Appendix–1
Matrix Algebra in R

William Revelle
Northwestern University

January 2, 2007

Prepared as part of a course on Latent Variable Modeling, Winter, 2007
and as a supplement to the Guide to R for psychologists.
email comments to: revelle@northwestern.edu

Much of psychometrics in particular, and psychological data analysis in general consists of operations on vectors and matrices. This appendix offers a quick review of matrix operations with a particular emphasis upon how to do matrix operations in R. For more information on how to use R, consult the short guide to R for psychologists (at http://personality-project.org/r/r.guide.html) or the even shorter guide

1 Vectors

A vector is a one dimensional array of n numbers. Basic operations on a vector are addition and subtraction. Multiplication is somewhat more complicated, for the order in which two vectors are multiplied changes the result. That is $AB \neq BA$.

Consider V1 = the first 10 integers, and V2 = the next 10 integers:

```
> V1 <- as.vector(seq(1, 10))

 [1]  1  2  3  4  5  6  7  8  9 10

> V2 <- as.vector(seq(11, 20))

 [1] 11 12 13 14 15 16 17 18 19 20
```

We can add a constant to each element in a vector

```
> V4 <- V1 + 20

 [1] 21 22 23 24 25 26 27 28 29 30
```

or we can add each element of the first vector to the corresponding element of the second vector

```
> V3 <- V1 + V2

 [1] 12 14 16 18 20 22 24 26 28 30
```

Strangely enough, a vector in R is dimensionless, but it has a length. If we want to multiply two vectors, we first need to think of the vector either as row or as a column. A column vector can be made into a row vector (and vice versa) by the transpose operation. While a vector has no dimensions, the transpose of a vector is two dimensional! It is a matrix with with 1 row and n columns. (Although the dim command will return no dimensions, in terms of multiplication, a vector is a matrix of n rows and 1 column.)

Consider the following:

```
> dim(V1)

NULL

> length(V1)

[1] 10

> dim(t(V1))

[1]  1 10

> dim(t(t(V1)))

[1] 10  1

> TV <- t(V1)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    2    3    4    5    6    7    8    9    10
```

```
> t(TV)
```

```
      [,1]
 [1,]    1
 [2,]    2
 [3,]    3
 [4,]    4
 [5,]    5
 [6,]    6
 [7,]    7
 [8,]    8
 [9,]    9
[10,]   10
```

1.1 vector multiplication

Just as we can add a number to every element in a vector, so can we multiply a number (a "scaler") by every element in a vector.

```
> V2 <- 4 * V1
```

```
 [1]  4  8 12 16 20 24 28 32 36 40
```

To multiply two vectors, we can find either the "inner" or the "outer" product. If we multiple a row vector by a column vector, we find the "inner-product" which will be the sum of the products of the corresponding elements: $\sum_{i=1}^{N} V1_i * V2_i =$

```
> t(V1)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    2    3    4    5    6    7    8    9    10
```

```
> V2
```

```
 [1]  4  8 12 16 20 24 28 32 36 40
```

```
> in.prod <- t(V1) %*% V2
```

```
     [,1]
[1,] 1540
```

Note that the inner product of two vectors is of length =1 but is a matrix with 1 row and 1 column.

But a column vector by a row vector produces the "outer product" which will be a matrix where each element is the product of the corresponding rows and columns of the two vectors. (Known to most school children as a multiplication table.)

```
> in.prod <- V1 %*% t(V2)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    4    8   12   16   20   24   28   32   36    40
 [2,]    8   16   24   32   40   48   56   64   72    80
 [3,]   12   24   36   48   60   72   84   96  108   120
 [4,]   16   32   48   64   80   96  112  128  144   160
 [5,]   20   40   60   80  100  120  140  160  180   200
 [6,]   24   48   72   96  120  144  168  192  216   240
 [7,]   28   56   84  112  140  168  196  224  252   280
 [8,]   32   64   96  128  160  192  224  256  288   320
 [9,]   36   72  108  144  180  216  252  288  324   360
[10,]   40   80  120  160  200  240  280  320  360   400
```

1.2 Simple statistics using vectors

Although there are built in functions in R to do most of our statistics, it is useful to understand how these operations can be done using vector and matrix operations. Here we consider how to find the mean of a vector, remove it from all the numbers, and then find the average squared deviation from the mean (the variance).

Consider the mean of all numbers in a vector. To find this we just need to add up the numbers (the inner product of the vector with a vector of 1's) and then divide by n (multiply by the scaler 1/n). First we create a vector of 1s by using the repeat operation.

```
> V <- V1

 [1]  1  2  3  4  5  6  7  8  9 10

> one <- rep(1, length(V))

 [1] 1 1 1 1 1 1 1 1 1 1

> sum.V <- t(one) %*% V

     [,1]
[1,]   55

> mean.V <- sum.V * (1/length(V))

     [,1]
[1,]  5.5

> mean.V <- t(one) %*% V * (1/length(V))

     [,1]
[1,]  5.5

> mean.V <- t(one) %*% V/length(V)

     [,1]
[1,]  5.5
```

The variance is the average squared deviation from the mean. To find the variance, we first find deviation scores by subtracting them mean from each value of the vector. Then, to find the sum of the squared deviations take the inner product of the result. This Sum of Squares becomes a variance if we divide by the degrees of freedom (n-1) to get an unbiased estimate of the population variance). First we find the mean centered vector:

```
> V - mean.V

 [1] -4.5 -3.5 -2.5 -1.5 -0.5  0.5  1.5  2.5  3.5  4.5
```

And then we find the variance as the mean square by taking the inner product:

```
> Var.V <- t(V - mean.V) %*% (V - mean.V) * (1/(length(V) - 1))

         [,1]
[1,] 9.166667
```

Compare these results with the more typical scale, mean and var operations:

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

```
      [,1]
 [1,] -4.5
 [2,] -3.5
 [3,] -2.5
 [4,] -1.5
 [5,] -0.5
 [6,]  0.5
 [7,]  1.5
 [8,]  2.5
 [9,]  3.5
[10,]  4.5
attr(,"scaled:center")
[1] 5.5
```

```
> mean(V)

[1] 5.5

> var(V)

[1] 9.166667
```

1.3 Combining vectors

We can form more complex data structures than vectors by combining the vectors, either by columns or by rows. The resulting data structure is a matrix.

```
> Xc <- cbind(V1, V2, V3)

      V1 V2 V3
 [1,]  1  4 12
 [2,]  2  8 14
 [3,]  3 12 16
 [4,]  4 16 18
 [5,]  5 20 20
 [6,]  6 24 22
 [7,]  7 28 24
 [8,]  8 32 26
 [9,]  9 36 28
[10,] 10 40 30

> Xr <- cbind(V1, V2, V3)

      V1 V2 V3
 [1,]  1  4 12
 [2,]  2  8 14
 [3,]  3 12 16
 [4,]  4 16 18
 [5,]  5 20 20
 [6,]  6 24 22
 [7,]  7 28 24
 [8,]  8 32 26
 [9,]  9 36 28
[10,] 10 40 30

> dim(Xc)
```

```
[1] 10  3
```

```
> dim(Xr)
```

```
[1] 10  3
```

2 Matrices

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 columns representing different variables and rows containing the responses of a particular subject. Thus, a 10 x 4 data matrix (10 rows, 4 columns) would contain the data of 10 subjects on 4 different variables. Note that the matrix operation has taken the numbers 1 through 40 and organized them column wise. That is, a matrix is just a way (and a very convenient one at that) of organizing a vector.

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.

```
> Xij <- matrix(seq(1:40), ncol = 4)
```

```
      [,1] [,2] [,3] [,4]
 [1,]    1   11   21   31
 [2,]    2   12   22   32
 [3,]    3   13   23   33
 [4,]    4   14   24   34
 [5,]    5   15   25   35
 [6,]    6   16   26   36
 [7,]    7   17   27   37
 [8,]    8   18   28   38
 [9,]    9   19   29   39
[10,]   10   20   30   40
```

```
> rownames(Xij) <- paste("S", seq(1, dim(Xij)[1]), sep = "")
```

```
 [1] "S1"  "S2"  "S3"  "S4"  "S5"  "S6"  "S7"  "S8"  "S9"  "S10"
```

```
> colnames(Xij) <- paste("V", seq(1, dim(Xij)[2]), sep = "")
```

```
[1] "V1" "V2" "V3" "V4"
```

```
> Xij
```

```
    V1 V2 V3 V4
S1   1 11 21 31
S2   2 12 22 32
S3   3 13 23 33
S4   4 14 24 34
S5   5 15 25 35
S6   6 16 26 36
S7   7 17 27 37
S8   8 18 28 38
S9   9 19 29 39
S10 10 20 30 40
```

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. Note that now the subjects are columns and the variables are the rows.

```
> t(Xij)
```

```
   S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
V1  1  2  3  4  5  6  7  8  9  10
V2 11 12 13 14 15 16 17 18 19  20
V3 21 22 23 24 25 26 27 28 29  30
V4 31 32 33 34 35 36 37 38 39  40
```

2.1 Matrix addition

The previous matrix is rather uninteresting, in that all the columns are simple linear sums of the first column. A more typical matrix might be formed by sampling from the digits 0-9. For the purpose of this demonstration, we will set the random number seed to a memorable number so that it will yield the same answer each time.

```
> set.seed(42)
> Xij <- matrix(sample(seq(0, 9), 40, replace = TRUE), ncol = 4)
> rownames(Xij) <- paste("S", seq(1, dim(Xij)[1]), sep = "")
> colnames(Xij) <- paste("V", seq(1, dim(Xij)[2]), sep = "")
> print(Xij)
```

```
    V1 V2 V3 V4
S1   9  4  9  7
S2   9  7  1  8
S3   2  9  9  3
S4   8  2  9  6
S5   6  4  0  0
S6   5  9  5  8
S7   7  9  3  0
S8   1  1  9  2
S9   6  4  4  9
S10  7  5  8  6
```

Just as we could with vectors, we can add, subtract, muliply or divide the matrix by a scaler (a number with out a dimension).

```
> Xij + 4
```

```
    V1 V2 V3 V4
S1  13  8 13 11
S2  13 11  5 12
S3   6 13 13  7
S4  12  6 13 10
S5  10  8  4  4
S6   9 13  9 12
S7  11 13  7  4
S8   5  5 13  6
S9  10  8  8 13
S10 11  9 12 10
```

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

```
      V1   V2   V3   V4
S1  4.33 2.67 4.33 3.67
S2  4.33 3.67 1.67 4.00
S3  2.00 4.33 4.33 2.33
```

```
S4  4.00 2.00 4.33 3.33
S5  3.33 2.67 1.33 1.33
S6  3.00 4.33 3.00 4.00
S7  3.67 4.33 2.33 1.33
S8  1.67 1.67 4.33 2.00
S9  3.33 2.67 2.67 4.33
S10 3.67 3.00 4.00 3.33
```

We can also multiply each row (or column, depending upon order) by a vector.

```
> V

 [1]  1  2  3  4  5  6  7  8  9 10

> Xij * V

    V1 V2 V3 V4
S1   9  4  9  7
S2  18 14  2 16
S3   6 27 27  9
S4  32  8 36 24
S5  30 20  0  0
S6  30 54 30 48
S7  49 63 21  0
S8   8  8 72 16
S9  54 36 36 81
S10 70 50 80 60
```

2.2 Matrix multiplication

Matrix multiplication is a combination of multiplication and addition. Consider our matrix Xij with 10 rows of 4 columns. Call an individual element in this matrix xij. We can find the sums for each column of the matrix by multiplying the matrix by our "one" vector with Xij. That is, we can find $\sum_{i=1}^{N} X_i j$ for the j columns, and then divide by the number (n) of rows. (Note that we can get the same result by finding colMeans(Xij).

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] 10  4

> n <- dim(Xij)[1]

[1] 10

> X.means <- t(one) %*% Xij/n

     V1  V2  V3  V4
[1,]  6 5.4 5.7 4.9

> colMeans(Xij)

 V1  V2  V3  V4
6.0 5.4 5.7 4.9
```

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 matrix of means. To make them have the same dimensions we premultiply the means vector by a vector of ones.

```
> one %*% X.means
```

```
      V1  V2  V3  V4
 [1,]  6 5.4 5.7 4.9
 [2,]  6 5.4 5.7 4.9
 [3,]  6 5.4 5.7 4.9
 [4,]  6 5.4 5.7 4.9
 [5,]  6 5.4 5.7 4.9
 [6,]  6 5.4 5.7 4.9
 [7,]  6 5.4 5.7 4.9
 [8,]  6 5.4 5.7 4.9
 [9,]  6 5.4 5.7 4.9
[10,]  6 5.4 5.7 4.9
```

To find the variance/covariance matrix, we can first find the the inner product of the means centered matrix Xij - X.means t(Xij-X.means) with (Xij-X.means) and divide by n-1. We can compare this result to the result of the *c*ov function (the normal way to find covariances).

```
> X.cov <- t(Xij - one %*% X.means) %*% (Xij - one %*% X.means)/(n - 1)
> round(X.cov, 2)
```

```
      V1    V2    V3    V4
V1  7.33  0.11 -3.00  3.67
V2  0.11  8.71 -3.20 -0.18
V3 -3.00 -3.20 12.68  1.63
V4  3.67 -0.18  1.63 11.43
```

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

```
      V1    V2    V3    V4
V1  7.33  0.11 -3.00  3.67
V2  0.11  8.71 -3.20 -0.18
V3 -3.00 -3.20 12.68  1.63
V4  3.67 -0.18  1.63 11.43
```

2.3 Finding and using the diagonal

Some operations need to find just the diagonal. 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). Compare this to the standard command for finding correlations.

```
> round(X.cov, 2)
```

```
      V1    V2    V3    V4
V1  7.33  0.11 -3.00  3.67
V2  0.11  8.71 -3.20 -0.18
V3 -3.00 -3.20 12.68  1.63
V4  3.67 -0.18  1.63 11.43
```

```
> round(diag(X.cov), 2)

   V1    V2    V3    V4
 7.33  8.71 12.68 11.43

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

     [,1] [,2] [,3] [,4]
[1,] 0.37 0.00 0.00  0.0
[2,] 0.00 0.34 0.00  0.0
[3,] 0.00 0.00 0.28  0.0
[4,] 0.00 0.00 0.00  0.3

> X.cor <- sdi %*% X.cov %*% sdi
> round(X.cor, 2)

      [,1]  [,2]  [,3]  [,4]
[1,]  1.00  0.01 -0.31  0.40
[2,]  0.01  1.00 -0.30 -0.02
[3,] -0.31 -0.30  1.00  0.14
[4,]  0.40 -0.02  0.14  1.00

> round(cor(Xij), 2)

      V1    V2    V3    V4
V1  1.00  0.01 -0.31  0.40
V2  0.01  1.00 -0.30 -0.02
V3 -0.31 -0.30  1.00  0.14
V4  0.40 -0.02  0.14  1.00
```

2.4 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],ncol=dim(X.cov)[2])

2.5 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 provides the beta weights.

Given the equation Y = bX + c, we can solve for b by multiplying both sides of the equation by X^{-1} or $YX^{-1} = b\ XX^{-1} = b$

We can find the inverse by using the solve function. To show that $XX^{-1} = X^{-1}X = I$, we do the multiplication.

```
> X.inv <- solve(X.cov)

            V1          V2          V3          V4
V1  0.19638636  0.01817060  0.06024476 -0.07130491
V2  0.01817060  0.12828756  0.03787166 -0.00924279
V3  0.06024476  0.03787166  0.10707738 -0.03402838
V4 -0.07130491 -0.00924279 -0.03402838  0.11504850

> round(X.cov %*% X.inv, 2)
```

```
   V1 V2 V3 V4
V1  1  0  0  0
V2  0  1  0  0
V3  0  0  1  0
V4  0  0  0  1
```

```
> round(X.inv %*% X.cov, 2)
```

```
   V1 V2 V3 V4
V1  1  0  0  0
V2  0  1  0  0
V3  0  0  1  0
V4  0  0  0  1
```

There are multiple ways of finding the matrix inverse, solve is just one of them.

3 Matrix operations for data manipulation

Using the basic matrix operations of addition and multiplication allow for easy manipulation of data. In particular, finding subsets of data, scoring multiple scales for one set of items, or finding correlations and reliabilities of composite scales are all operations that are easy to do with matrix operations.

In the next example we consider 5 extraversion items for 200 subjects collected as part of the Synthetic Aperture Personality Assessment project. The items are taken from the International Personality Item Pool (ipip.ori.org). The data are stored at the personality-project.org web site and may be retrieved in R.

```
> datafilename = "http://personality-project.org/R/datasets/extraversion.items.txt"
> items = read.table(datafilename, header = TRUE)
> items <- items[, -1]
> dim(items)
```

```
[1] 200   5
```

We first use functions from the psych package to describe these data both numerically and graphically.

```
> library(psych)
```

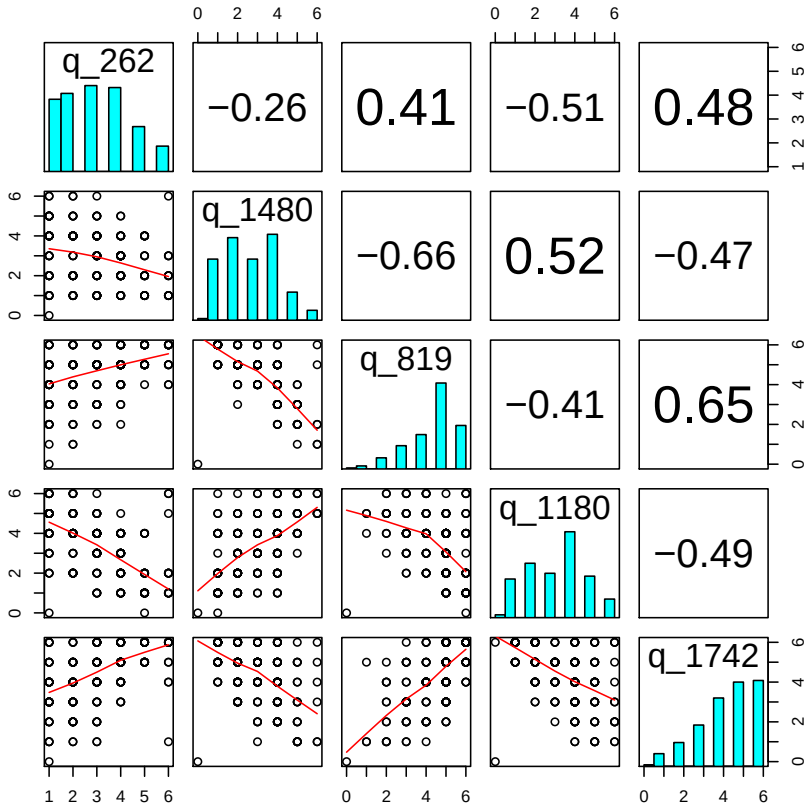
```
[1] "sem"       "psych"     "methods"   "stats"     "graphics"  "grDevices" "utils"     "datasets"
```

```
> describe(items)
```

```
       var   n mean   sd median min max range   se
q_262    1 200 3.07 1.49      3   1   6     5 0.11
q_1480   2 200 2.88 1.38      3   0   6     6 0.10
q_819    3 200 4.57 1.23      5   0   6     6 0.09
q_1180   4 200 3.29 1.49      4   0   6     6 0.11
q_1742   5 200 4.38 1.44      5   0   6     6 0.10
```

```
> pairs.panels(items)
```

```
NULL
```



We can form two composite scales, one made up of the first 3 items, the other made up of the last 2 items. Note that the second (q1480) and fourth (q1180) are negatively correlated with the remaining 3 items. This implies that we should reverse these items before scoring.

To form the composite scales, reverse the items, and find the covariances and then correlations between the scales may be done by matrix operations on either the items or on the covariances between the items. In either case, we want to define a "keys" matrix describing which items to combine on which scale. The correlations are, of course, merely the covariances divided by the square root of the variances.

3.1 matrix operations on the raw data

```
> keys <- matrix(c(1, -1, 1, 0, 0, 0, 0, 0, -1, 1), ncol = 2)
> X <- as.matrix(items)
> X.ij <- X %*% keys
> n <- dim(X.ij)[1]
> one <- rep(1, dim(X.ij)[1])
> X.means <- t(one) %*% X.ij/n
> X.cov <- t(X.ij - one %*% X.means) %*% (X.ij - one %*% X.means)/(n - 1)
> round(X.cov, 2)

      [,1] [,2]
[1,] 10.45 6.09
[2,]  6.09 6.37

> X.sd <- diag(1/sqrt(diag(X.cov)))
> X.cor <- t(X.sd) %*% X.cov %*% (X.sd)
```

```
> round(X.cor, 2)

     [,1] [,2]
[1,] 1.00 0.75
[2,] 0.75 1.00
```

3.2 matrix operations on the correlation matrix

```
> keys <- matrix(c(1, -1, 1, 0, 0, 0, 0, 0, -1, 1), ncol = 2)
> X.cor <- cor(X)
> X.cov <- t(keys) %*% X.cor %*% keys
> X.sd <- diag(1/sqrt(diag(X.cov)))
> X.cor <- t(X.sd) %*% X.cov %*% (X.sd)
> round(X.cor, 2)

     [,1] [,2]
[1,] 1.00 0.74
[2,] 0.74 1.00
```

3.3 Using matrices to find test reliability

The reliability of a test may be thought of as the correlation of the test with a test just like it. One conventional estimate of reliability, based upon the concepts from domain sampling theory, is coefficient alpha (*alpha*). For a test with just one factor, α is an estimate of the amount of the test variance due to that factor. However, if there are multiple factors in the test, α neither estimates how much the variance of the test is due to one, general factor, nor does it estimate the correlation of the test with another test just like it. (See Zinbarg et al., 2005 for a discussion of alternative estimates of reliabiity.)

Given either a covariance or correlation matrix of items, α may be found by simple matrix operations:

1) V = the correlation or covariance matrix
2) Let Vt = the sum of all the items in the correlation matrix for that scale.
3) Let n = number of items in the scale
3) alpha = (Vt - diag(V))/Vt * n/(n-1)

To demonstrate the use of matrices to find coefficient α, consider five items measuring extraversion taken from the International Personality Item Pool. Two of the items need to be weighted negatively (reverse scored).

Alpha may be found from either the correlation matrix (standardized alpha) or the covariance matrix (raw alpha). In the case of standardized alpha, the diag(V) is the same as the number of items. Using a key matrix, we can find the reliability of 3 different scales, the first is made up of the first 3 items, the second of the last 2, and the third is made up of all the items.

```
> datafilename = "http://personality-project.org/R/datasets/extraversion.items.txt"
> items = read.table(datafilename, header = TRUE)
> items <- items[, -1]
> key <- matrix(c(1, -1, 1, 0, 0, 0, 0, 0, -1, 1, 1, -1, 1, -1, 1), ncol = 3)
> raw.r <- cor(items)
> V <- t(key) %*% raw.r %*% key
> round(V, 2)

     [,1] [,2]  [,3]
[1,] 5.66 3.05  8.72
[2,] 3.05 2.97  6.03
[3,] 8.72 6.03 14.75
```

```
> n <- diag(t(key) %*% key)
> alpha <- (diag(V) - n)/(diag(V)) * (n/(n - 1))
> round(alpha, 2)

[1] 0.71 0.66 0.83
```

4 Multiple correlation

Given a set of n predictors of a criterion variable, what is the optimal weighting of the n predictors? This is, of course, the problem of multiple correlation or multiple regression. Although we would normally use the linear model (lm) function to solve this problem, we can also do it from the raw data or from a matrix of covariances or correlations by using matrix operations and the solve function.

At the data level, with X a matrix of deviation scores, we can write the equation

$$\hat{Y} = X\beta + \epsilon \tag{1}$$

and solve for

$$\beta = (XX')^{-1}X'Y \tag{2}$$

.

At the structure level, R = XX' and X'Y may be replaced with r_{xy} and we solve the equation

$$\beta = R^{-1}r_{xy} \tag{3}$$

Consider the set of 3 variables with intercorrelations (R)

```
     x1   x2   x3
x1 1.00 0.56 0.48
x2 0.56 1.00 0.42
x3 0.48 0.42 1.00
```

and correlations of x with y (r_{xy})

```
   x1   x2  x3
y 0.4 0.35 0.3
```

From the correlation matrix, we can use the solve function to find the optimal beta weights.

```
> R <- matrix(c(1, 0.56, 0.48, 0.56, 1, 0.42, 0.48, 0.42, 1), ncol = 3)
> rxy <- matrix(c(0.4, 0.35, 0.3), ncol = 1)
> colnames(R) <- rownames(R) <- c("x1", "x2", "x3")
> rownames(rxy) <- c("x1", "x2", "x3")
> colnames(rxy) <- "y"
> beta <- solve(R, rxy)
> round(beta, 2)

      y
x1 0.26
x2 0.16
x3 0.11
```

4.1 Non optimal weights and the goodness of fit

Although the beta weights are optimal given the data, it is well known (e.g., the robust beauty of linear models by Robyn Dawes) that if the predictors are all adequate, the error in prediction of Y is rather insensitive to the weights that are used. This can be shown graphically by comparing varying the weights of x1 and x2 relative to x3 and then finding the error in prediction. Note that the surface is relatively flat at its minimum.

We show this for sevaral different values of the r_{xy} and R matrix by first defining two functions (f and g) and then applying these functions with different values of R and r_{xy}. The first, f, finds the multiple r for values of $b_{x1}/b_{x3}$and b_{x2}/b_{x3} for any value or set of values given by the second function. ranging from low to high and then find the error variance $(1 - r^2)$ for each case.

```
> f <- function(x, y) {
+     xy <- diag(c(x, y, 1))
+     c <- rxy %*% xy
+     d <- xy %*% R %*% xy
+     cd <- sum(c)/sqrt(sum(d))
+     return(cd)
+ }
> g <- function(rxy, R, n = 60, low = -2, high = 4, ...) {
+     op <- par(bg = "white")
+     x <- seq(low, high, length = n)
+     y <- x
+     z <- outer(x, y)
+     for (i in 1:n) {
+         for (j in 1:n) {
+             r <- f(x[i], y[j])
+             z[i, j] <- 1 - r^2
+         }
+     }
+     persp(x, y, z, theta = 40, phi = 30, expand = 0.5, col = "lightblue", ltheta = 120, shade = 0.
+         ticktype = "detailed", zlim = c(0.5, 1), xlab = "x1/x3", ylab = "x2/x3", zlab = "Error")
+     zmin <- which.min(z)
+     ymin <- trunc(zmin/n)
+     xmin <- zmin - ymin * n
+     xval <- x[xmin + 1]
+     yval <- y[trunc(ymin) + 1]
+     title(paste("Error as function of relative weights  min values at x1/x3 = ", round(xval,
+         1), " x2/x3 = ", round(yval, 1)))
+ }
> R <- matrix(c(1, 0.56, 0.48, 0.56, 1, 0.42, 0.48, 0.42, 1), ncol = 3)
> rxy <- matrix(c(0.4, 0.35, 0.3), nrow = 1)
> colnames(R) <- rownames(R) <- c("x1", "x2", "x3")
> colnames(rxy) <- c("x1", "x2", "x3")
> rownames(rxy) <- "y"
```

Graph of residual error variance as a function of relative weights of $b_{x1}/b_{x3}$and b_{x2}/b_{x3} for

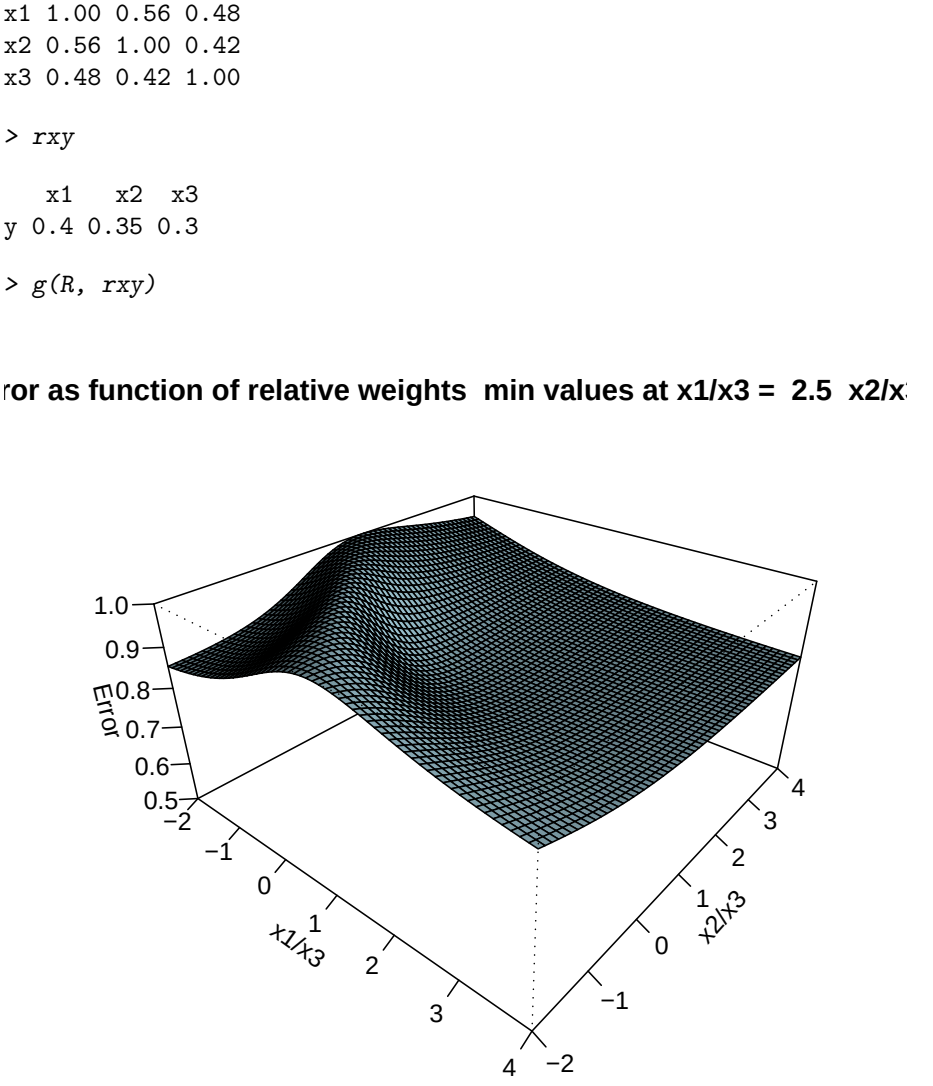
```
> R

     x1   x2   x3
x1 1.00 0.56 0.48
x2 0.56 1.00 0.42
x3 0.48 0.42 1.00

> rxy

   x1   x2  x3
y 0.4 0.35 0.3

> g(R, rxy)
```



Show the response surface for uncorrelated predictors:

```
   x1 x2 x3
x1  1  0  0
x2  0  1  0
x3  0  0  1

   x1   x2  x3
y 0.4 0.35 0.3
```