs between Stata and R, a careful review indicates that both of them are carrying out a 5 step algorithm.

1. Calculate the eigen decomposition of $\boldsymbol{\Sigma}$.
2. Check that $\boldsymbol{\Sigma}$ is positive definite by inspecting the eigenvalues.
   a) If an eigenvalue is intolerably negative, terminate with an error message.
   b) Tolerably negative eigenvalues are reset to 0.
3. Create a scaling matrix, $\mathbf{S}$. The two programs differ in this stage. R uses the eigen decomposition while Stata uses Cholesky roots.
4. Create a candidate $n \times p$ matrix of random vectors by drawing from $N(0, 1)$.
5. Apply $\mathbf{y} = \boldsymbol{\mu} + \mathbf{Sx}$ to rescale the candidate random draws.

To help the reader line up the critical parts for comparison, I offer Table 1, which enumerates the algorithmic steps with line numbers in the code.

For students who do not read much computer code, one detail is worth mentioning. In computer calculations, one of the slowest phases is allocation of memory, say for an $n \times p$ matrix. In numerical linear algebra, there is a tradition of writing new results on top of old matrices that already exist in memory. In R, for example,

```
X <- matrix(rnorm(p * n), n)
if(empirical) {
    X <- scale(X, TRUE, FALSE) # remove means
    X <- X %*% svd(X, nu = 0)$v # rotate to PCs
    X <- scale(X, FALSE, TRUE) # rescale PCs to unit variance
}
X <- drop(mu) + eS$vectors %*% diag(sqrt(pmax(ev, 0)), p) %*% t(X)
```

or in Stata,

```
qui mat accum `T' = `varlist', noc dev
mat `T' = `T'/(`nobs'-1)
mat `T' = cholesky(syminv(`T'))
```

In an analytical report, we would usually create new labels for the successive matrices, allowing us to differentiate them in our discussion. Efficient software coders don't allocate fresh memory unless they really need to.

4.1 Checking Σ for positive definiteness: Eigenvalues

The user supplies a variance matrix $\mathbf{\Sigma}$, but software cannot trust the user to supply a coherent matrix. Obviously, it would not make sense to ask for simulations from a silly matrices like

$$\begin{bmatrix} 0 & -1 \\ 2 & 0 \end{bmatrix} \; or \; \begin{bmatrix} 1 & .3 & 4 & 3 \\ .4 & 1 & 3 & 4 \\ 2 & 5 & 1 & 3 \\ 5 & 4 & 2 & 1 \end{bmatrix}, \tag{44}$$

but users sometimes ask silly questions.

All MVN programs should fail if the elements on the main diagonal, the variances, are not positive, and also if the other terms are mathematically incoherent (it should be symmetric and positive definite).

The programs `mvrnorm` and `drawnorm` check for trouble by evaluating the eigennvalues of the user-specified matrix $\mathbf{\Sigma}$. The key idea here is that $\mathbf{\Sigma}$ must be either positive definite ($\lambda_j > 0$) or positive semi-definite ($\geq \lambda_j$). The difference between "all positive eigenvalues" and "some positive eigenvalues and some equal to 0" is the difference between saying $\mathbf{\Sigma}$ is positive definite and $\mathbf{\Sigma}$ is positive semi-definite. If $\mathbf{\Sigma}$ has a column with no unique information (say, it is full of 0's or is a copy of another column), then we will find that there are several positive eigenvalues, but one or more might be exactly equal to 0. As we will explain below, this exercise only truly makes sense if $\mathbf{\Sigma}$is positive definite, but we can avoid outright failure if it is merely positive semi-definite.

It may happen that the "correct" (pencil-and-paper) eigenvalue is zero or a small positive number but rounding error in a digital computer leads to a negative value. In both `drawnorm` and `mvrnorm`, an allowance is made for that kind of numeric wobble. The positive definiteness test is failed only if there is an eigenvalue that is negative and outside a tolerance region, as specified by

$$\lambda_j < -tol\,|\lambda_1|. \tag{45}$$

Table 1: Aligning MVN Code

Step	mvrnorm (R)		drawnorm (Stata)	
	line	comments	line	comments
1. Eigenvalues	7-8	`eigen` is in base R.	79	`_checkpd` writes eigenvector and eigenvalues in memory
2a. Check positive semi-definite	9	λ_j less than $-tol\|\lambda_1\|$ causes termination. *tol* defaults to 10^{-6}, while in Stata it defaults to 10^{-8}	79	`_checkpd` returns `r(npos)`, an integer number representing the number of non-negative eigenvectors
2b. Reform eigenvalues	16	`pmax(ev, 0)`	91	`max(0, 'D'[1,'i'])`
3. Create re-scaling matrix	16	Uses eigenvalue method. The rescaling matrix $\mathbf{V}\,diag(\sqrt{\boldsymbol{\lambda}})$	85-93	if `r(npos)`=`p`, then use Cholesky root. Otherwise, use $\mathbf{V}\sqrt{diag(\boldsymbol{\lambda})}$
4. Create $\mathbf{X}$, $n \times p$ matrix of candidates from $N(0,1)$	10	`rnorm` uses a table-based CDF lookup procedure	121-27	Stata updated the normal random generator. Two versions are maintained.
5. Re-scale the candidate data into desired MVN	16	Result must be transposed before return to user	129-36	Stata score function is "inner product" of data row with weight matrix
			142	Standard deviation is vector of 1's (having no effect when user supplies covar matrix). M is the mean vector requested by user.
Step for generating data with mean and variance exactly equal to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.				
4.5. Rescale the candidate $\mathbf{X}$ so that the column means are $\mathbf{0}$ and variance matrix is $\mathbf{I}$	12	Mean center the columns		corr2data.ado
			144	Mean center the columns.
	13	Use SVD (principal components) to create uncorrelated columns.	147-54	Form $(\frac{1}{(n-1)}\mathbf{X}^T\mathbf{X})$, invert and take Cholesky root.

In `mvrnorm`, the default value of `tol` is 10^{-6} , while in Stata it is 10^{-8}. The key idea here is that the true value of the smaller eigenvalues might actually be positive, but the digital calculations have returned values just slightly below 0. If an eigenvalue is far enough from 0 to convince us that it truly is negative, then we conclude the user's matrix is not a valid, positive semi-definite matrix. The programs will stop and return an error message to the user.

4.2 Reforming the eigenvalues (and positive semi-definiteness).

Suppose none of the eigenvalues are negative enough to fail the test, but some are negative. None of the procedures we have for creating scaling matrices will work properly. We reform the negative eigenvalues by changing them to 0:

$$\lambda_j = max(0, \lambda_j).$$

4.3 The Scaling Matrix: Eigenvalue-based weights? Or the Cholesky-based weights?

The major difference between `drawnorm` and `mvrnorm` is in the selection of the MVN scaling matrix. In equation (22), we require a matrix $\mathbf{S}$ that can serve as a square root of $\mathbf{\Sigma}$.

In Stata, the procedure is conditional. If $\mathbf{\Sigma}$ is positive definite, then a Cholesky decomposition is used. If $\mathbf{\Sigma}$ is not positive definite, the alternative approach using the eigen decomposition is used.

In `mvrnorm`, the eigen decomposition is used, whether $\mathbf{\Sigma}$ is positive definite or semi-definite.[7] The eigen decomposition was used to check whether the eigenvalues are intolerably negative. It is put to use to calculate the scaling matrix.

The procedure in Stata wastes a bit of computation. The eigenvalue decomposition was calculated by the check for positive definiteness. The decision to create a Cholesky decomposition when $\mathbf{\Sigma}$ is full rank wastes some time. If one is running a simulation with hundreds of thousands of samples, this would have a noticeable effect.

When there are some 0's in the reformed eigenvalue vector, $\mathbf{\Sigma}$ is positive semi-definite. Both Stata and R use the eigen decomposition. Basically, this means that the user-specified $\mathbf{\Sigma}$ includes some redundant columns.

It deserves mention that the eigen decomposition is not really a solution for the inadequacies of $\mathbf{\Sigma}$, but rather it is a way to ignore them. The program will run without generating an error. But one should not assume this is a good outcome. In fact, the simulated MVN draws are don't really fill up the full p dimensional space.

It is easier to illustrate than explain. Suppose, the user input is

$$\boldsymbol{\mu} = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} \; and \; \mathbf{\Sigma} = \begin{bmatrix} 11 & 0 & 0 & 0 & 0 \\ 0 & 13 & 0 & 0 & 0 \\ 0 & 0 & 22 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{46}$$

$\mathbf{\Sigma}$ is positive semi-definite. The reformed eigenvalue vector is $(\lambda_1 = 22, \lambda_2 = 13, \lambda_3 = 11, 0, 0)$. The MVN generator will, basically, ignore the two columns of $\mathbf{\Sigma}$. The generator will return the mean in positions 4 and 5 of all simulated vectors. The first four draws (the rows in this output) from `rmvnorm` are [8]

[7] On the Wikipedia page for this topic discusses both, `https://en.wikipedia.org/wiki/Multivariate_normal_distribution`

[8]

	1	2	3	4	5
1	4.03	-2.76	7.66	5.00	6.00
2	1.46	9.51	10.83	5.00	6.00
3	4.69	1.26	0.98	5.00	6.00
4	9.29	5.24	-3.28	5.00	6.00

There's no variance in the columns for which the eigenvalue is 0. Hence, the ability to "work around" a positive semi-definite matrix is not a get out of jail free card. The simulation procedure here will not crash, but it not return useful output for the 4th and 5th elements of the simulated data rows. The simulation effectively generates an MVN draw with $3 = 5 - 2$ dimensions. The ability to tolerate positive semi-definite variance matrices is not hugely beneficial, except in avoiding crashes. Users are well advised to revise their variance matrices by eliminating linearly dependent columns.

4.4 Rescaling

To re-scale a single vector $\mathbf{x}$, we draw p values from $MVN(\mathbf{0}, \mathbf{I})$ and place them into $\mathbf{x} = (x_1, \ldots, x_p)^T$. We manufacture $\mathbf{y}$ with equation (22), which is reprinted (again) for reference.

$$\mathbf{y} = \boldsymbol{\mu} + \mathbf{S}\mathbf{x}.$$

There is a little wrinkle coming our way. The re-scaling equation handles column vectors, but the input data will be row vectors. In both `mvrnorm` and `drawnorm`, the input matrix $\mathbf{X}$ is an $n \times p$ matrix, where the candidates to be rescaled are rows. The programs fill up an $n \times p$ matrix candidate matrix,

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & & x_{1p} \\ x_{21} & x_{22} & & \\ & & \vdots & \\ x_{n1} & & & x_{np} \end{bmatrix}. \tag{47}$$

with draws from $N(0, 1)$ and a row can be thought of as a draw from $MVN(\mathbf{0}, \mathbf{I})$.

The calculation strategy in `mvrnorm` is to transpose $\mathbf{X}$. The candidate vectors, which are rows in $\mathbf{X}$, become columns of $\mathbf{X}^T$.

$$\mathbf{Y}^T = \boldsymbol{\mu}\mathbf{1}_n^T + \mathbf{S}\mathbf{X}^T. \tag{48}$$

$$\mathbf{Y}^T = \begin{bmatrix} y_{11} & y_{21} & & y_{n1} \\ y_{12} & y_{22} & & \\ & & & \\ y_{1p} & & & y_{np} \end{bmatrix} = \begin{bmatrix} \mu_1 & \mu_1 & & \mu_1 \\ \mu_2 & \mu_2 & (n & \mu_2 \\ \mu_3 & \mu_3 & cols) & \mu_3 \\ \vdots & \vdots & & \vdots \\ \mu_p & \mu_p & & \mu_p \end{bmatrix} + \begin{bmatrix} Scaling \\ matrix \\ (p \times p) \end{bmatrix} \begin{bmatrix} x_{11} & x_{21} & & x_{n1} \\ x_{12} & x_{22} & & \\ & & & \\ x_{1p} & & & x_{np} \end{bmatrix}.$$

As one can see, the matrix algebra requires us to have n copies of $\boldsymbol{\mu}$ side-by-side. ($\mathbf{1}_n^T$ is a row vector with n elements, all of which are equal to 1). A side effect of this approach is that the result is also a transposed matrix, $\mathbf{Y}^T$.

```
library(MASS)
set.seed(12345)
Sigma <- matrix(c(11, rep(0, 5), 13, rep(0, 5), 22, rep(0, 12)), ncol = 5)
mu <- c(2, 3, 4, 5, 6)
mvrnorm(4, mu, Sigma)
```

The Stata code works row by row, performing a series of inner-product calculations. To describe that, a mathematically equivalent representation would be

$$\mathbf{Y} = \mathbf{1}_n \boldsymbol{\mu}^T + \mathbf{X}\,\mathbf{S}^T. \tag{49}$$

One can think of the rescaling calculation as (48) or (49). I understood the method in (48) more readily, but (49) is perhaps more intuitive.

4.5 The $100,000 Question.

How can the two completely different scaling matrices lead to equally good MVN draws?

It is not intuitively reasonable to suppose that the the two versions,

$$\begin{bmatrix} r_{11} & 0 & 0 & 0 \\ r_{12} & r_{22} & 0 & 0 \\ & & \ddots & 0 \\ r_{1p} & r_{1p} & & r_{pp} \end{bmatrix}, \text{and} \quad \begin{bmatrix} \sqrt{\lambda_1}v_{11} & \sqrt{\lambda_2}v_{12} & & \sqrt{\lambda_p}v_{1p} \\ \sqrt{\lambda_1}v_{21} & \sqrt{\lambda_2}v_{22} & & \sqrt{\lambda_p}v_{2p} \\ & & \ddots & \\ \sqrt{\lambda_1}v_{p1} & \sqrt{\lambda_2}v_{p2} & & \sqrt{\lambda_p}v_{pp} \end{bmatrix}, \tag{50}$$

are equally good candidates to serve as $\mathbf{S}$. The one on the left is half filled with 0's. It is apparent that we will get different simulated draws from these two matrices.

My intuition resists the idea that these two scaling matrices are equally valid. I have come to accept the fact that they are adequate for two reasons. First, it is apparent that $\mathbf{S}^T\mathbf{S} = \mathbf{S}\mathbf{S}^T = \boldsymbol{\Sigma}$, using either matrix in (50) as $\mathbf{S}$. They are both square roots of the same $\boldsymbol{\Sigma}$, meaning the data re-scaled with them is drawn from $MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Second, I became aware of this fact that, generally speaking, matrix square roots are not unique.

Proposition 3. *Given a symmetric $\boldsymbol{\Sigma}$ for which a square root S exists ($\mathbf{S}^T\mathbf{S} = \boldsymbol{\Sigma}$), and given $\mathbf{V}$ is orthonormal, then $\mathbf{VS}$ is also a square root.*

Proof. Recall $\mathbf{V}\mathbf{V}^T = \mathbf{I}$.

$$(\mathbf{VS})^T(\mathbf{VS}) \;=\; \mathbf{S}^T\mathbf{V}^T\mathbf{VS} = \mathbf{S}^T\mathbf{S}. \tag{51}$$

There are various suggestions about other ways to find a square-root decomposition that is most numerically stable, or fast to compute, or unique within some restricted class of matrices. For example, it can be shown that, if we insist the main diagonal of the Cholesky triangle is positive, then there is a unique triangular $\mathbf{R}$ satisfying $\mathbf{R}^T\mathbf{R}$ if $\boldsymbol{\Sigma}$ is symmetric. Golub and Van Loan (1996, p. 149) show (by a singular value decomposition of the Cholesky triangle) that there is a unique square root if we insist the square root matrix itself is symmetric and positive semi-definite. Unfortunately, the two "unique" square roots are not equal to each other.

Some authors are drawn to the Cholesky decomposition, when it exists, because we can, at least, get a basic understanding of what's going on. Write out the scaling equation (48) for a 4 dimensional problem (keep this simple by letting $\boldsymbol{\mu} = 0$).

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} r_{11} & 0 & 0 & 0 \\ r_{12} & r_{22} & 0 & 0 \\ r_{13} & r_{23} & r_{33} & 0 \\ r_{14} & r_{24} & r_{34} & r_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$= \begin{bmatrix} r_{11}x_1 \\ r_{12}x_1 + r_{22}x_2 \\ r_{13}x_1 + r_{23}x_2 + r_{33}x_3 \\ r_{14}x_1 + r_{24}x_2 + r_{34}x_3 + r_{44}x_4 \end{bmatrix}$$

The first term $y_1 = r_{11}x_1$ is easy to understand. The candidate x_1 has been weighted by r_{11}, which is a standard deviation. That row is precisely analogous to equation (19). The following rows are less easy to grasp, however.

4.6 There's More: Its reversible.

If we are given a draw from a correlated MVN, it is possible to "de-correlate" the columns. This is sometimes referred to as "whitening", as in converting columns to "white noise."

Recall that the focal calculation is (22), $\mathbf{y} = \boldsymbol{\mu}+\mathbf{Sx}$. In equation (26), we have the tool understand the transform of a draw like $\mathbf{y}$ into some other distribution. This works whether we want the transformation to go back to $MVN(0, I)$ or something else. If $\mathbf{y} \sim MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, a new variable $\mathbf{z} = \mathbf{b} + \mathbf{Ay}$ is distributed as $N(\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T)$. So $\mathbf{y}$ can be converted to $\mathbf{z} \sim MVN(\mathbf{0},\mathbf{I})$ by setting $\mathbf{A} = \mathbf{S}^{-1}$ and $\mathbf{b} = -\boldsymbol{\mu}\mathbf{S}^{-1}$. It is especially easy to see that if we write

$$\begin{aligned} \mathbf{z} &= \mathbf{b} + \mathbf{Ay} = \mathbf{b} + \mathbf{A}\ (\boldsymbol{\mu} + \mathbf{Sx}) && (52) \\ &= \mathbf{b} + \mathbf{A}\boldsymbol{\mu} + \mathbf{ASx} && (53) \end{aligned}$$

In order to end up with $\mathbf{z} \sim MVN(\mathbf{0},\mathbf{I})$, it is necessary that $\mathbf{AS} = \mathbf{I}$ and $\mathbf{b} + \mathbf{A}\boldsymbol{\mu} = \mathbf{0}$.

5 Forcing the Sample Average to Have Observed Statistics μ and Σ.

Both $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are characteristics of a data generator (a.k.a "population parameters"). We often want to compare the sample estimates, which I'll refer to as $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$.

The summary statistics from a sample are not equal to the parameters of the data generator. There is variation among samples. Newcomers often expect that the mean of a sample drawn from $N(10, 20)$ will be exactly 10. It takes a little time for them to become reconciled to the reality of this.

It never occurred to me that anybody might like to manipulate a data generator so that the observed mean would exactly equal $\boldsymbol{\mu}$ (or that they would want $\hat{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}$). It never occurred to me until I heard about the Stata function `corr2data`. It does exactly that.

The Stata simulation code used by Guo and Fraser uses `corr2data` to draw a sample that is supposedly multivariate normal. They describe the variables thus:

> "where x_1, x_2, x_3, Z, and u are random variables, normally distributed with a mean vector of (3 2 10 5 0), a standard deviation vector (.5 .6 9.5 2 1) and the following symmetric correlation matrix:
>
> $$r(x_1, x_2, x_3, Z, u) = \begin{bmatrix} 1 & & & & \\ .2 & 1 & & & \\ .3 & 0 & 1 & & \\ 0 & 0 & 0 & 1 & \\ 0 & 0 & 0 & .4 & 1 \end{bmatrix}.$$
>
> In addition, v is a random variable that is normally distributed with mean zero and variance 1..." (2015, p. 350).

However, according to its documentation (StataCorp, 2015a), `corr2data` does not purport to generate a multivariate normally distributed sample:

> `corr2data` ... creates a new dataset with a specified covariance (correlation) structure.... The purpose of this is to allow you to perform analyses from summary statistics (correlations/covariances and maybe the means) when these summary statistics are all you know and summary statistics are sufficient to obtain results.
>
> The data created by `corr2data` are artificial; they are not the original data, and it (*sic*) is not a sample from an underlying population with the summary statistics specified. See `drawnorm` if you want to generate a random sample....
>
> The dataset `corr2data` creates is suitable for one purpose only: performing analyses when all that is known are summary statistics and those summary statistics are sufficient for the analysis at hand.

After digesting this information, I realized that the `mvrnorm` function for R has an equivalent, setting the argument `empirical = TRUE`.

At the moment, I have two questions.

1. How do they do that? For that part, I have a good answer.
2. What do they get when they do that? This part has no good answer yet, but there are interesting questions. Is there any formal way to understand the distortion caused by usage of `corr2data` when a draw from an MVN is needed instead?

5.1 How Do They Do That?

This is a "data standardization" chore. It can be accomplished by inserting another step in the middle of the 5 step algorithm for MVN draws (see the last entry in Table 1).

5.1.1 The big picture: standardizing variables

One standardized variable

Standardizing data is familiar to most social scientists. Consider x, a column variable, with an estimated mean $\hat{\mu} = \frac{1}{n}\sum x_i$ and standard deviation $\hat{\sigma} = \sqrt{\frac{1}{(n-1)}\sum(x_i - \hat{\mu})^2}$. The value we often refer to as a "standardized variable" is calculated as

$$\hat{Z}_i = \frac{x_i - \hat{\mu}}{\hat{\sigma}}. \tag{54}$$

That variable has a mean of 0 and standard deviation 1. This is true, no matter what data generator supplies x_i, even if it is not normal.

$\hat{Z}$ is an estimate of how x_i is fluctuating above and below the center of the random process. It is an estimate of variable that is centered and standardized on the true population parameters,

$$Z_i = \frac{x_i - \mu}{\sigma}. \tag{55}$$

This variable Z_i is normally distributed if x_i is normal. Note $Z_i = -\frac{\mu}{\sigma} + \frac{1}{\sigma}x_i$, a linear translation of x_i. By the same derivation as (19), we are certain this is normal, $N(0, 1)$.

The empirically standardized $\hat{Z}_i$ does not have an expected value of 0, or parametric variance 1, but the vector $\hat{\mathbf{Z}}$ has an observed mean exactly 0 and variance equal to 1. Its not normal any more, but that doesn't stop us from rescaling it. A research who needs to manufacture a new column with empirical mean a and standard deviation b can apply this rescaling equation:

$$a + b\hat{Z}_i = a + b\left(\frac{x_i - \hat{\mu}}{\hat{\sigma}}\right). \tag{56}$$

I don't know what the name for that thing is.

It is not $N(a, b^2)$. But the empirical mean is a and the variance is b^2.

Multivariate standardization: step 4.5

By analogy to the procedure in (56), a multivariate process is employed in `corr2data`.

The candidate data matrix in which each row is $MVN(\mathbf{0}, \mathbf{I})$, is transformed so that the observed mean of each column is 0 and the observed variance matrix is the identity matrix. This is not the same as standardizing each column separately. Not only must each column's empirical mean be 0 with standard deviation is 1, but also the observed correlation between any pair of columns must be 0.

The mean-centered matrix, $\mathbf{X}_c$, is made up of columns $\mathbf{X}[\,, j] - mean(\mathbf{X}[\,, j])$. After this, the empirical mean of each column in $\mathbf{X}_c$ is 0.

The unbiased empirical estimate of the variance matrix is

$$Var(\mathbf{X}_c) = \frac{1}{(n-1)}\mathbf{X}_c^T\mathbf{X}_c \tag{57}$$

We need to find a de-correlating matrix $\mathbf{A}$ with the property that

$$\mathbf{X}_u = \mathbf{X}_c\mathbf{A} \; such\, that \, Var(\mathbf{X}_c\mathbf{A}) = \mathbf{I}. \tag{58}$$

We have competing ways to calculate $\mathbf{A}$, one which is more immediately understandable, one of which is more numerically accurate. The more immediately understandable approach is the one implemented in Stata's `corr2data`. Because $Var(\mathbf{X}_c\mathbf{A}) = \mathbf{A}^T Var(\mathbf{X}_c)\mathbf{A}$, we can write the requirement

$$Var(\mathbf{X}_c\mathbf{A}) = \mathbf{A}^T Var(\mathbf{X}_c)\mathbf{A} \quad = \quad \mathbf{I}.$$

Then the inverse of both sides is

$$\mathbf{A}^{-1}Var(\mathbf{X}_c)^{-1}\mathbf{A}^{T^{-1}} = \mathbf{I} \tag{59}$$

and we find

$$Var(\mathbf{X}_c)^{-1} = \mathbf{A}\mathbf{A}^T. \tag{60}$$

Thus, the desired matrix $\mathbf{A}$ is clearly a square root of $Var(\mathbf{X}_c)^{-1}$.

That approach has some bad numerical properties (high roundoff error). The condition (index of numerical instability, see) of $\mathbf{X}_c^T\mathbf{X}_c$ is the square of the condition of $\mathbf{X}_c^T$. It is recommended instead employ a solution that does not require the explicit formulation of $\mathbf{X}_c^T\mathbf{X}_c$ or the calculation of its inverse. Those concerns are clearly in the forefront of the approach in the R, where the singular value decomposition is used.

After step 4.5 is complete, the de-correlated matrix X_u has been created. It replaces $\mathbf{X}$ in step 5. Because $\mathbf{X}_u$ has column means equal to 0 and empirical variance $\mathbf{I}$, the result has summary statistics that exactly match the user request.

5.1.2 Implementation Details: mvrnorm with `empirical = TRUE`

The `mvrnorm` function's argument `empirical` is documented as follows: "mu and Sigma specify the empirical not population mean and covariance matrix." In the `mvrnorm` code (see Appendix 1), only 5 lines between steps 4 and 5 are altered (see lines 11 through 15).

3. Line 12. Create a mean-centered matrix $\mathbf{X}_c$.

4. Line 13. Use a singular value decomposition (principal components analysis) generate a new candidate matrix in which the empirically observed correlations among the columns are 0. New "empirically de-correlated" columns are produced using estimates of the eigenvalues of $\mathbf{X}_c^T\mathbf{X_c}$ that are produced with principal component analysis.

$$\mathbf{X}_c\mathbf{V} \tag{61}$$

   The columns are referred to as principal component scores; they are orthogonal columns, empirically uncorrelated (Pearson's r between columns is 0; it is easy to show that $Var(\mathbf{X}_c\mathbf{V}) = \mathbf{I}$). I suspect the reader will take my word for that, or else some background reading on principal components might be in order.

   The principal component scores are organized so that column 1 has the greatest variance and the last column has the smallest variance.

5. Line 14. Suppose the standard deviation estimates are $\hat{\boldsymbol{\sigma}} = (\hat{\sigma}_1, \ldots, \hat{\sigma}_p)^T$. Create X_u, a "standardized" set of columns of $\mathbf{X_c}\mathbf{V}$. In effect, we divide each column of $\mathbf{X_c}\mathbf{V}$ by its observed standard deviation.

$$\mathbf{X}_u = (\mathbf{X}_c\mathbf{V}) \begin{pmatrix} \frac{1}{\hat{\sigma}_1} & 0 & \ldots & 0 \\ 0 & \frac{1}{\hat{\sigma}_2} & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & \frac{1}{\hat{\sigma}_p} \end{pmatrix} = (\mathbf{X}_c\mathbf{V})diag(\frac{1}{\hat{\boldsymbol{\sigma}}}) \tag{62}$$

   It turns out that the variance of the columns of X_cV can be calculated from the singular values, which are also available from the SVD. The variance of the j'th column of the scores in $\mathbf{X}_c\mathbf{V}$ is $\delta^2/(n-1)$, so the standard deviations are $\hat{\sigma}_j = \delta_j/\sqrt{n-1}$. Hence,

$$diag(\frac{1}{\hat{\sigma}}) = diag(\sqrt{n-1}/\boldsymbol{\delta}) \tag{63}$$

   Steps 2 and 3 can be carried out in one single step, combining the de-correlating and re-scaling effort.

$$\mathbf{X}_u = \mathbf{X}_c\left(\mathbf{V}\, diag(\frac{\sqrt{n-1}}{\delta})\right) \tag{64}$$

   If we write it in that way, we see that the de-correlating matrix $\mathbf{A}$ in (58) is $\mathbf{V}diag(\sqrt{n-1}/\delta)$.

Because these changes purge the candidate matrix $\mathbf{X}$ of its randomized "individuality", step 5 in the algorithm will produce a data structure in which the observed mean and variance matrix exactly match the user's request.

SVD Implementation

The authors of `mvrnorm` choose to use singular value decomposition (SVD) of $\mathbf{X}_c$ because of its superior numerical stability. Recall the SVD is a product of 3 matrices.

$$\mathbf{X}_c = \mathbf{U}\mathbf{D}\mathbf{V}^T. \tag{65}$$

$\mathbf{U}$ is an orthogonal matrix that is $n \times p$, $\mathbf{V}$ is also orthogonal $p \times p$, and $\mathbf{D} = diag(\boldsymbol{\delta})$.

A de-correlating matrix consistent with (58) is obtained by replacing $\mathbf{X}_c$ in the variance formula with $\mathbf{U}\mathbf{D}\mathbf{V}^T$.

$$\begin{aligned} Var(\mathbf{X}_c) &= \frac{1}{(n-1)}\mathbf{X}_c^T\mathbf{X}_c \\ &= (\frac{1}{\sqrt{n-1}}\mathbf{D}\mathbf{V}^T)^T\left(\frac{1}{\sqrt{n-1}}\mathbf{D}\mathbf{V}^T\right). \end{aligned} \tag{66}$$

Because the inverse of a product is the product of the inverses, in reverse order,

$$\begin{aligned} Var(\mathbf{X}_c)^{-1} &= (\mathbf{V}\sqrt{n-1}\mathbf{D}^{-\mathbf{1}})(\mathbf{V}\sqrt{n-1}\mathbf{D}^{-\mathbf{1}})^T \\ &= \mathbf{V}diag(\sqrt{n-1}/\boldsymbol{\delta})\,(\mathbf{V}diag(\sqrt{n-1}/\boldsymbol{\delta}))^T \end{aligned} \tag{67}$$

The term $\mathbf{V}diag(\sqrt{n-1}/\boldsymbol{\delta})$ can be a square root of $Var(\mathbf{X}_c)^{-1}$.

5.1.3 Implementation Details: Stata's corr2data

The Stata `corr2data` function is in a file named `corr2data.ado` that is presented in Appendix 3.

Step 4.5 takes on a different appearance partly because this code is written in Stata, but mostly because instead of using an SVD based square root of the variance matrix, they use the Cholesky root of the inverse of the variance matrix.

6. Line 144. The mean-centered candidate matrix, $\mathbf{X}_c$.

7. Recall the inverse of the variance matrix approach in (60) The Stata code explicitly calculates $\mathbf{X}^T\mathbf{X}$, inverts the variance matrix and extracts the square roots of the inverse calculating $\mathbf{A}^T\mathbf{A} = Cholesky(Var(\mathbf{X}_c)^{-1})$. That approach uses several of the calculations that are discouraged in Golub & Van Loan (1996). Other strategies might be more accurate.

8. The calculation that manufactures the de-correlated matrix $\mathbf{X}_u$ is found to be:

$$\mathbf{X}_c\mathbf{A} \tag{68}$$

The end product is a matrix in which, except for roundoff error, the mean of each column is 0 and the columns are uncorrelated and have variance equal to 1.

5.2 What do they get when they do that?

I now appreciate the ambiguous commentary in the Stata documentation for `corr2data`, it "is not a sample from an underlying population with the summary statistics specified."

The $n \times p$ manufactured data set does not have rows drawn from $MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, but what does it have? Intuition suggests that this new data set is probably pretty close to multivariate normal, perhaps it is a multivariate t distribution.

I do not (yet) know the distribution of these corr2data draws, but I see reasons to expect this is more like a multivariate t than a normal distribution. We can see that the rows of the corr2data draws are not independently and identically distributed.

Lets start by considering just one "standardized variable". Let the input values be x_i, let the empirically standardized scores be $\hat{Z}_i = \frac{1}{\hat{\sigma}}(x_i - \hat{\mu})$. The vector $\hat{\mathbf{Z}} = (\hat{Z}_1, \hat{Z}_2, \ldots, \hat{Z}_n)^T$ collects together n standardized values.

In introductory statistics, they told us to act as if $\hat{Z}_i$ is a sample drawn from $N(0,1)$. That's obviously wrong. The values $\hat{\mathbf{Z}}$ are not statistically independent. Each element $\hat{Z}_i$ depends on fluctuations in any of the x_i's because the estimated mean and standard deviation include all of those scores.

What can we say for sure?

1. If $n > 1$, the empirical average is 0 and the standard deviation is 1.
2. $E[\hat{Z}_i] = 0$ because $E[x_i] = \mu$.
3. From sample to sample, the mean $\hat{\mu}$ cannot change, it is fixed at 0, so its variance has to be 0. Hence, the standard error of the mean of $\hat{Z}$ is 0.

Is the Stata method in corr2data more "wrong" than the method used in R's mvrnorm?

The answers are "No" and "Yes".

The answer is "No" in the sense that of these functions empirically standardizes the candidate $\mathbf{X}$ matrix and then rescale the result. By showing that each one derives the matrix square root of the inverse of a variance matrix, they more-or-less aiming at the same thing. I've compared quite a few calculations using the two matrix methods and the calculated values are the same up to the 8th decimal place.

The answer is "Yes" because `corr2data` handles the random number stream very badly. Like R, Stata uses a system-wide pseudo random generator (PRNG) stream that powers many different calculations. If one runs `drawnorm` several times, it generates different MVN draws each time because the position in the system-wide PRNG is advanced each time. In contrast, `corr2data` gives the exact same data set every time we call it, unless we explicitly set the argument `seed()`. `corr2data` does not advance the system-wide PRNG and there is simply no way to feel confident that one block of data returned after setting seed(234234) is not correlated with a block returned from corr2data with seed(432432).

Aside from the numerical precision, and the peculiarity of calling corr2data, it is difficult to say that we have fully understood the properties of corr2data. We end up with the same frustrating "too many square roots" problem that frustrated the conclusion of the MVN generation discussion in section 4.5. We have two seemingly different scaling matrices are equally good on theoretical terms.

6 Conclusions

This discussion tries to combine lessons in matrix algebra with a detailed comparison of software that generates multivariate normal samples. I have learned a good deal about the details of creating simulated draws and have been reminded of many details in linear algebra that I had forgotten, or never knew.

It has been shown that draws from a multivariate normal distribution can be created in a 5 step algorithm. This algorithm, which is similarly implemented in R and Stata, works in an

understandable way. The algorithm inspects the user's request for internal coherence (the positive definite variance matrix), creates a scaling matrix by extracting the square root of the variance matrix, and then reshapes candidate draws that can be pulled from a standard normal distribution $N(0,1)$. Since some software frameworks do not offer pre-packaged MVN data generators, an understanding of this procedure might be helpful to some readers.

Appendix 1. mvrnorm

In the MASS package (Venables & Ripley, 2002) for R (R Core Team, 2015), one finds the file "mvrnorm.R", in which the `mvrnorm` function is found.

```
mvrnorm <-
    function(n = 1, mu, Sigma, tol = 1e-6, empirical = FALSE, EISPACK = FALSE)
{
    p <- length(mu)
    if(!all(dim(Sigma) == c(p,p))) stop("incompatible arguments")
    if (missing(EISPACK)) EISPACK <- getOption("mvnorm_use_EISPACK", FALSE)
    eS <- eigen(Sigma, symmetric = TRUE, EISPACK = EISPACK)
    ev <- eS$values
    if(!all(ev >= -tol*abs(ev[1L]))) stop("'Sigma' is not positive definite")
    X <- matrix(rnorm(p * n), n)
    if(empirical) {
        X <- scale(X, TRUE, FALSE) # remove means
        X <- X %*% svd(X, nu = 0)$v # rotate to PCs
        X <- scale(X, FALSE, TRUE) # rescale PCs to unit variance
    }
    X <- drop(mu) + eS$vectors %*% diag(sqrt(pmax(ev, 0)), p) %*% t(X)
    nm <- names(mu)
    if(is.null(nm) && !is.null(dn <- dimnames(Sigma))) nm <- dn[[1L]]
    dimnames(X) <- list(nm, NULL)
    if(n == 1) drop(X) else t(X)
}
```

Appendix 2. drawnorm.ado

```
*! version 7.3.0  03feb2015
program define drawnorm
  local xeqversion : di "version " string(_caller()) ":"
  version 8.2
  local version _caller()

  gettoken first 0: 0, parse(",")
  #del;
  syntax [,
    n(string)
    SEED(string)
    Double
    CORR(string)
    COV(string)
    CStorage(string)
    Means(string)
    SDs(string)
    CLEAR
    FORCEPSD
    TOL(passthru) // undocumented
```

```
21  ] ;
22  #del cr
23  quietly count
24  local curn = r(N)
25  if `"`n'"' != "" {
26    confirm integer n `n'
27    if `n' == `curn' {
28      local n
29    }
30  }
31  if `"`n'"' == "" {  /* add newvarlist to existing dataset */
32    local nobs = r(N)
33    if `nobs' <= 0 {
34      error 2000
35    }
36    if "`clear'" != "" {
37      drop _all
38      qui set obs `nobs'
39      local n `nobs'
40    }
41  }
42  else {      /* generate new dataset */
43    if `n' <= 0 {
44      error 2000
45    }
46    qui count
47    if `n' != r(N) {
48      qui des, short
49      if r(changed) & ("`clear'" == "" ) {
50        error 4
51      }
52      drop _all
53    }
54    local nobs = `n'
55    qui set obs `nobs'
56  }
57  local 0 "`first'"
58  syntax newvarlist
59  local k : word count `varlist'
60  if "`seed'" != "" {
61    `xeqversion' set seed `seed'
62  }
63  tempname C D L M P S
64  if "`cov'" != "" | "`corr'" != "" {
65    if "`cov'" != "" & "`corr'" != "" {
66      dis as err "cov() and corr() " ///
67          "may not be specified together"
68      exit 198
69    }
70
71    if `"`corr'"' != "" {
72      _m2matrix `C' corr `k' "`corr'" "`cstorage'"
73      local Cname `corr'
74    }
75    else {
76      _m2matrix `C' cov  `k' "`cov'"  "`cstorage'"
77      local Cname `cov'
78    }
79    _checkpd `C', matname(`Cname') check(psd) `forcepsd' `tol'
```

```
    if r(npos)==`k' {
      // C is positive definite;
      // for backward compatibility: use Cholesky root
      matrix `P' = cholesky(`C')
    }
    else {
      // in the singular case, we use eigen decomposition
      // already available from _checkpd
      matrix `L' = r(L)
      matrix `D' = r(Ev)
      forvalues i = 1/`k' {
        matrix `D'[1,`i'] = sqrt(max(0,`D'[1,`i']))
      }
      matrix `P' = `L'*diag(`D')
    }
  }
  else {
    matrix `P' = I(`k')
  }
  /* M = means */
  if "`means'" != "" {
    _m2matrix `M' means `k' "`means'"
  }
  else {
    matrix `M' = J(1,`k', 0)
  }
  /* S = stds */
  if `"`sds'"' != "" {
    if `"`cov'"' != "" {
      dis as err "cov() and sds() may not be specified together"
      exit 198
    }
    _m2matrix `S' sds `k' "`sds'"
  }
  else {
    matrix `S' = J(1,`k',1)
  }
  /* generate new variables */
  tokenize `varlist'
  local newlist `varlist'
  foreach var of local newlist {
    if `version' <= 10 {
      qui gen `double' `var' = invnormal(uniform())
    }
    else {
      qui gen `double' `var' = rnormal()
    }
  }
  /* transform to desired corr */
  mat roweq `P' = " "
  mat coleq `P' = " " /* remove possible equation names from P */
  mat rownames `P' = `varlist'
  mat colnames `P' = `varlist'
  forvalues i = 1 / `k' {
    tempname new`i' row
    mat `row' = `P'[`i', 1...]
    mat score `new`i'' = `row'
  }

```

```
  /* transform to desired means and std */
  tokenize `varlist'
  forvalues i = 1 / `k' {
    qui replace ``i'' = `new`i'' * `S'[1,`i'] + `M'[1,`i']
  }

  if "`n'" != "" {
    local nobs = string(`nobs',"%12.0fc")
    dis as txt "(obs `nobs')"
  }
end
```

Appendix 3. corr2data.ado

```
*! version 7.3.0  03feb2015
program corr2data
  version 8.2

  query sortseed
  local sortseed = r(sortseed)
  local currseed = c(seed)
  capture noisily  Make `0'
  set seed `currseed'
  set sortseed `sortseed'
  if _rc {
    exit _rc
  }
end

program Make
  gettoken first 0: 0, parse(",")
  #del ;
  syntax [,
    n(string)
    CORR(string)
    COV(string)
    CStorage(string)
    Means(string)
    SDs(string)
    SEED(int 0)
    Double
    CLEAR
    FORCEPSD
    TOL(passthru) // undocumented
  ] ;
  #del cr

  if `"`n'"' != "" {
    confirm integer n `n'
    if `n' == _N {
      local n
    }
  }
  if "`n'" == "" {  /* add newvarlist to existing dataset */
    local nobs = _N
    if `nobs' <= 0 {
      error 2000
    }
```

```
45      if "`clear'" != "" {
46        drop _all
47        qui set obs `nobs'
48        local n `nobs'
49      }
50    }
51    else {      /* generate new dataset */
52      if `n' <= 0 {
53        error 2000
54      }
55      qui count
56      if `n' != r(N) {
57        qui des, short
58        if r(changed) & ("`clear'" == "" ) {
59          error 4
60        }
61        drop _all
62      }
63      local nobs = `n'
64      qui set obs `nobs'
65    }
66    local 0 "`first'"
67    syntax newvarlist
68    local k : word count `varlist'
69    if `nobs' <= `k' {
70      dis as err "number of observations should exceed number of variables"
71      exit 2001
72    }
73    tempname C D L M P S T
74
75    if "`cov'" != "" | "`corr'" != "" {
76      if "`cov'" != "" & "`corr'" != "" {
77        dis as err "cov() and corr() " ///
78            "may not be specified together"
79        exit 198
80      }
81      if `"`corr'"' != "" {
82        _m2matrix `C' corr `k' "`corr'" "`cstorage'"
83        local Cname `corr'
84      }
85      else {
86        _m2matrix `C' cov  `k' "`cov'"  "`cstorage'"
87        local Cname `cov'
88      }
89      tempname Cmatrix
90      matrix `Cmatrix' = `C'
91      local rows = rowsof(`"`Cmatrix'"')
92      local cols = colsof(`"`Cmatrix'"')
93      if `rows' != `cols' | `rows' != `k' | `cols' != `k' {
94        di as err "{p}" ///
95        "matrix is not conformable with the number of " ///
96        "variables requested:  rows and columns must " ///
97        "equal the number of specified variables{p_end}"
98        exit 503
99      }
100     _checkpd `C', matname(`Cname') check(psd) `forcepsd' `tol'
101
102     if r(npos)==`k' {
103       // C is positive definite;
```

```
104       // for backward compatibility: use Cholesky root
105       matrix `P' = cholesky(`C')
106     }
107     else {
108       // in the singular case, we use eigen decomposition
109       // already available from _checkpd
110       matrix `L' = r(L)
111       matrix `D' = r(Ev)
112       forvalues i = 1/`k' {
113         matrix `D'[1,`i'] = sqrt(max(0,`D'[1,`i']))
114       }
115       matrix `P' = `L'*diag(`D')
116     }
117   }
118   else {
119     matrix `P' = I(`k')
120   }
121   /* M = means */
122   if `"`means'"' != "" {
123     _m2matrix `M' means `k' "`means'"
124   }
125   else {
126     matrix `M' = J(1,`k', 0)
127   }
128   /* S = stds */
129   if "`sds'" != "" {
130     if "`cov'" != "" {
131       dis as err "cov() and sds() may not be specified together"
132       exit 198
133     }
134     _m2matrix `S' sds `k' "`sds'"
135   }
136   else {
137     matrix `S' = J(1,`k',1)
138   }
139   /* generate new variables */
140   set seed0 `seed'
141   foreach var of local varlist {
142     qui gen `double' `var' = invnorm(uniform0())
143     qui sum `var'
144     qui replace `var' = `var' - r(mean)
145   }
146   /* reform them to be zero corr */
147   qui mat accum `T' = `varlist', noc dev
148   mat `T' = `T'/(`nobs'-1)
149   mat `T' = cholesky(syminv(`T'))
150   forvalues i = 1 / `k' {
151     tempname new`i' row
152     mat `row' = (`T'[1..., `i'])'
153     mat score `new`i'' = `row'
154   }
155   tokenize `varlist'
156   forvalues i = 1 / `k' {
157     qui replace ``i'' = `new`i''
158   }
159   /* transform to desired corr */
160   mat roweq `P' = " "
161   mat coleq `P' = " " /* remove possible equation names from P */
162   mat rownames `P' = `varlist'
```

```
  mat colnames `P' = `varlist'
  forvalues i = 1 / `k' {
    tempname new`i' row
    mat `row' = `P'[`i', 1...]
    mat score `new`i'' = `row'
  }

  /* transform to desired means and std */
  tokenize `varlist'
  forvalues i = 1 / `k' {
    qui replace ``i'' = `new`i'' * `S'[1,`i'] + `M'[1,`i']
  }

  if "`n'" != "" {
    local nobs = string(`nobs',"%12.0fc")
    dis as txt  "(obs `nobs')"
  }
end
```

References

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer, 1986 edition edition.

Golub, G. H. & Van Loan, C. F. (1996). *Matrix computations*. Johns Hopkins studies in the mathematical sciences. Baltimore: Johns Hopkins University Press, 3rd ed edition.

Greene, W. H. (2008). *Econometric analysis*. Upper Saddle River, N.J: Prentice Hall, 6th ed edition.

Guo, S. & Fraser, M. W. (2015). *Propensity Score Analysis: Statistical Methods and Applications*. Los Angeles: SAGE Publications, Inc, 2ed edition.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Scheuer, E. M. & Stoller, D. S. (1962). On the Generation of Normal Random Vectors. *Technometrics*, 4(2), 278.

StataCorp (2015a). *Stata 14 Base Reference Manual*. Stata Press., College Station, TX.

StataCorp (2015b). *Stata Statistical Software: Release 14*. StataCorp LP, College Station, TX.

Venables, W. N. & Ripley, B. D. (2002). *Modern Applied Statistics with S*. New York: Springer, fourth edition. ISBN 0-387-95457-0.