Lab 2: A Review of Matrix Algebra

EDP 380C.28: Simulation in R

Fall 2022 – Unique: 12104

Brian T. Keller, Ph.D.
bk@utexas.edu
University of Texas at Austin
August 29, 2022

Objectives

- Gain familiarity with Linear Algebra in R.
- Learn how implement basic statistical computations via matrices.
- Practice implementing equations into code.
- Learn to generate data from a multivariate normal distribution.
- Learn about linear combinations and the properties of their composites.

This lab will help review and familiarize you with matrix computations and properties important in multivariate statistics. These relationships and computations are essential when simulating data because data simulation is inherently a multivariate technique where we generate data that follows some hypothesized multivariate distribution. While many of the data-generating processes we use may not require matrices, certain situations can see computational efficiencies when the data sets have high dimensionality (e.g., a high number of variables).

In R:

(1) Run the following included code to set up a `list()` to store answers.

```
answers <- vector('list', 8L)
```

Note: It is essential to follow the steps in the listed order to have answers graded as correct.

1 A Brief Introduction to Linear Algebra

This section aims to provide a brief non-technical approach to linear algebra for those unfamiliar or who need a refresher.

1.1 Definitions

Scalar: A single unit (e.g., a number).

Elements: The scalars that compose an array.

Array: An ordered arrangements of scalars.

Vector: A one-dimensional array. We will use bolded lower case letters to represent a vector:

$$\mathbf{y}$$

Matrix: A two-dimensional array, where the rows represent one dimension and the columns represent another. We will use bolded upper case letters to represent a matrix:

$$\underset{n\times p}{\mathbf{X}}$$

Note, sometimes we will include the row (first number) and column size (second number) underneath it. In addition, we can subset a matrix by rows $\mathbf{X}_{i\bullet}$ or columns $\mathbf{X}_{\bullet j}$. In R, we use the following command to create a matrix:

```
matrix(data, nrow, ncol)
```

See `?matrix` for full list of inputs. R will construct a matrix from `data` going down each column in the first row and then moving to the second row. For example the following command creates the following matrix.

$$\texttt{matrix(1:4, 2, 2)} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

Row vector: A matrix of order $1 \times p$. Notationally, we will represent row vectors with a "prime" mark—e.g., $\mathbf{y}'$.

Column vector: A matrix of order $n \times 1$. Sometimes we may refer to this simply as a "vector," which is to say that we have coerced a one-dimensional vector into a two-dimensional column vector.

Note that in R, vectors will be coerced (i.e., transformed) into matrices when multiplied by a matrix. This coercion will sometimes be a column vector or a row vector depending on what is conformable.

Unit Matrix: A matrix containing all ones.

$$\begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}$$

This includes the special case of a ***vector of ones***, which we will denote as as a bolded one:

$$\underset{n\times 1}{\mathbf{1}} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

Note, we will often see the computation

$$\mathbf{11}' = \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}$$

which will produce a $ntimesn$ unit matrix. We can further simplify this by using the `matrix(1, n, n)` instead.

Zero Matrix: A matrix that includes only zeros.

$$\underset{p\times p}{\mathbf{0}} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

Square Matrix: A matrix with the same rows and columns.

Main Diagonal: The elements of a square matrix that have the same row and column number.

Off Diagonal: The elements of a square matrix that are not apart of the off diagonal. The ***upper diagonal*** are those elements where row $<$ column, and the lower ***lower diagonal*** are those elements where row $>$ column

Diagonal Matrix: A matrix with only values on the main diagonal and zeroes elsewhere.

Identity Matrix: A diagonal matrix with ones on the main diagonal and zeros elsewhere.

$$\underset{p\times p}{\mathbf{I}} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Triangular Matrix: A square matrix where either the lower diagonal (***lower triangular matrix***) or upper diagonal (***upper triangular matrix***) are all zeros.

1.2 Basic Matrix Operations

Transpose: We write the rows of a matrix as the columns of its transpose. We will denote transpose with the following notation:

$$\mathbf{A}'$$

In R we can transpose a matrix using the `t()` function.

Matrix Addition: Add two matrices with equal rows and columns together.

$$\underset{n\times p}{\mathbf{A}} + \underset{n\times p}{\mathbf{B}} = \underset{n\times p}{\mathbf{C}}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

In R we can add two matrices: `A + B`.

Matrix Subtraction: Subtract two matrices with equal rows and columns from each other.

$$\underset{n\times p}{\mathbf{A}} - \underset{n\times p}{\mathbf{B}} = \underset{n\times p}{\mathbf{C}}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} - \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1-5 & 2-6 \\ 3-7 & 4-8 \end{bmatrix} = \begin{bmatrix} -4 & -4 \\ -4 & -4 \end{bmatrix}$$

In R we can add two matrices: `A - B`.

Matrix Multiplication: Multiply two matrices with $n \times p$ and $p \times q$ dimensions to provide the resulting $n \times q$ dimensional matrix.

$$\underset{n\times p}{\mathbf{A}} \cdot \underset{p\times q}{\mathbf{B}} \equiv \mathbf{AB} = \underset{n\times q}{\mathbf{C}}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 7 & 8 \\ 9 & 10 \end{bmatrix} = \begin{bmatrix} (1\times 7)+(2\times 9) & (1\times 8)+(2\times 10) \\ (3\times 7)+(4\times 9) & (3\times 8)+(4\times 10) \\ (5\times 7)+(6\times 9) & (5\times 8)+(6\times 10) \end{bmatrix} = \begin{bmatrix} 25 & 28 \\ 57 & 64 \\ 89 & 100 \end{bmatrix}$$

In R we can add two matrices: `A %*% B`.

Matrix Inversion: Because division is not defined for matrix operations, we define a matrix's reciprocal (i.e., to the power of -1) as follows.

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

In other words, a matrix multiplied by its inverse results in an identity matrix. We can take the inverse of a matrix in R using the `solve()` function. Note that matrix inversion is typically slow and often optimized when implementing computations. For now, we will focus on implementing the formulas and not optimizing out the inversion.

Element Wise Operations: Also known as Hadamard product ($A \odot B$; `A * B` in R) and division ($A \oslash B$; `A / B` in R). These function in a similar fashion as matrix addition and subtraction but with multiplication and division.

Cholesky Decomposition: Defines a decomposition of a symmetric positive-definite matrix (e.g., a covariance matrix) that results in:

$$\mathbf{A} = \mathbf{L}\mathbf{L}'$$

where **L** is a lower triangular matrix that is the Cholesky factor. Note, for our purposes in this class, this operation will be analogous to taking the square root of a covariance matrix. Note that R returns the upper triangular factorization when using `chol()` function—i.e.:

$$\mathbf{A} = \mathbf{U}'\mathbf{U}$$

Scalar Operators: Scalar operators will apply the operation to every element of a matrix. For example, $n \cdot \mathbf{A}$ will multiply each element of A by n.

In R:

(1) Construct the following two matrices.

$$\mathbf{A} = \begin{bmatrix} 1 & 9 & 3 \\ 4 & 2 & 8 \\ 7 & 6 & 5 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 2 \\ 9 & 0 \\ 5 & 4 \end{bmatrix}$$

(2) Compute the following such that the result is a matrix:

(a) $\mathbf{A}_{\bullet 1} + \mathbf{B}_{\bullet 1}$

(b) $\mathbf{A}_{\bullet 2} - \mathbf{B}_{\bullet 2}$

(c) $\mathbf{A} \cdot \mathbf{B}$

(d) $2 \cdot \mathbf{B}'\mathbf{A} - \mathbf{1}\mathbf{B}'_{\bullet 1}$

(e) $\mathbf{B}_{\bullet 1}\mathbf{A}_{3\bullet}(2 \cdot \mathbf{I})$

Hint: When selecting a row or column vector from a matrix in R, the result will be a simplified vector (not a column or row vector). To prevent this behavior, we will need to include `drop = FALSE`:

```
mat[, i, drop = FALSE]
```

Answer:

Save the computations to `answers[[1]]`. The output should be a list with the following labels.

```
list(a, b, c, d, e)
```

2 Generating Multivariate Normally Distributed Data

Recall in Lab 1 that we generated bivariate normally distributed data. Consider Figure 1 (from Lab 1), a directed acyclic graph we used to generate bivariate normally distributed data. As discussed above, we can consider a Cholesky decomposition of a covariance matrix analogous to the square root. Therefore, we can use the Cholesky factorization of the covariance matrix to generate multivariate normally distributed data. Generally speaking, if we have a vector of p standard normal deviates, $\mathbf{z}$, we can transform them to be

$$\underset{p\times 1}{\mathbf{x}} \sim \mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{1}$$

via the following formula.

$$\mathbf{x}' = \boldsymbol{\mu}' + \mathbf{z}' \cdot \text{chol}(\boldsymbol{\Sigma})$$

We can further expand this computation in terms of matrices to obtain a $n \times p$ data matrix that is independent and identically distributed samples from the multivariate normal distribution as follows.

$$\underset{n\times p}{\mathbf{X}} = \mathbf{1}\boldsymbol{\mu}' + \mathbf{Z} \cdot \text{chol}(\boldsymbol{\Sigma}) \tag{2}$$

Each row of the resulting $\mathbf{X}$ matrix will be an independent draw from the Equation (1).

(A) (B)

Figure 1: Two Data Generating Processes for Bivariate Normal

To gain an intuition of how the Cholesky decomposition works, reconsider Figure 1. Usually, we graphically visualize this computation as Figure 1B because it is thought of transforming two variables to be correlated. Let us look at the symbolic solution to a Cholesky decomposition for a 2×2 matrix.

$$\text{chol}\left(\begin{bmatrix} a & b \\ b & c \end{bmatrix}\right) = \begin{bmatrix} \sqrt{a} & \frac{b}{\sqrt{a}} \\ 0 & \sqrt{c - \left(\frac{b}{\sqrt{a}}\right)^2} \end{bmatrix} \tag{3}$$

These computations may look somewhat familiar, but let us substitute our covariance matrix to make it more concrete.

$$\text{chol}\left(\begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{bmatrix}\right) = \begin{bmatrix} \sigma_1 & \frac{\sigma_{12}}{\sigma_1} \\ 0 & \sqrt{\sigma_2^2 - \left(\frac{\sigma_{12}}{\sigma_1}\right)^2} \end{bmatrix}$$

Substituting the above decomposition into Equation (2) gives us the following.

$$\begin{aligned} \mathbf{x}' &= \begin{bmatrix} \mu_1 & \mu_2 \end{bmatrix} + \begin{bmatrix} z_1 & z_2 \end{bmatrix} \begin{bmatrix} \sigma_1 & \frac{\sigma_{12}}{\sigma_1} \\ 0 & \sqrt{\sigma_2^2 - \left(\frac{\sigma_{12}}{\sigma_1}\right)^2} \end{bmatrix} \\ &= \begin{bmatrix} \mu_1 & \mu_2 \end{bmatrix} + \begin{bmatrix} (z_1 \times \sigma_1) + (0 \times z_2) & (z_1 \times \frac{\sigma_{12}}{\sigma_1}) + \left(z_2 \times \sqrt{\sigma_2^2 - \left(\frac{\sigma_{12}}{\sigma_1}\right)^2}\right) \end{bmatrix} \\ &= \begin{bmatrix} \mu_1 + \sigma_1 z_1 & \mu_2 + \left(\frac{\sigma_{12}}{\sigma_1}\right) z_1 + z_2 \sqrt{\sigma_2^2 - \left(\frac{\sigma_{12}}{\sigma_1}\right)^2} \end{bmatrix} \end{aligned}$$

This result should look familiar to Equations in Lab 1 (i.e., 4 and 10) but scaled using the covariance instead of the correlation. Importantly, this illustrates how even when we are generating data conceptually from Figure 1B, computationally, we are transforming the data via Figure 1A.

In R:

(1) Create a Cholesky decomposition function for a 2×2 matrix only by implementing Equation (3). The function should return the upper triangular matrix and also check to make sure the input matrix is 2×2.

Hint: You can use the `stop()` function to throw errors in R.

(2) Verify that the 2×2 Cholesky decomposition function maps onto the output from R's `chol()` function. Use $\mathbf{B}'\mathbf{B}$ to test the function.

(3) Provide an implementation for a function to generate data from a p-dimensional multivariate normal distribution:

```
rmvnorm(n, mu, Sigma)
```

Make sure the function verifies that the input dimensions are match.

Answer:

Set the seed to `87293` and generate $n = 10000$ samples using the `rmvnorm()` function with the following parameters.

$$\boldsymbol{\mu} = \begin{bmatrix} 12 \\ 32 \\ -9 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 144 & 28 & -14 \\ 28 & 64 & 4 \\ -14 & 4 & 9 \end{bmatrix}$$

Verify that the covariance matrix and mean vectors from the draws map onto what we would expect. Save a list with the empirical mean and covariance matrix using the following labels into `answers[[2]]`.

```
list(mu, Sigma)
```

3 Fundamental Statistics with Matrices

This section describes some fundamental statistical computations in terms of matrices. Note that the following computations are often not the optimal way to compute quantities but are useful nevertheless. To begin, we define the following data matrix.

$$\underset{n \times p}{\mathbf{X}} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$$

3.1 Arithmetic Means

Note, that all of these computations are much slower than using built-in R functions (e.g., `rowMeans`, `colMeans`), which are generally preferable.

Column Vector of Row Means

To compute the $n \times 1$ column vector of row means we use the following formula:

$$(p^{-1})\mathbf{X1} \tag{4}$$

where **1** is a $p \times 1$ vector of ones.

Column Vector of Column Means

To compute the $p \times 1$ column vector of row means we use the following formula:

$$(n^{-1})\mathbf{X}'\mathbf{1} \tag{5}$$

where **1** is a $n \times 1$ vector of ones.

In R:

Using the generated data from the previous section:

(1) Implement the computation for the row means of the given data matrix.

(2) Implement the computation for the column means of the given data matrix.

(3) Compute the mean for the entire data matrix using only matrices.

Answer:

Make sure to verify all computations using R's built-in functions. Save a list with the empirical row means, column means, and total mean using the following labels into `answers[[3]]`.

```
list(row, col, total)
```

3.2 Variation and Covariation of a Matrix

Sum of Square's and Cross Product Matrix

Recall the sum of squares for any score (i.e., the squared deviation around the mean),

$$SS_x = \sum_{i=1}^{n} (x_i - \overline{x})^2 \equiv \sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2$$

and the sum of products for any two scores (i.e., the product of the deviation around the mean for two variables).

$$SP_{xy} = \sum_{i=1}^{n} (x_i - \overline{x})(y - \overline{y}) \equiv \sum_{i=1}^{n} (x_i y_i) - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)$$

To compute the sum of squares for a given column vector ($\mathbf{x}_j = \mathbf{X}_{\bullet j}$), we use the second definitional formula and implement it via matrix computations.

$$\mathbf{x}_j'\mathbf{x}_j - \left(\frac{1}{n}\right)\mathbf{x}_j'\mathbf{1}\mathbf{1}'\mathbf{x}_j$$

Similarly, we can use two different vectors and compute the sum of products.

$$\mathbf{x}_i'\mathbf{x}_j - \left(\frac{1}{n}\right)\mathbf{x}_i'\mathbf{1}\mathbf{1}'\mathbf{x}_j$$

While this formula is useful, we are generally more interested in the matrix generalization of sum of squares known as the ***sum of squares and cross product (SSCP) matrix***.

$$\underset{p\times p}{\mathbf{SSCP}} = \mathbf{X}'\mathbf{X} = \begin{bmatrix} \sum x_1^2 & \sum x_1x_2 & \dots & \sum x_1x_p \\ \sum x_2x_1 & \sum x_2^2 & \dots & \sum x_2x_p \\ \vdots & \vdots & \ddots & \vdots \\ \sum x_px_1 & \sum x_px_2 & \dots & \sum x_p^2 \end{bmatrix}$$

The resulting matrix is square with $p \times p$ dimensions. We will see this computation quite often throughout this class. We have currently computed the ***uncorrected*** form, which only produces the appropriate sums of squares if $x_1, \dots, x_p$ are mean centered. We can correct the above matrix by subtracting out the means as we did before.

$$\underset{p\times p}{\mathbf{P}} = \mathbf{X}'\mathbf{X} - \left(n^{-1}\right)\mathbf{X}'\mathbf{1}\mathbf{1}'\mathbf{X} \tag{6}$$

Computing the Covariance Matrix

Recall that variances and covariances are the average SS and SP, respectively; thus, we can use the corrected SSCP matrix in Equation (6) to compute the covariance matrix for a sample as follows.

$$\underset{p\times p}{\mathbf{S}} = \left(\frac{1}{n-1}\right)\mathbf{P} \tag{7}$$

Computing the Correlation Matrix

Recall the formula for the Pearson correlation for X_1 and X_2.

$$\rho_{xy} = \frac{\sigma_{12}}{\sigma_1\sigma_2} \equiv \frac{SP_{12}}{\sqrt{SS_1}\sqrt{SS_2}}$$

To extend the above calculation to matrices, we must create a diagonal matrix equivalent to the denominator.

$$\mathbf{D}_\mathbf{P} = \sqrt{\text{diag}\left(\mathbf{P}\right)} = \begin{bmatrix} \sqrt{SS_1} & 0 & \dots & 0 \\ 0 & \sqrt{SS_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sqrt{SS_p} \end{bmatrix}$$

We can then pre and post multiply **P** by the inverse of our diagonal matrix to obtain the correlation matrix.

$$\underset{p\times p}{\mathbf{R}} = \mathbf{D}_\mathbf{P}^{-1}\mathbf{P}\mathbf{D}_\mathbf{P}^{-1} \tag{8}$$

Note that because $\mathbf{D}_\mathbf{P}$ is a diagonal matrix, its inverse equals taking the inverse of each diagonal element.

In R:

Using the generated data from the previous section:

(1) Implement the computation in Equation (6).

(2) Implement the computation in Equation (7).

(3) Implement the computation in Equation (8).

Answer:

Make sure to verify all computations using R's built-in functions. Save a list with the corrected SSCP, covariance matrix, and correlation matrix using the following labels into `answers[[4]]`.

```
list(sscp, cov_mat, cor_mat)
```

Measures of Covariation Between Two Matrices

Thus far, we have computed variation and covariation within a matrix. We will extend these formulas to examine the covariation between two different matrices. First, let us define our second matrix.

$$\underset{n\times q}{\mathbf{Y}} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1q} \\ y_{21} & y_{22} & \cdots & y_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nq} \end{bmatrix}$$

The extensions are a straightforward application of the previous equations. The critical difference is substituting the **Y** matrix in place of some of the **X** matrices, thus creating a product in place of squares.

Computing the covariation between two matrices.

$$\underset{p\times q}{\mathbf{P}_{\mathbf{XY}}} = \mathbf{X}'\mathbf{Y} - \left(n^{-1}\right)\mathbf{X}'\mathbf{1}\mathbf{1}'\mathbf{Y} \tag{9}$$

Computing the covariance between two matrices.

$$\underset{p\times q}{\mathbf{S}_{\mathbf{XY}}} = \left(\frac{1}{n-1}\right)\mathbf{P}_{\mathbf{XY}} \tag{10}$$

Computing the correlation between two matrices.

$$\underset{p\times q}{\mathbf{R}_{\mathbf{XY}}} = \mathbf{D}_{\mathbf{P}_{\mathbf{X}}}^{-1}\mathbf{P}\mathbf{D}_{\mathbf{P}_{\mathbf{Y}}}^{-1} \tag{11}$$

In R:

Create three functions that implement Equation (9), Equation (10), and Equation (11). Each function should have two inputs, `X` and `Y`, and output a matrix. As an additional requirement, if `Y` input is missing, the function ought to compute Equation (6), Equation (7), and Equation (8).

Hint: One way to accomplish this is by setting a default input.

Answer:

Use each function on the previously generated data matrix. Save a list with the corrected SSCP, covariance matrix, and correlation matrix using the following labels into `answers[[5]]`.

```
list(sscp, cov_mat, cor_mat)
```

Verify that `answers[[4]]` and `answers[[5]]` are the same. Save this result in the form `answers[[6]] <- list(identical = )`

Covariation Measures when Combining Two Matrices into One

As we might expect, many similar properties hold when we have composites from two different data matrices. The easiest way to think about this is by concatenating the two matrices. For example suppose we have a data matrix $\mathbf{M}$ partitioned into two parts $\mathbf{X}$ and $\mathbf{Y}$.

$$\underset{n\times(p+q)}{\mathbf{M}} = \left[\mathbf{X} \mid \mathbf{Y}\right] \equiv \left[\begin{array}{ccc|ccc} x_{11} & \dots & x_{1p} & y_{11} & \dots & y_{1q} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n1} & \dots & x_{np} & y_{n1} & \dots & y_{nq} \end{array}\right]$$

We can then apply many of the same formulas as before now onto the combined data matrix $\mathbf{M}$. For example, we can obtain the corrected SSCP matrix using the following formula.

$$\mathbf{P}_{\mathbf{M}} = \mathbf{M}'\mathbf{M} - \left(n^{-1}\right)\mathbf{M}'\mathbf{1}\mathbf{1}'\mathbf{M}$$

Conceptually we have created a partitioned matrix with four different partitions.

$$\mathbf{P}_{\mathbf{M}} = \left[\begin{array}{c|c} \mathbf{P}_{\mathbf{XX}} & \mathbf{P}_{\mathbf{XY}} \\ \hline \mathbf{P}_{\mathbf{YX}} & \mathbf{P}_{\mathbf{YY}} \end{array}\right]$$

As we might expect, the correlation and covariance matrix have similar partitions.

$$\mathbf{S}_{\mathbf{M}} = \left[\begin{array}{c|c} \mathbf{S}_{\mathbf{XX}} & \mathbf{S}_{\mathbf{XY}} \\ \hline \mathbf{S}_{\mathbf{YX}} & \mathbf{S}_{\mathbf{YY}} \end{array}\right] \quad \mathbf{R}_{\mathbf{M}} = \left[\begin{array}{c|c} \mathbf{R}_{\mathbf{XX}} & \mathbf{R}_{\mathbf{XY}} \\ \hline \mathbf{R}_{\mathbf{YX}} & \mathbf{R}_{\mathbf{YY}} \end{array}\right]$$

3.3 Linear Combinations

Linear combinations or ***composites*** are a common occurrence in multivariate statistics, and they will be vital as we discuss simulating data from regression models. We construct a linear combination, U, as a sum of p variables multiplied by a weight, a.

$$u_i = \sum_{j=1}^{n} a_j x_{ij} \equiv a_1 x_{i1} + a_2 x_{i2} + \ldots + a_p x_{ip}$$

A composite we often use is the predicted score in multiple regression analyses. More generally, we can compute composites for all observations via matrix multiplication as follows

$$\underset{n\times 1}{\mathbf{u}} = \underset{n\times p}{\mathbf{X}} \cdot \underset{p\times 1}{\mathbf{a}} \tag{12}$$

with the matrices written out below.

$$\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix}$$

Statistical Properties of Linear Combinations

An essential feature of a linear combination is that we can compute the composite's first and second moments (i.e., mean and variance, respectively) via transformation. Returning to our example, we compute the mean of U ($\overline{U}$) by taking a weighted sum of the mean vector for X ($\overline{\mathbf{X}}$).

$$\overline{U} = \overline{\mathbf{X}}'\mathbf{a} = \begin{bmatrix} \overline{X}_1 & \overline{X}_2 & \dots & \overline{X}_p \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} \tag{13}$$

To compute the sum of squares for a composite, let's first plug U into Equation (6).

$$SS_u = \mathbf{u}'\mathbf{u} - \left(n^{-1}\right)\mathbf{u}'\mathbf{1}\mathbf{1}'\mathbf{X}$$

Now we can substitute Equation (12) into the above expression to obtain the follow.

$$SS_u = \mathbf{a}'\mathbf{X}'\mathbf{X}\mathbf{a} - \left(n^{-1}\right)\mathbf{a}'\mathbf{X}'\mathbf{1}\mathbf{1}'\mathbf{X}\mathbf{a}$$

Finally, we can factor out the weights to obtain the following result.

$$\begin{aligned} SS_u &= \mathbf{a}'\left(\mathbf{X}'\mathbf{X} - \left(n^{-1}\right)\mathbf{X}'\mathbf{1}\mathbf{1}'\mathbf{X}\right)\mathbf{a} \\ &= \mathbf{a}'\mathbf{P}_\mathbf{X}\mathbf{a} \end{aligned} \tag{14}$$

Thus, just like we did with the mean, we can compute the measure of variation by transforming it based on the weights. The above form (weight vector multiplied by matrix multiplied by weight vector) is known as a ***quadrtic form*** and will often appear throughout the semester. As we might expect, this transformation holds for the variance as well.

$$s_u^2 = \mathbf{a}'\mathbf{S}_\mathbf{X}\mathbf{a} \tag{15}$$

Often multiple linear combinations may be computed from the same base variables. As we might expect, this results in unique statistical properties that can be computed via transformations. To illustrate, suppose we have another linear combination.

$$\mathbf{v} = \mathbf{X}\mathbf{b}$$

We can compute the covariation between **u** and **v** using the quadratic form.

$$SP_{uv} = \mathbf{a}'\mathbf{P}_\mathbf{X}\mathbf{b} \tag{16}$$

And these computations extend to the covariance

$$s_{uv} = \mathbf{a}'\mathbf{S}_\mathbf{X}\mathbf{b} \tag{17}$$

and correlation.

$$r_{uv} = \frac{\mathbf{a}'\mathbf{P}_\mathbf{X}\mathbf{b}}{\sqrt{\mathbf{a}'\mathbf{P}_\mathbf{X}\mathbf{a}}\sqrt{\mathbf{b}'\mathbf{P}_\mathbf{X}\mathbf{b}}} \tag{18}$$

In R:

(1) Using the previously generated data matrix create two composites with the following weights.

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ -5 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -2 \\ 7 \\ 3 \end{bmatrix}$$

(2) Using **a** and **b**:

(a) Implement the computation for Equation (16).

(b) Implement the computation for Equation (17).

(c) Implement the computation for Equation (18).

(3) Use the created functions to compute the corrected sum of products between the two composites, the covariance between the two composites, and the correlation via the actual composite scores. Make sure to verify that they match.

Answer:

In `answers[[7]]`, store a list with the results from the implemented computations and the following labels.

```
list(sp, cov, cor)
```

In `answers[[8]]`, store a list with the results from using the previously created functions and the following labels.

```
list(sp, cov, cor)
```

4 Some Array Specific Functions in R

Like any language, R has specific functions to simplify and speed up computations. This section will discuss some major ones but is by no means exhaustive.

4.1 `crossprod` and `tcrossprod`

The `crossprod()` and `tcrossprod()` functions are a useful set of functions to handle transpose multiply operations. The `crossprod(X, Y)` function is a computationally faster implementation of the following command.

```
t(X) %*% Y
```

If we do not specify a `Y` value, we obtain the following command,

```
t(X) %*% X
```

which, if this Lab has taught you anything, will be used regularly to compute the SSCP matrix. Note, the `tcrossprod(X, Y)` is similar to the `crossprod()` function, but it computes the following

```
Y %*% t(Y)
```

With both functions, we can see the full documentation by using `?crossprod`.

To illustrate the speed increase, we can demonstrate using the `microbenchmark` package. This package is helpful in checking for speed optimizations on specific code. The microbenchmark package runs a simulation that times the speed it takes to run each command a user-specified number of times.

```
X <- matrix(1:10, 10000, 100)
microbenchmark::microbenchmark(
    t(X) %*% X,
    crossprod(X),
    times = 1000
)

## Output will vary depending on run
#Unit: milliseconds
#         expr      min       lq     mean    median        uq      max neval
#   t(X) %*% X 7.546747 9.282686 13.97149 12.059525 15.394872 94.19097  1000
# crossprod(X) 2.817457 3.617813  5.93218  3.994165  6.612856 67.33690  1000
```

The rather large computation, on average, took about half the time to run using the `crossprod` function; however, notice the scale we are looking at, milliseconds. This demonstrates another important aspect of any optimizations, the utility. The above results may be rather unrealistic for many of our simulations and even then are only shaving off about 6 milliseconds on average. As a cost for a minor speed up, the implementation details start to lose some readability and do not directly map onto how we might view the equation.

4.2 sweep function

The `sweep` function in R provides a useful tool for subtracting out values and vectors from matrices. Below is the major inputs that will be used throughout this course.

```
sweep(x, MARGIN, STATS, FUN = "-")
```

The definitions of the inputs are as follows:

- The `x` input is our matrix that we want to perform some sweep on.
- The `MARGIN` input is the dimensions that correspond to the dimensions of `STATS`. For example, a `MARGIN = 1` would mean that `STATS` is in reference to each row.
- The `STATS` is a scalar, vector, or matrix that we want to sweep out from the matrix.
- The `FUN` is a function that will carry out the sweeping. For example, the default is to subtract (i.e., `"-"`) the `STATS` across each `MARGIN`.

While these descriptions may seem abstract, let's look at how the operations perform in R.

```
# Create Data to sweep
X <- matrix(1:6, 3, 2)
# Create values to sweep out
y <- matrix(c(5, 10), 2)

## Print X
X

#     [,1] [,2]
#[1,]    1    4
#[2,]    2    5
#[3,]    3    6
```

Let's first sweep out `y` from `X` going down each column:

```
## Subtract
sweep(X, 2, y)

#     [,1] [,2]
#[1,]   -4   -6
#[2,]   -3   -5
#[3,]   -2   -4
```

As we can see, the `sweep` function subtracts out `y` from each row. This may seem counterintuitive, but `MARGIN` is in reference to the dimensions that belongs to `y`. So a `MARGIN = 2` is saying that `y` values should be subtracted across columns. Here are some other common sweep operations:

```
## Add
sweep(X, 2, y, '+')

#     [,1] [,2]
#[1,]    6   14
#[2,]    7   15
#[3,]    8   16

## Divide
sweep(X, 2, y, '/')
#     [,1] [,2]
#[1,]  0.2  0.4
#[2,]  0.4  0.5
#[3,]  0.6  0.6
```

In R: Create a new function, `rmvnorm_fast`, that implements `rmvnorm` with the inclusion of the `sweep` function. Run a simulation using microbenchmark to investigate the performance difference. This simulation should have a total of 500 replications, and each function should be used to generate a 1000×100 data matrix.

Answer:

Copy the output from microbenchmark and include it as a comment in the code.