A review of linear algebra: Applications in R

Notes for a course in Psychometric Theory
to accompany *Psychometric Theory with Applications in R*

William Revelle

Department of Psychology
Northwestern University
Evanston, Illinois USA

October, 2018

Outline

Why linear algebra?

- Linear algebra is the fundamental notational technique used in multiple correlation, factor analysis, and structural equation modeling
- Although it is possible to use R without understanding linear algebra, it is helpful to do so.
- Linear algebra is a convenient notational system that allows us to think about data at a higher (broader) level rather than data point by data point.

Linear Algebra

- Matrices were used by the Babylonians and Chinese (ca. 100 BCE) to do basic calculations and solve simultaneous equations but were not introduced in Western mathematics until the early 19th century.
- Introduced to psychologists by Thurstone in 1933 who had learned about them from a mathematician colleague.
  - Until then, all analysis was done on "tables" with fairly laborious ad hoc procedures.
- Matrices may be thought of as "spreadsheets" but with their own algebra.
- Commercial stats programs do their calculations in linear algebra but "protect" the user from their seeming complexity.
- R is explicit in its use of matrices, so am I.

Scalers, Vectors and Matrices

- A *scaler* is just a single value, an integer or a real number.
- A ***vector*** is a one dimensional array of n elements where the most frequently used elements are integers, reals (numeric), characters, or logical.
  - ***vectors*** have length $\boldsymbol{x} = \begin{pmatrix} 42 & 17 & 3 & 2 & 9 & 4 \end{pmatrix}$
  - elements are indexed by location in the vector. x_i is the i^{th} element. $\boldsymbol{x}_2 = 17$
- A *matrix* is a two dimensional array of m vectors, each with n elements.
  - ***Matrices*** have 2 dimensions (rows and columns) ${}_{\boldsymbol{r}}\boldsymbol{X}_{\boldsymbol{c}}$ e.g., ${}_{\mathbf{2}}\boldsymbol{X}_{\mathbf{6}} = \begin{pmatrix} 42 & 17 & 3 & 2 & 9 & 4 \\ 39 & 21 & 7 & 4 & 8 & 6 \end{pmatrix}$
  - elements are indexed by location in the matrix. $X_{i,j}$ is the element in the i^{th} row and j^{th} column. $\boldsymbol{X}_{2,3} = 7$
- (In an attempt at consistent notation, vectors will be ***bold*** faced lower case letters, matrices will be ***CAPITALIZED***).

Basic operations

- Basic operations on a ***vector*** or a ***matrix*** are addition, subtraction and multiplication.
- First consider addition, subtraction and multiplication by scalers.
- Consider ***v1*** = the first 6 integers, and ***v2*** = the next 6 integers:

```
> v1 <- seq(1, 6)
> v2 <- seq(7, 12)
> v3 <- v1 + 20

> v1
[1] 1 2 3 4 5 6
> v2
[1]  7  8  9 10 11 12
> v3
[1] 21 22 23 24 25 26
```

Basic operations

We can add a constant to each element in a vector, add each element of the first vector to the corresponding element of the second vector, multiply each element by a *scaler*, or multiply each element in the first by the corresponding element in the second:

```
> v3 <- v1 + 20
> v4 <- v1 + v2
> v5 <- v1 * 3
> v6 <- v1 * v2
> v3
[1] 21 22 23 24 25 26
> v4
[1]  8 10 12 14 16 18
> v5
[1]  3  6  9 12 15 18
> v6
[1]  7 16 27 40 55 72
```

row and column vectors and the transpose operator

- vectors can be either row vectors or column vectors.
- the *transpose*, t, of a row vector is a column vector and vice versa

$$\boldsymbol{v1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$$

$$t(\boldsymbol{v1}) = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix}$$

Outer product = multiplication of a column vector by a row vector

Although addition and subtraction are straightforward, multiplication is somewhat more complicated, for the order in which two vectors are multiplied changes the result. That is $\boldsymbol{ab} \neq \boldsymbol{ba}$. A column vector times a row vector (also known as the outer product or the tensor product) yields a matrix but a row vector times a column vectors (the dot product) yields a scaler. Consider $\boldsymbol{v2} \otimes \boldsymbol{v1}$

$$\begin{pmatrix} 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{pmatrix} \%*\% \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix} = \begin{pmatrix} 7 & 14 & 21 & 28 & 35 & 42 \\ 8 & 16 & 24 & 32 & 40 & 48 \\ 9 & 18 & 27 & 36 & 45 & 54 \\ 10 & 20 & 30 & 40 & 50 & 60 \\ 11 & 22 & 33 & 44 & 55 & 66 \\ 12 & 24 & 36 & 48 & 60 & 72 \end{pmatrix}$$

Vector multiplication of a row vector by a column vector

But the dot product (or inner product) of a row vector by a column vector is a scaler. Consider ***v1*** · ***v2***

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix} \%*\% \begin{pmatrix} 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{pmatrix} = \sum_{i=1}^{n} v1_i v2_i = \sum_{i=1}^{n} v6_i = 217$$

It is this operation, the dot product which is a very powerful matrix operation, for it does summations of products in one line. This inner product will become even more useful with matrices. In both the inner and outer product, the same rule is followed: the i^{th}, j^{th} element of the result is the sum of the products of the i^{th} row of the first vector and the j^{th} column of the second vector.

More on outer products

It is important to realize that the dimensions of the vectors must match to do either inner or outer products. Consider $\underset{(6x1)}{\boldsymbol{v1}} \bigotimes \underset{(1x4)}{\boldsymbol{v7}'}$ and $\underset{(4x1)}{\boldsymbol{v7}} \bigotimes \underset{(1x6)}{\boldsymbol{v1}'}$ which can be done,

$$\underset{(6x1)}{\boldsymbol{v1}} \,\%*\%\, \underset{(1x4)}{\boldsymbol{v7}'} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix} \%*\% \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 8 & 12 & 16 \\ 5 & 10 & 15 & 20 \\ 6 & 12 & 18 & 24 \end{pmatrix} = \underset{(6x4)}{\boldsymbol{V8}} \tag{1}$$

and

$$\underset{(4x1)}{\boldsymbol{v7}} \,\%*\%\, \underset{(1x6)}{\boldsymbol{v1}'} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} \%*\% \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 6 & 8 & 10 & 12 \\ 3 & 6 & 9 & 12 & 15 & 18 \\ 4 & 8 & 12 & 16 & 20 & 24 \end{pmatrix} = \underset{(4x6)}{\boldsymbol{V9}} \tag{2}$$

but that $\underset{(6x1)}{\boldsymbol{v1}} \bigotimes \underset{(4x1)}{\boldsymbol{v7}}$ can not.

Matrices and data

- A *matrix* is just a two dimensional (rectangular) organization of numbers.
  - It is a vector of vectors.
- For data analysis, the typical data matrix is organized with rows containing the responses of a particular subject and the columns representing different variables.
  - Thus, a 6 x 4 data matrix (6 rows, 4 columns) would contain the data of 6 subjects on 4 different variables.
- In the example below the matrix operation has taken the numbers 1 through 24 and organized them column wise. That is, a matrix is just a way (and a very convenient one at that) of organizing a data vector in a way that highlights the correspondence of multiple observations for the same individual. (The matrix is an ordered n-tuplet where n is the number of columns).

Matrices in R

R provides numeric row and column names (e.g., [1,] is the first row, [,4] is the fourth column, but it is useful to label the rows and columns to make the rows (subjects) and columns (variables) distinction more obvious. We do this using the `rownames` and `colnames` functions, combined with the `paste` and `seq` functions.

```
> Xij <- matrix(seq(1:24), ncol = 4)
> rownames(Xij) <- paste("S", seq(1, dim(Xij)[1]), sep = "")
> colnames(Xij) <- paste("V", seq(1, dim(Xij)[2]), sep = "")
> Xij
```

```
   V1 V2 V3 V4
S1  1  7 13 19
S2  2  8 14 20
S3  3  9 15 21
S4  4 10 16 22
S5  5 11 17 23
S6  6 12 18 24
```

Transpose of a matrix

Just as the *transpose of a vector* makes a column vector into a row vector, so does the *transpose of a matrix* swap the rows for the columns. Applying the `t` function to the matrix ***Xij*** produces ***Xij′***. Note that now the subjects are columns and the variables are the rows.

```
> t(Xij)
```

```
   S1 S2 S3 S4 S5 S6
V1  1  2  3  4  5  6
V2  7  8  9 10 11 12
V3 13 14 15 16 17 18
V4 19 20 21 22 23 24
```

Adding or multiplying by a scaler

Just as we could with vectors, we can add, subtract, multiply or divide the matrix by a *scalar* (a number without a dimension).

```
> Xij + 4
```

```
   V1 V2 V3 V4
S1  5 11 17 23
S2  6 12 18 24
S3  7 13 19 25
S4  8 14 20 26
S5  9 15 21 27
S6 10 16 22 28
```

```
> round((Xij + 4)/3, 2)
```

```
     V1   V2   V3   V4
S1 1.67 3.67 5.67 7.67
S2 2.00 4.00 6.00 8.00
S3 2.33 4.33 6.33 8.33
S4 2.67 4.67 6.67 8.67
S5 3.00 5.00 7.00 9.00
S6 3.33 5.33 7.33 9.33
```

Multiplying by a vector, caution required

```
> v <- 1:4

[1] 1 2 3 4

> Xij + v

   V1 V2 V3 V4
S1  2 10 14 22
S2  4 12 16 24
S3  6 10 18 22
S4  8 12 20 24
S5  6 14 18 26
S6  8 16 20 28

> Xij * v

   V1 V2 V3 V4
S1  1 21 13 57
S2  4 32 28 80
S3  9  9 45 21
S4 16 20 64 44
S5  5 33 17 69
S6 12 48 36 96
```

- We can also add or multiply each row (or column, depending upon order) by a vector.
  - This is more complicated that it would appear, for R does the operations columnwise.
  - This is best seen in an example:
- This is not what we expected!

Multiplying by a vector, transpose and then transpose again

```
>  t(t(Xij) + v)

   V1 V2 V3 V4
S1  2  9 16 23
S2  3 10 17 24
S3  4 11 18 25
S4  5 12 19 26
S5  6 13 20 27
S6  7 14 21 28

> V10 <-  t(t(Xij) * v)
> V10

   V1 V2 V3 V4
S1  1 14 39 76
S2  2 16 42 80
S3  3 18 45 84
S4  4 20 48 88
S5  5 22 51 92
S6  6 24 54 96
```

- These are not the expected results if the intent was to add or multiply a different number to each column!
- R operates on the columns and wraps around to the next column to complete the operation.
- To add the n elements of ***v*** to the n columns of ***Xij***,
  - use the `t` function to transpose ***Xij***
  - and then transpose the result back to the original order:

The scale function

To find a matrix of deviation scores, just subtract the means vector from each cell. The `scale` function does this with the option scale=FALSE. The default for `scale` is to convert a matrix to standard scores.

```
> scale(V10,scale=FALSE)

     V1 V2   V3  V4
S1 -2.5 -5 -7.5 -10
S2 -1.5 -3 -4.5  -6
S3 -0.5 -1 -1.5  -2
S4  0.5  1  1.5   2
S5  1.5  3  4.5   6
S6  2.5  5  7.5  10
attr(,"scaled:center")
  V1   V2   V3   V4
 3.5 19.0 46.5 86.0
```

```
> round(scale(v10),2)

      V1    V2    V3    V4
S1 -1.34 -1.34 -1.34 -1.34
S2 -0.80 -0.80 -0.80 -0.80
S3 -0.27 -0.27 -0.27 -0.27
S4  0.27  0.27  0.27  0.27
S5  0.80  0.80  0.80  0.80
S6  1.34  1.34  1.34  1.34
attr(,"scaled:center")
  V1   V2   V3   V4
 3.5 19.0 46.5 86.0
attr(,"scaled:scale")
      V1       V2       V3       V4
1.870829 3.741657 5.612486 7.483315
```

Matrix multiplication

Matrix multiplication is a combination of multiplication and addition and is one of the most used and useful matrix operations. For a matrix $\underset{(rxp)}{\boldsymbol{X}}$ of dimensions r*p and $\underset{(pxc)}{\boldsymbol{Y}}$ of dimension p * c, the product, $\underset{(rxp)(pxc)}{\boldsymbol{X}\ \boldsymbol{Y}}$, is a r * c matrix where each element is the sum of the products of the rows of the first and the columns of the second. That is, the matrix $\underset{(rxc)}{\boldsymbol{XY}}$ has elements xy_{ij} where each

$$xy_{ij} = \sum_{k=1}^{n} x_{ik} * y_{kj}$$

Matrix multiplication

The resulting x_{ij} cells of the product matrix are sums of the products of the column elements of the first matrix times the row elements of the second. There will be as many cell as there are rows of the first matrix and columns of the second matrix.

$$\underset{(r_x \text{ x } p)(p \text{ x } c_y)}{\boldsymbol{XY}} = \xrightarrow{\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \end{pmatrix}} \downarrow \begin{pmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \\ y_{31} & y_{32} \\ y_{41} & y_{42} \end{pmatrix} = \begin{pmatrix} \sum_i^p x_{1i}y_{i1} & \sum_i^p x_{1i}y_{i2} \\ \sum_i^p x_{2i}y_{i1} & \sum_i^p x_{2i}y_{i2} \end{pmatrix}$$

It should be obvious that matrix multiplication is a very powerful operation, for it represents in one product the r * c summations taken over p observations.

Matrix addition

Analogous to matrix multiplication is a function to add elements from row and column vectors to fill a complete matrix. This is a non-standard function %+%

$$\underset{(r_x \text{ x } p)(p \text{ x } c_y)}{\boldsymbol{XY}} = \begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \end{pmatrix} \rightarrow \downarrow \begin{pmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \\ y_{31} & y_{32} \\ y_{41} & y_{42} \end{pmatrix} = \begin{pmatrix} \sum_i^p x_{1i} + y_{i1} & \sum_i^p x_{1i} + y_{i2} \\ \sum_i^p x_{2i} + y_{i1} & \sum_i^p x_{2i} + y_{i2} \end{pmatrix}$$

It should be obvious that matrix addition is a very powerful operation, for it represents in one operation the r * c summations taken over p observations.

Note that matrix addition done this way is a function unique to the *psych* package. (Adapted from Krus, D. J. (2001) Matrix addition. Journal of Visual Statistics, 1, (February, 2001)

Examples of matrix "addition" and normal matrix addition

R code

```
x <- c(1,2,3,4)
y <- x + x  #normal vector addition adds corresponding elements
y

xx <- x %+% t(x) #"addition" adds row entries to  column entries
xx

x %*% t(x)  #matrix multiplication
```

```
[1] 2 4 6 8      #normal vector addtion

     [,1] [,2] [,3] [,4]   #special addition
[1,]    2    3    4    5
[2,]    3    4    5    6
[3,]    4    5    6    7
[4,]    5    6    7    8

     [,1] [,2] [,3] [,4]   #matrix multiplication
[1,]    1    2    3    4
[2,]    2    4    6    8
[3,]    3    6    9   12
[4,]    4    8   12   16
```

Matrix multiplication can be used with vectors as well as matrices. Consider the product of a vector of ones, **1**, and the matrix $\underset{(rxc)}{\boldsymbol{Xij}}$ with 6 rows of 4 columns. Call an individual element in this matrix x_{ij}. Then the sum for each column of the matrix is found multiplying the matrix by the "one" vector with ***Xij***. Dividing each of these resulting sums by the number of rows (cases) yields the mean for each column. That is, find

$$\mathbf{1}'\boldsymbol{Xij} = \sum_{i=1}^{n} X_{ij}$$

for the c columns, and then divide by the number (n) of rows. Note that the same result is found by the `colMeans(Xij)` function.

$$\mathbf{1}'\,\boldsymbol{Xij}\frac{1}{n} = \xrightarrow{\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}} \downarrow \begin{pmatrix} 1 & 7 & 13 & 19 \\ 2 & 8 & 14 & 20 \\ 3 & 9 & 15 & 21 \\ 4 & 10 & 16 & 22 \\ 5 & 11 & 17 & 23 \\ 6 & 12 & 18 & 24 \end{pmatrix} \frac{1}{6} = \begin{pmatrix} 21 & 57 & 93 & 129 \end{pmatrix} \frac{1}{6} =$$

$$\begin{pmatrix} 3.5 & 9.5 & 15.5 & 21.5 \end{pmatrix}$$

Means for columns

We can use the `dim` function to find out how many cases (the number of rows) or the number of variables (number of columns). `dim` has two elements: dim(***Xij***)[1] = number of rows, dim(***Xij***)[2] is the number of columns.

```
> dim(Xij)

[1] 6 4

 #a vector of 1s
> one <- rep(1,dim(Xij)[1])
#find the column sum
> t(one)  %*% Xij

     V1 V2 V3  V4
[1,] 21 57 93 129

 #find the column average
> X.means <-  t(one) %*% Xij /dim(Xij)[1]

  V1   V2   V3   V4
 3.5  9.5 15.5 21.5
```

Or, just use the `colMeans` function:

```
> colMeans(Xij)

   V1   V2   V3   V4
 3.5  9.5 15.5 21.5
```

See `rowMeans` for the equivalent for rows.

Deviation scores

To form a matrix of deviation scores, where the elements of each column are deviations from that column mean, it is necessary to either do the operation on the transpose of the ***Xij*** matrix, or to create a matrix of means by premultiplying the means vector by a vector of ones and subtracting this from the data matrix.

```
>  X.diff <- Xij - one %*% X.means
```

```
> X.diff
     V1   V2   V3   V4
S1 -2.5 -2.5 -2.5 -2.5
S2 -1.5 -1.5 -1.5 -1.5
S3 -0.5 -0.5 -0.5 -0.5
S4  0.5  0.5  0.5  0.5
S5  1.5  1.5  1.5  1.5
S6  2.5  2.5  2.5  2.5
```

Variances and covariances

Variances and covariances are measures of dispersion around the mean. We find these by first subtracting the means from all the observations. This means centered matrix is the original matrix minus a vector of means. To make a more interesting data set, randomly order (in this case, `sample` without replacement) from the items in ***Xij*** and then find the ***X.means*** and ***X.diff*** matrices.

```
> set.seed(42) #set random seed for a repeatable example
> Xij <- matrix(sample(Xij),ncol=4) #random sample from Xij
> rownames(Xij) <- paste("S", seq(1, dim(Xij)[1]), sep = "")
> colnames(Xij) <- paste("V", seq(1, dim(Xij)[2]), sep = "")
> Xij
```

```
   V1 V2 V3 V4
S1 22 14 12 15
S2 24  3 17  6
S3  7 11  5  4
S4 18 16  9 21
S5 13 23  8  2
S6 10 19  1 20
```

```
> X.means <-  t(one) %*% Xij /dim(Xij)[1] #find the column average
> X.diff <- Xij -one %*% X.means
> X.diff
```

```
           V1           V2          V3         V4
S1  6.333333  -0.3333333  3.3333333  3.666667
S2  8.333333 -11.3333333  8.3333333 -5.333333
S3 -8.666667  -3.3333333 -3.6666667 -7.333333
S4  2.333333   1.6666667  0.3333333  9.666667
S5 -2.666667   8.6666667 -0.6666667 -9.333333
S6 -5.666667   4.6666667 -7.6666667  8.666667
```

Compare this result to just using the `scale` function to mean center the data:

```
X.cen <- scale(Xij,scale=FALSE).
```

Variances and covariances

To find the variance/covariance matrix, find the the matrix product of the means centered matrix ***X.diff*** with itself and divide by n-1. Compare this result to the result of the `cov` function (the normal way to find covariances). The differences between these two results is the rounding to whole numbers for the first, and to two decimals in the second.

```
> X.cov <- t(X.diff) %*% X.diff /(dim(X.diff)[1]-1)
> round(X.cov)
```

```
     V1  V2  V3  V4
V1   46 -23  34   8
V2 -23   48 -25  12
V3   34 -25  31 -12
V4    8  12 -12  70
```

```
> round(cov(Xij),2)
```

```
       V1     V2     V3     V4
V1  45.87 -22.67  33.67   8.13
V2 -22.67  47.87 -24.87  11.87
V3  33.67 -24.87  30.67 -12.47
V4   8.13  11.87 -12.47  70.27
```

- Some operations need to find just the *diagonal* of the matrix.
  - For instance, the diagonal of the matrix ***X.cov*** (found above) contains the variances of the items.
  - To extract just the diagonal, or create a matrix with a particular diagonal we use the `diag` command.
- We can convert the covariance matrix ***X.cov*** to a correlation matrix ***X.cor*** by pre and post multiplying the covariance matrix with a diagonal matrix containing the reciprocal of the standard deviations (square roots of the variances).
- Remember that the correlation, r_{xy}, is the ratio of the covariance to the squareroot of the product of the variances:

$$r_{xy} = \frac{C_{xy}}{\sqrt{V_x V_y}} = \frac{C_{xy}}{\sigma_x \sigma_y}.$$

Correlations from linear algebra

```
> X.var <- diag(X.cov)

      V1       V2       V3       V4
45.86667 47.86667 30.66667 70.26667

> sdi <- diag(1/sqrt(diag(X.cov)))
> colnames(sdi) <- colnames(X.cov)
> rownames(sdi) <- colnames(X.cov)
> round(sdi, 2)

     V1   V2   V3   V4
V1 0.15 0.00 0.00 0.00
V2 0.00 0.14 0.00 0.00
V3 0.00 0.00 0.18 0.00
V4 0.00 0.00 0.00 0.12
```

```
> X.cor <- sdi %*% X.cov %*% sdi #pre an
> rownames(X.cor) <- colnames(X.cor) <-
> round(X.cor, 2)

     V1    V2    V3    V4
V1  1.00 -0.48  0.90  0.14
V2 -0.48  1.00 -0.65  0.20
V3  0.90 -0.65  1.00 -0.27
V4  0.14  0.20 -0.27  1.00
```

Compare this to the standard command for finding correlations `cor`.

```
> round(cor(Xij), 2)

     V1    V2    V3    V4
V1  1.00 -0.48  0.90  0.14
V2 -0.48  1.00 -0.65  0.20
V3  0.90 -0.65  1.00 -0.27
V4  0.14  0.20 -0.27  1.00
```

The identity matrix

The *identity matrix* is merely that matrix, which when multiplied by another matrix, yields the other matrix. (The equivalent of 1 in normal arithmetic.) It is a diagonal matrix with 1 on the diagonal.

```
> I <- diag(1,nrow=dim(X.cov)[1])
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
```

Simultaneous equations without matrices

Many problems in data analysis require solving a system of simultaneous equations. For instance, in multiple regression with two predictors and one criterion with a set of correlations of:

$$\left\{ \begin{array}{ccc} r_{x1x1} & r_{x1x2} & r_{x1y} \\ r_{x1x2} & r_{x2x2} & r_{x2y} \\ r_{x1y} & r_{x2y} & r_{yy} \end{array} \right\} \tag{3}$$

we want to find the find weights, β_i, that when multiplied by x_1 and x_2 maximize the correlations with y. That is, we want to solve the two simultaneous equations

$$\left\{ \begin{array}{l} r_{x1x1}\beta_1 + r_{x1x2}\beta_2 = r_{x1y} \\ r_{x1x2}\beta_1 + r_{x2x2}\beta_2 = r_{x2y} \end{array} \right\}. \tag{4}$$

Solving two simultaneous equations

We can directly solve these two equations by adding and subtracting terms to the two such that we end up with a solution to the first in terms of β_1 and to the second in terms of β_2:

$$\left\{ \begin{array}{l} \beta_1 + r_{x1x2}\beta_2/r_{x1x1} = r_{x1y}/r_{x1x1} \\ r_{x1x2}\beta_1/r_{x2x2} + \beta_2 = r_{x2y}/r_{x2x2} \end{array} \right\} \tag{5}$$

which becomes

$$\left\{ \begin{array}{l} \beta_1 = (r_{x1y} - r_{x1x2}\beta_2)/r_{x1x1} \\ \beta_2 = (r_{x2y} - r_{x1x2}\beta_1)/r_{x2x2} \end{array} \right\} \tag{6}$$

Substituting the second row of (6) into the first row, and vice versa we find

$$\left\{ \begin{array}{l} \beta_1 = (r_{x1y} - r_{x1x2}(r_{x2y} - r_{x1x2}\beta_1)/r_{x2x2})/r_{x1x1} \\ \beta_2 = (r_{x2y} - r_{x1x2}(r_{x1y} - r_{x1x2}\beta_2)/r_{x1x1})/r_{x2x2} \end{array} \right\}$$

Solving simultaneous equations – continued

Collecting terms, we find:

$$\left\{ \begin{array}{c} \beta_1 r_{x1x1} r_{x2x2} = (r_{x1y} r_{x2x2} - r_{x1x2}(r_{x2y} - r_{x1x2}\beta_1)) \\ \beta_2 r_{x2x2} r_{x1x1} = (r_{x2y} r_{x1x1} - r_{x1x2}(r_{x1y} - r_{x1x2}\beta_2) \end{array} \right\}$$

and rearranging once again:

$$\left\{ \begin{array}{c} \beta_1 r_{x1x1} r_{x2x2} - r^2_{x1x2}\beta_1 = (r_{x1y} r_{x2x2} - r_{x1x2}(r_{x2y}) \\ \beta_2 r_{x1x1} r_{x2x2} - r^2_{x1x2}\beta_2 = (r_{x2y} r_{x1x1} - r_{x1x2}(r_{x1y} \end{array} \right\}$$

Struggling on:

$$\left\{ \begin{array}{c} \beta_1 (r_{x1x1} r_{x2x2} - r^2_{x1x2}) = r_{x1y} r_{x2x2} - r_{x1x2} r_{x2y} \\ \beta_2 (r_{x1x1} r_{x2x2} - r^2_{x1x2}) = r_{x2y} r_{x1x1} - r_{x1x2} r_{x1y} \end{array} \right\}$$

And finally:

$$\left\{ \begin{array}{c} \beta_1 = (r_{x1y} r_{x2x2} - r_{x1x2} r_{x2y})/(r_{x1x1} r_{x2x2} - r^2_{x1x2}) \\ \beta_2 = (r_{x2y} r_{x1x1} - r_{x1x2} r_{x1y})/(r_{x1x1} r_{x2x2} - r^2_{x1x2}) \end{array} \right\}$$

Using matrices to solve simultaneous equations

Alternatively, these two equations (4) may be represented as the product of a vector of unknowns (the βs) and a matrix of coefficients of the predictors (the r_{xi}'s) and a matrix of coefficients for the criterion (rx_iy):

$$(\beta_1 \beta_2) \begin{pmatrix} r_{x1x1} & r_{x1x2} \\ r_{x1x2} & r_{x2x2} \end{pmatrix} = (r_{x1y} \quad r_{x2x2}) \tag{7}$$

If we let $\beta = (\beta_1 \beta_2)$, R = $\begin{pmatrix} r_{x1x1} & r_{x1x2} \\ r_{x1x2} & r_{x2x2} \end{pmatrix}$ and $r_{xy} = (r_{x1y} \quad r_{x2x2})$ then equation (7) becomes

$$\beta R = r_{xy} \tag{8}$$

and we can solve (8) for β by multiplying both sides by the inverse of R.

$$\beta = \beta R R^{-1} = r_{xy} R^{-1}$$

This works for any number of variables! But, it requires R^{-1}.

Matrix Inversion

The *inverse* of a square matrix is the matrix equivalent of dividing by that matrix. That is, either pre or post multiplying a matrix by its inverse yields the identity matrix. The inverse is particularly important in multiple regression, for it allows us to solve for the beta weights.
Given the equation

$$\hat{\boldsymbol{y}} = \boldsymbol{bX} + c$$

we can solve for $\boldsymbol{b}$ by multiplying both sides of the equation by $\boldsymbol{X}$' to form a square matrix $\boldsymbol{XX'}$ and then take the inverse of that square matrix:

$$\boldsymbol{yX'} = \boldsymbol{bXX'} <=> \boldsymbol{b} = \boldsymbol{yX'}(\boldsymbol{XX'})^{-1}$$

Matrix operations for finding an inverse

But, how do we find the inverse (R^{-1})? As an example we solve the inverse of a 2 x2 matrix, but the technique may be applied to a matrix of any size. First, define the identity matrix, I, as

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

and then the equation

$$R = IR$$

may be represented as

$$\begin{pmatrix} r_{x1x1} & r_{x1x2} \\ r_{x1x2} & r_{x2x2} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} r_{x1x1} & r_{x1x2} \\ r_{x1x2} & r_{x2x2} \end{pmatrix}$$

Transform both sides of the equation

Dropping the x subscript (for notational simplicity) we have

$$\begin{pmatrix} r_{11} & r_{12} \\ r_{12} & r_{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} \\ r_{12} & r_{22} \end{pmatrix} \tag{9}$$

We may multiply both sides of equation (9) by a simple transformation matrix (T) without changing the equality. If we do this repeatedly until the left hand side of equation (9) is the identity matrix, then the first matrix on the right hand side will be the inverse of R. We do this in several steps to show the process. Let

$$T_1 = \begin{pmatrix} \frac{1}{r_{11}} & 0 \\ 0 & \frac{1}{r_{22}} \end{pmatrix}$$

then we multiply both sides of equation (9) by T_1 in order to make the diagonal elements of the left hand equation = 1 and we have

$$T_1 R = T_1 IR \tag{10}$$

Keep transforming

$$\begin{pmatrix} 1 & \frac{r_{12}}{r_{11}} \\ \frac{r_{12}}{r_{22}} & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{r_{11}} & 0 \\ 0 & \frac{1}{r_{22}} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} \\ r_{12} & r_{22} \end{pmatrix} \tag{11}$$

Then, by letting

$$T_2 = \begin{pmatrix} 1 & 0 \\ -\frac{r_{12}}{r_{22}} & 1 \end{pmatrix}$$

and multiplying T_2 times both sides of equation (11) we can make the lower off diagonal element = 0. (Functionally, we are subtracting $\frac{r_{12}}{r_{22}}$ times the first row from the second row).

$$\begin{pmatrix} 1 & \frac{r_{12}}{r_{11}} \\ 0 & 1 - \frac{r_{12}^2}{r_{11} r_{22}} \end{pmatrix} = \begin{pmatrix} 1 & \frac{r_{12}}{r_{11}} \\ 0 & \frac{r_{11} r_{22} - r_{12}^2}{r_{11} r_{22}} \end{pmatrix} = \begin{pmatrix} \frac{1}{r_{11}} & 0 \\ -\frac{r_{12}}{r_{11} r_{22}} & \frac{1}{r_{22}} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} \\ r_{12} & r_{22} \end{pmatrix} \tag{12}$$

Continue to diagonalize

Then, in order to make the diagonal elements all = 1 , we let

$$T_3 = \begin{pmatrix} 1 & 0 \\ 0 & \frac{r_{11} r_{22}}{r_{11} r_{22} - r_{12}^2} \end{pmatrix}$$

and multiplying T_3 times both sides of equation (12) we have

$$\begin{pmatrix} 1 & \frac{r_{12}}{r_{11}} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{r_{11}} & 0 \\ -\frac{r_{12}}{r_{11} r_{22} - r_{12}^2} & \frac{r_{11}}{r_{11} r_{22} - r_{12}^2} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} \\ r_{12} & r_{22} \end{pmatrix} \qquad (13)$$

Then, to make the upper off diagonal element = 0, we let

$$T_4 = \begin{pmatrix} 1 & -\frac{r_{12}}{r_{11}} \\ 0 & 1 \end{pmatrix}$$

The inverse by successive transformations

and multiplying T_4 times both sides of equation (13) we have

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{r_{22}}{r_{11}r_{22}-r_{12}^2} & -\frac{r_{12}}{r_{11}r_{22}-r_{12}^2} \\ -\frac{r_{12}}{r_{11}r_{22}-r_{12}^2} & \frac{r_{11}}{r_{11}r_{22}-r_{12}^2} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} \\ r_{12} & r_{22} \end{pmatrix}$$

That is, the inverse of our original matrix, R, is

$$R^{-1} = \begin{pmatrix} \frac{r_{22}}{r_{11}r_{22}-r_{12}^2} & -\frac{r_{12}}{r_{11}r_{22}-r_{12}^2} \\ -\frac{r_{12}}{r_{11}r_{22}-r_{12}^2} & \frac{r_{11}}{r_{11}r_{22}-r_{12}^2} \end{pmatrix} \tag{14}$$

Finding the inverse as a series of transformations

The previous example was drawn out to be easier to follow, and it would be possible to combine several steps together. The important point is that by successively multiplying equation 9 by a series of transformation matrices, we have found the inverse of the original matrix.

$$T_4 T_3 T_2 T_1 R = T_4 T_3 T_2 T_1 IR$$

or, in other words

$$T_4 T_3 T_2 T_1 R = I = R^{-1} R$$

$$T_4 T_3 T_2 T_1 I = R^{-1} \tag{15}$$

Empirical examples of the inverse – use `solve`

Original matrix

```
> a
     [,1] [,2]
[1,]  1.0  0.5
[2,]  0.5  1.0
> b
     [,1] [,2]
[1,]  1.0  0.8
[2,]  0.8  1.0
> c
     [,1] [,2]
[1,]  1.0  0.9
[2,]  0.9  1.0

> B
     [,1] [,2] [,3]
[1,]  1.0  0.0  0.5
[2,]  0.0  1.0  0.3
[3,]  0.5  0.3  1.0
```

Inverse of Matrix

```
> round(solve(a),2)
      [,1]  [,2]
[1,]  1.33 -0.67
[2,] -0.67  1.33
> round(solve(b),2)
      [,1]  [,2]
[1,]  2.78 -2.22
[2,] -2.22  2.78
> round(solve(c),2)
      [,1]  [,2]
[1,]  5.26 -4.74
[2,] -4.74  5.26

> round(solve(B),2)
      [,1]  [,2]  [,3]
[1,]  1.38  0.23 -0.76
[2,]  0.23  1.14 -0.45
[3,] -0.76 -0.45  1.52
```

```
>C
     [,1] [,2] [,3]
[1,]  1.0  0.8  0.5
[2,]  0.8  1.0  0.3
[3,]  0.5  0.3  1.0
> D
     [,1] [,2] [,3]
[1,]  1.0  0.9  0.5
[2,]  0.9  1.0  0.3
[3,]  0.5  0.3  1.0
> E
     [,1] [,2] [,3]
[1,] 1.00 0.95  0.5
[2,] 0.95 1.00  0.3
[3,] 0.50 0.30  1.0
> F
     [,1] [,2] [,3]
[1,] 1.00 0.99  0.5
[2,] 0.99 1.00  0.3
[3,] 0.50 0.30  1.0
```

```
> round(solve(C),2)
     [,1]  [,2]  [,3]
[1,]  3.5 -2.50 -1.00
[2,] -2.5  2.88  0.38
[3,] -1.0  0.38  1.38
> round(solve(D),2)
      [,1]  [,2]  [,3]
[1,]  7.58 -6.25 -1.92
[2,] -6.25  6.25  1.25
[3,] -1.92  1.25  1.58
> round(solve(E),2)
       [,1]   [,2]  [,3]
[1,]  21.41 -18.82 -5.06
[2,] -18.82  17.65  4.12
[3,]  -5.06   4.12  2.29
> round(solve(F),2)
       [,1]   [,2]  [,3]
[1,] -39.39  36.36  8.79
[2,]  36.36 -32.47 -8.44
[3,]   8.79  -8.44 -0.86
```

As the correlations become bigger, the inverse becomes numerically less stable, and eventually not positive semidefinite.

Instability of the inverse with high values of correlations

The problem of collinearity arises when the inverse becomes unstable. As we shall see, this is when the matrix has 0 or negative eigenvalues. Consider what happens if one correlation changes in the 5th decimal place:

```
> F
       [,1]   [,2] [,3]
[1,] 1.0000 0.9761  0.5
[2,] 0.9761 1.0000  0.3
[3,] 0.5000 0.3000  1.0

> F2
        [,1]    [,2] [,3]
[1,] 1.00000 0.97613  0.5
[2,] 0.97613 1.00000  0.3
[3,] 0.50000 0.30000  1.0


> F3
        [,1]    [,2] [,3]
[1,] 1.00000 0.97615  0.5
[2,] 0.97615 1.00000  0.3
[3,] 0.50000 0.30000  1.0
```

```
> solve(F)
           [,1]        [,2]        [,3]
[1,]  15478.823 -14051.709 -3523.8986
[2,] -14051.709  12757.272  3198.6732
[3,]  -3523.899   3198.673   803.3473

> solve(F2)
          [,1]      [,2]       [,3]
[1,]  98665.31 -89571.84 -22461.103
[2,] -89571.84  81317.56  20390.650
[3,] -22461.10  20390.65   5114.357

> solve(F3)
          [,1]       [,2]      [,3]
[1,] -38199.18  34679.400  8695.771
[2,]  34679.40 -31482.842 -7894.847
[3,]   8695.77  -7894.847 -1978.431
```

Collinearity is not just very high correlations

It is when one variable is a linear sum of other variables. Examine what happens when the $x_{1,3}$ correlation changes.

```
> F1
     [,1] [,2] [,3]
[1,] 1.00  0.9 0.43
[2,] 0.90  1.0 0.00
[3,] 0.43  0.0 1.00
> F2
      [,1] [,2]  [,3]
[1,] 1.000  0.9 0.435
[2,] 0.900  1.0 0.000
[3,] 0.435  0.0 1.000
> F3
     [,1] [,2] [,3]
[1,] 1.00  0.9 0.44
[2,] 0.90  1.0 0.00
[3,] 0.44  0.0 1.00
```

```
> solve(F1)
           [,1]       [,2]      [,3]
[1,]  196.07843 -176.47059 -84.31373
[2,] -176.47059  159.82353  75.88235
[3,]  -84.31373   75.88235  37.25490
> solve(F2)
           [,1]       [,2]      [,3]
[1,]  1290.3226 -1161.2903 -561.2903
[2,] -1161.2903  1046.1613  505.1613
[3,]  -561.2903   505.1613  245.1613
> solve(F3)
           [,1] [,2]       [,3]
[1,] -277.7778  250  122.22222
[2,]  250.0000 -224 -110.00000
[3,]  122.2222 -110  -52.77778
```

Colinearity and computational singularity

```
> F1
     [,1] [,2] [,3]
[1,] 1.00  0.8 0.59
[2,] 0.80  1.0 0.00
[3,] 0.59  0.0 1.00
> F2
      [,1] [,2]  [,3]
[1,] 1.000  0.8 0.599
[2,] 0.800  1.0 0.000
[3,] 0.599  0.0 1.000
> F3
      [,1] [,2]  [,3]
[1,] 1.000  0.8 0.601
[2,] 0.800  1.0 0.000
[3,] 0.601  0.0 1.000
> F
     [,1] [,2] [,3]
[1,]  1.0  0.8  0.6
[2,]  0.8  1.0  0.0
[3,]  0.6  0.0  1.0
```

```
> solve(F1)
           [,1]      [,2]      [,3]
[1,]  84.03361 -67.22689 -49.57983
[2,] -67.22689  54.78151  39.66387
[3,] -49.57983  39.66387  30.25210
> solve(F2)
           [,1]      [,2]      [,3]
[1,]  834.0284 -667.2227 -499.5830
[2,] -667.2227  534.7781  399.6664
[3,] -499.5830  399.6664  300.2502
> solve(F2)
           [,1]      [,2]      [,3]
[1,] -832.6395  666.1116  500.4163
[2,]  666.1116 -531.8893 -400.3331
[3,]  500.4163 -400.3331 -299.7502

> solve(F)
Error in solve.default(F) :
  system is computationally singular:
  reciprocal condition number = 9.25186e
```

Eigen Values and Eigen Vectors

The *eigenvectors* of a matrix are said to provide a *basis space* for the matrix. This is a set of orthogonal vectors which when multiplied by the appropriate scaling vector of *eigenvalues* will reproduce the matrix.
Given a $n * n$ matrix $\boldsymbol{R}$, each eigenvector solves the equation

$$\boldsymbol{x_i R} = \lambda_i \boldsymbol{x_i}$$

and the set of n eigenvectors are solutions to the equation

$$\boldsymbol{XR} = \boldsymbol{\lambda X}$$

where $\boldsymbol{X}$ is a matrix of orthogonal eigenvectors and $\boldsymbol{\lambda}$ is a diagonal matrix of the the eigenvalues, λ_i. Then

$$\boldsymbol{x_i R} - \lambda_i \boldsymbol{XI} = 0 <=> \boldsymbol{x_i}(\boldsymbol{R} - \lambda_i \boldsymbol{I}) = 0$$

Finding eigen values

Finding the eigenvectors and values is computationally tedious, but may be done using the `eigen` function which uses a ***QR*** *decomposition* of the matrix. That the vectors making up $\boldsymbol{X}$ are orthogonal means that

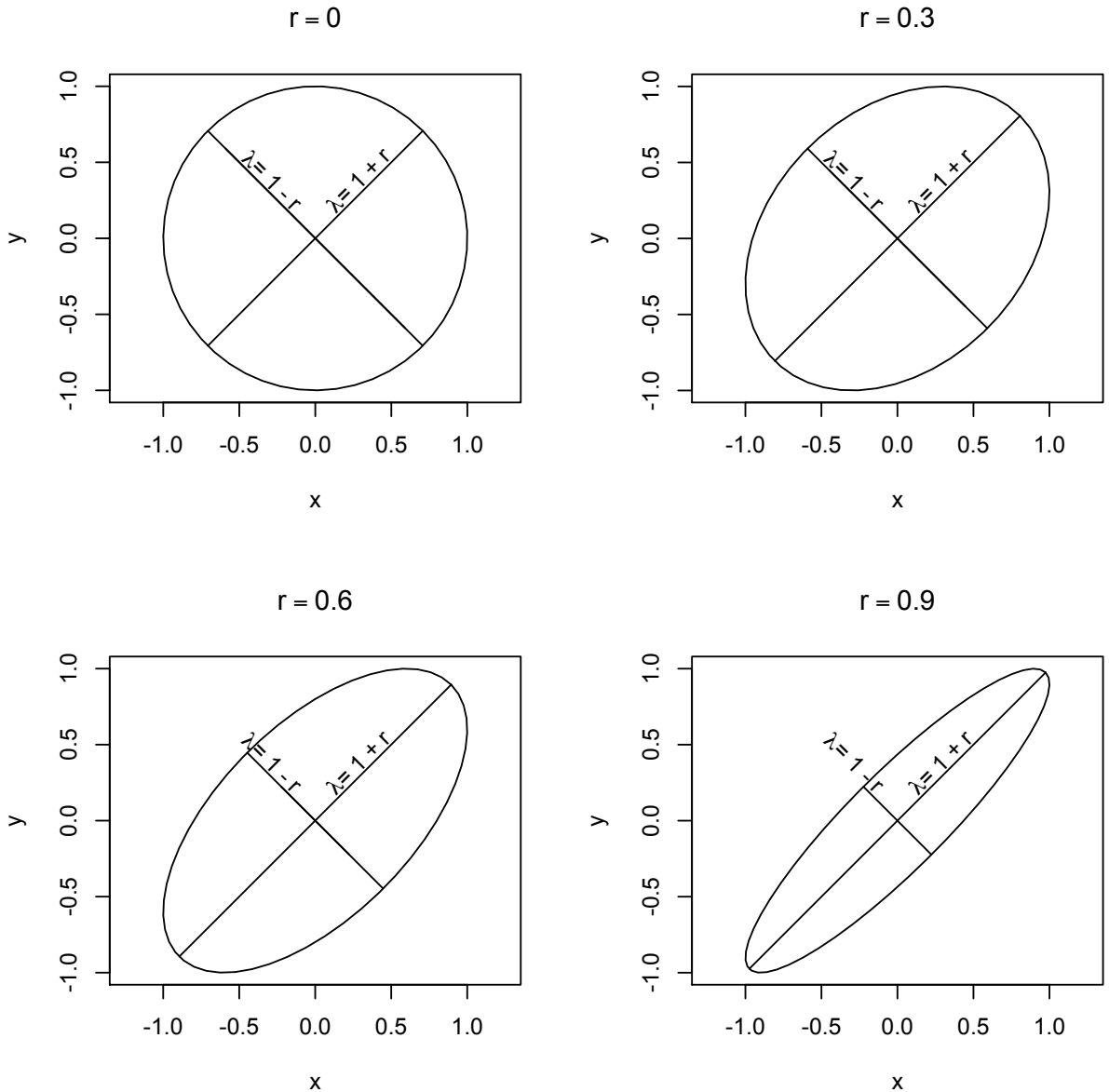
$$\boldsymbol{X}\boldsymbol{X}' = \boldsymbol{I}$$

and because they form the *basis space* for $\boldsymbol{R}$ that

$$\boldsymbol{R} = \boldsymbol{X}\lambda\boldsymbol{X}'.$$

That is, it is possible to recreate the correlation matrix $\boldsymbol{R}$ in terms of an orthogonal set of vectors (the *eigenvectors*) scaled by their associated *eigenvalues*.

Eigen vectors of a 2 x 2 correlation matrix



Although the length (eigen values) of the axes differ, their orientation (eigen vectors) are the same.

```
> r2 <- matrix(c(1,.6,.6,1),2,2
> print(eigen(r2),2)

$values
[1] 1.6 0.4

$vectors
     [,1]  [,2]
[1,] 0.71 -0.71
[2,] 0.71  0.71
```

Eigenvalue decomposition and matrix inverses

1. A correlation matrix can be recreated by its (orthogonal) eigenvectors and eigen values
   - $\boldsymbol{R} = \boldsymbol{X}\lambda\boldsymbol{X'}$ where
   - $\boldsymbol{XX'} = \boldsymbol{I} = \boldsymbol{X'X}$ the eigenvectors are orthogonal.
2. The inverse of a matrix $\boldsymbol{R}^{-1}$ is that matrix which when multiplied by $\boldsymbol{R}$ is the Identify matrix $\boldsymbol{I}$.
   - $\boldsymbol{R}\boldsymbol{R}^{-1} = \boldsymbol{R}^{-1}\boldsymbol{R} = \boldsymbol{I}$
3. Combine these two concepts and we see that the inverse is $\boldsymbol{X}(1/\lambda)\boldsymbol{X'}$ since
   - $\boldsymbol{R}\boldsymbol{R}^{-1} = (\boldsymbol{X}\lambda\boldsymbol{X'})(\boldsymbol{X}(1/\lambda)\boldsymbol{X'}) = (\boldsymbol{X}\lambda)(\boldsymbol{X'X})(1/\lambda)\boldsymbol{X'})$
   - $(\boldsymbol{X}\lambda)\boldsymbol{I}(1/\lambda)\boldsymbol{X'}) = \boldsymbol{X}(\lambda\boldsymbol{I}(1/\lambda)\boldsymbol{X'} = \boldsymbol{XIX'} = \boldsymbol{I}$
4. Thus, the problem of a non-semidefinite matrix is really a problem of 0 or negative eigen values.

Using eigen values and eigen vectors to smooth a matrix

Consider the `burt` correlation matrix:

```
lowerMat(burt)
            Sclty Sorrw Tndrn Joy   Wondr Elatn Dsgst Anger Sex   Fear  Sbjct
Sociality    1.00
Sorrow       0.83  1.00
Tenderness   0.81  0.87  1.00
Joy          0.80  0.62  0.63  1.00
Wonder       0.71  0.59  0.37  0.49  1.00
Elation      0.70  0.44  0.31  0.54  0.54  1.00
Disgust      0.54  0.58  0.30  0.30  0.34  0.50  1.00
Anger        0.53  0.44  0.12  0.28  0.55  0.51  0.38  1.00
Sex          0.59  0.23  0.33  0.42  0.40  0.31  0.29  0.53  1.00
Fear         0.24  0.45  0.33  0.29  0.19  0.11  0.21  0.10 -0.09  1.00
Subjection   0.13  0.21  0.36 -0.06 -0.10  0.10  0.08 -0.16 -0.10  0.41  1.00

Unfortunately, one eigen value is negative:
round( eigen(burt)$values,2)
 [1]  5.17  1.79  0.97  0.78  0.69  0.62  0.51  0.35  0.13  0.01 -0.02
```

1. Because the matrix is not positive, semi-definite (it has a negative determinant – see below –and has negative eigen values) we can not apply normal multivariate techniques.
2. We can smooth the matrix by finding its eigen value/vector decomposition, and adjusting the eigen values to be all positive.

$$\boldsymbol{R} = \boldsymbol{X}' \lambda \boldsymbol{X}$$

3. We use `cor.smooth` and then compare the results by using the `lowerUpper` function.

Using eigen values and eigen vectors to smooth a matrix

R code

```
smoothed <- cor.smooth(burt)
lu <- lowerUpper(lower=burt, upper=smoothed,diff=TRUE)
df2latex(lu)
```

Table: The original (lower off diagonal) and difference from smoothed (upper off diagonal) matrix

Variable	Sclty	Sorrw	Tndrn	Joy	Wondr	Elatn	Dsgst	Anger	Sex	Fear	Sbjct
Sociality		0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Sorrow	0.83		0.02	0.00	0.01	0.00	0.01	0.01	0.00	0.00	0.00
Tenderness	0.81	0.87		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Joy	0.80	0.62	0.63		0.00	0.00	0.00	0.00	0.00	0.00	0.00
Wonder	0.71	0.59	0.37	0.49		0.00	0.00	0.00	0.00	0.00	0.00
Elation	0.70	0.44	0.31	0.54	0.54		0.00	0.00	0.00	0.00	0.00
Disgust	0.54	0.58	0.30	0.30	0.34	0.50		0.00	0.00	0.00	0.00
Anger	0.53	0.44	0.12	0.28	0.55	0.51	0.38		0.00	0.00	0.00
Sex	0.59	0.23	0.33	0.42	0.40	0.31	0.29	0.53		0.00	0.00
Fear	0.24	0.45	0.33	0.29	0.19	0.11	0.21	0.10	-0.09		0.00
Subjection	0.13	0.21	0.36	-0.06	-0.10	0.10	0.08	-0.16	-0.10	0.41	

Determinants

- The *determinant* of an n * n correlation matrix may be thought of as the proportion of the possible n-space spanned by the variable space and is sometimes called the *generalized variance* of the matrix. As such, it can also be considered as the volume of the variable space.
- If the correlation matrix is thought of a representing vectors within a n dimensional space, then the eigenvalues are the lengths of the axes of that space. The product of these, the determinant, is then the volume of the space.
- It will be a maximum when the axes are all of unit length and be zero if at least one axis is zero.
  - Think of a three dimensional sphere (and then generalize to a n dimensional hypersphere.)
  - If it is squashed in a way that preserves the sum of the lengths of the axes, then volume of the *oblate hyper sphere* will be reduced.

Determinants and redundancy

The determinant is an inverse measure of the redundancy of the matrix. The smaller the determinant, the more variables in the matrix are measuring the same thing (are correlated). The determinant of the identity matrix is 1, the determinant of a matrix with at least two perfectly correlated (linearly dependent) rows or columns will be 0. If the matrix is transformed into a lower diagonal matrix, the determinant is the product of the diagonals. The determinant of a n * n square matrix, $\boldsymbol{R}$ is also the product of the n *eigenvalues* of that matrix.

$$det(\boldsymbol{R}) = \|\boldsymbol{R}\| = \Pi_{i=1}^{n} \lambda_i \tag{16}$$

and the *characteristic equation* for a square matrix, $\boldsymbol{X}$, is

$$\|\boldsymbol{X} - \lambda \boldsymbol{I}\| = 0$$

where λ_i is an eigenvalue of $\boldsymbol{X}$.

Finding and using the determinant

1. The determinant may be found by the `det` function.
2. A negative determinant implies the matrix is not positive semi-definite. It will have negative eigen values.
3. A determinant of 0 means the matrix is not invertible
4. The determinant may be used in estimating the goodness of fit of a particular model ($\boldsymbol{\Sigma}$) to the data ($\boldsymbol{S}$)
   - for when the model fits perfectly, then the inverse of the model times the data ($\boldsymbol{\Sigma}^{-1}\boldsymbol{S}$) will be an identity matrix and the determinant (det($\boldsymbol{\Sigma}^{-1}\boldsymbol{S}$) will be 1.
   - A poor model fit will have a determinant much less than 1.
   - Remember, that the determinant is just the product of the eigen values

Multiple R and matrix multiplication

$$\hat{\boldsymbol{y}} = \boldsymbol{bX} + c$$

Because we can not divide by a matrix, nor take the inverse of a non-square matrix, we can solve for $\boldsymbol{b}$ by multiplying both sides of the equation by $\boldsymbol{X}$' to form a square matrix $\boldsymbol{XX'}$ (essentially the covariance matrix) and then take the inverse of that square matrix:

$$\boldsymbol{yX'} = \boldsymbol{bXX'} <=> \boldsymbol{b} = \boldsymbol{yX'}(\boldsymbol{XX'})^{-1}$$

1. The elements of the diagonal of the inverse are the reciprocals of the amount of unique variance in each variable.
2. Thus, the squared multiple correlation of each variable with each of the other variables is known as the SMC and is

$$\boldsymbol{smc} = 1 - 1/diag(\boldsymbol{R}^{-1})$$

3. The partial correlations when all other variables are removed are the negative values of the inverse of the correlation matrix divided by the diagonal of the inverse.

Consider the following matrix

```
R
     V1   V2   V3   V4
V1 1.00 0.56 0.48 0.40
V2 0.56 1.00 0.42 0.35
V3 0.48 0.42 1.00 0.30
V4 0.40 0.35 0.30 1.00

round(solve(R),2)
      V1    V2    V3    V4
V1  1.71 -0.66 -0.45 -0.32
V2 -0.66  1.55 -0.28 -0.20
V3 -0.45 -0.28  1.37 -0.13
V4 -0.32 -0.20 -0.13  1.24

> round(cov2cor(solve(R)),2)
      V1    V2    V3    V4
V1  1.00 -0.40 -0.29 -0.22
V2 -0.40  1.00 -0.19 -0.14
V3 -0.29 -0.19  1.00 -0.10
V4 -0.22 -0.14 -0.10  1.00
```

```
The determinant is
det(R)
[1] 0.40842
```

```
The inverse is

The diag of the inverse
is the reciprocal of the
unexplained variance
 round(1/diag(solve(R)),2)
  V1   V2   V3   V4
0.58 0.64 0.73 0.81
```

```
 smc(R)
       V1        V2        V3        V4
0.4159588 0.3566163 0.2715891 0.1918748

round(partial.r(R),2)
     V1   V2   V3   V4
V1 1.00 0.40 0.29 0.22
V2 0.40 1.00 0.19 0.14
V3 0.29 0.19 1.00 0.10
V4 0.22 0.14 0.10 1.00
```

```
The Squared Multiple Correlation  (SMC)
1 - 1/diag(solve(R))
 V1   V2   V3   V4
0.42 0.36 0.27 0.19
```

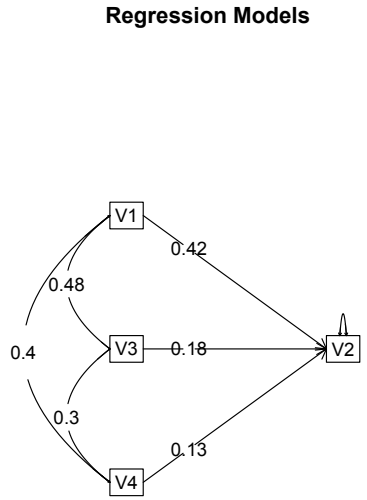
```
The determinant of the partial R
det(partial.r(R))
[1] 0.719825
```

Multple R from correlation matrix input using setCor

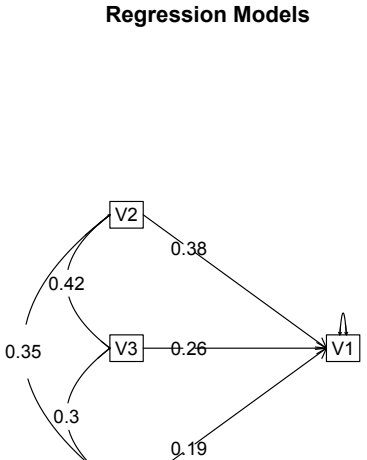
R code

```
setCor(V1 ~ V2 + V3 + V4,data=R)
setCor(V2 ~ V1 + V3 + V4, data =R)
round(smc(R),2)
```

```
Call: setCor(y = V1 ~ V2 + V3 + V4, data = R)
Multiple Regression from matrix input

 DV =  V1   slope  VIF
V2  0.38 1.30
V3  0.26 1.25
V4  0.19 1.18

 Multiple Regression
      R   R2  Ruw R2uw
V1 0.64 0.42 0.64  0.4
```




```
Call: setCor(y = V2 ~ V1 + V3 + V4, data = R)
Multiple Regression from matrix input

 DV =  V2   slope  VIF
V1  0.42 1.43
V3  0.18 1.32
V4  0.13 1.21

 Multiple Regression
     R   R2  Ruw R2uw
V2 0.6 0.36 0.57 0.33
```



R code

```
R
round(partial.r(R),2)
partial.r(R,1:2,3:4)
partial.r(R,c(1,3),c(2,4))
partial.r(R,c(1,4),c(2,3))
```

```
R
     V1   V2   V3   V4
V1 1.00 0.56 0.48 0.40
V2 0.56 1.00 0.42 0.35
V3 0.48 0.42 1.00 0.30
V4 0.40 0.35 0.30 1.00
> round(partial.r(R),2)
     V1   V2   V3   V4
V1 1.00 0.40 0.29 0.22
V2 0.40 1.00 0.19 0.14
V3 0.29 0.19 1.00 0.10
V4 0.22 0.14 0.10 1.00
> partial.r(R,1:2,3:4)
partial correlations
    V1  V2
V1 1.0 0.4
V2 0.4 1.0
> partial.r(R,c(1,3),c(2,4))
partial correlations
     V1   V3
V1 1.00 0.29
V3 0.29 1.00
> partial.r(R,c(1,4),c(2,3))
partial correlations
     V1   V4
V1 1.00 0.22
V4 0.22 1.00
```

Using the partial.r function

The original correlation matrix

The partial correlations (everything from everything)

Partial of 1 and 2 removing 3 and 4

Partial of 1 and 3 removing 2 and 4

Partial of 1 and 4 removing 2 and 3

Multiple correlation and partial R

R code

```
setCor(V1 ~ V2 + V3 + V4, data=R)
setCor(V1 ~ V2 + V3 - V4, data=R)
setCor( V1 ~ V2 - V3 - V4, data = R)
```

```
setCor(V1 ~ V2 + V3 + V4, data=R)
Call: setCor(y = V1 ~ V2 + V3 + V4, data = R)
Multiple Regression from matrix input
 DV =  V1   slope  VIF
V2  0.38 1.30                                  The complete regression of 1 on 2,3,4
V3  0.26 1.25
V4  0.19 1.18
 Multiple Regression
      R   R2  Ruw R2uw
V1 0.64 0.42 0.64  0.4
> setCor(V1 ~ V2 + V3 - V4, data=R)
Call: setCor(y = V1 ~ V2 + V3 - V4, data = R)   Regression of 1 on 2, 3 removing 4
Multiple Regression from matrix input
 DV =  V1   slope  VIF
V2  0.38 1.14
V3  0.26 1.14
 Multiple Regression
      R  R2 Ruw R2uw
V1 0.55 0.3 0.5 0.25                           Regression of 1 on 2, removing 3 and 4
> setCor(V1 ~ V2 - V3 - V4, data=R)
Call: setCor(y = V1 ~ V2 - V3 - V4, data = R)
Multiple Regression from matrix input
 DV =  V1   slope VIF
V2  0.38   1
 Multiple Regression
     R   R2  Ruw R2uw
V1 0.4 0.16 0.34 0.11
```