7.0 Matrices, Lists and Functions in `Splus`

Notes based on those of Professor Joe Schafer

7.1 Removing stuff from your workspace

To see what objects are stored in your workspace (the .Data directory), use `ls()`:

```
> ls() # list the objects in the workspace
```

To remove objects, use `rm()`:

```
> rm(x,y,z)     # removes x, y, and z
```

If you want to remove everything from your .Data workspace, one simple way to do it is to exit from Splus and use the ls command in Unix. For example:

```
> q()
relative:  > rm .Data/*
```

7.2 More about matrices

Look at the following example, in which we create a vector of length 4 and then reshape it into a 3x8 matrix.

```
> x <- 1:4
> matrix(x,3,8)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    4    3    2    1    4    3    2
[2,]    2    1    4    3    2    1    4    3
[3,]    3    2    1    4    3    2    1    4
```

Because the length of the matrix exceeded the length of the vector x, Splus filled in the matrix by repeating x over and over. If the length of the matrix had not been an integer multiple of length(x), a warning message would have been given.

Suppose you want to perform some computation on the rows or columns of a matrix–for example, compute the mean of each column. The function `apply()` will do it. If x is an n x p matrix, then

```
> apply(x,2,mean)
```

will return a vector of length p containing the means of the columns in x. The argument "2" in "apply(x,2,mean)" tells Splus to perform the computation on the second dimension (i.e. the columns) of x. Typing

```
> apply(x,1,mean)
```

will create a vector of length n containing the means of the rows in x. In addition to means, `apply()` can calculate variances, medians, sums, products, etc. You can supply as the third argument to `apply()` just about any Splus function that computes a single-number summary

of a vector. Here are some examples.

```
> apply(x,2,sum) # column sums
> apply(x,2,var) # column variances
> sqrt(apply(x,2,var))  # standard deviations
```

The function `diag()` can be used to create a diagonal matrix (i.e. a matrix in which all the off-diagonal elements are zero) or to extract the diagonal elements of a square matrix. If you supply a vector as the argument to `diag()`, Splus creates a diagonal matrix with that vector as its diagonal.

```
> diag(rep(1,5)) # creates a 5 x 5 identity matrix
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
```

If you supply a square matrix as the argument to `diag()`, Splus extracts the diagonal and returns it as a vector.

```
> x
     [,1] [,2] [,3]
[1,]    2    1    0
[2,]    1    3    5
[3,]    3    1    4
> diag(x)
[1] 2 3 4
```

7.3 Example

Here's an example of a fairly elaborate matrix computation. Suppose x is a two-way contingency table–i.e., an r x c matrix of counts–and we want to perform the standard Pearson χ^2 test for independence. To do this, we must calculate the expected counts using the well-known formula

$$\text{expected count} = \frac{\text{row total x column total}}{\text{grand total}}$$

and then take the sum of

$$\frac{\text{(observed count - expected count)\^{}2}}{\text{expected count}}$$

over all cells.

First, we create a matrix called "rowtot" with the same dimensions as x, but whose elements are the row totals. We do this by calculating the row totals with `apply()` and then reshaping the resulting vector into an r x c matrix:

```
> rowtot <- apply(x,1,sum)
> rowtot <- matrix(rowtot,nrow(x),ncol(x))
```

Then we do the same with the column totals:

```
> coltot <- apply(x,2,sum)
> coltot <- matrix(coltot,nrow(x),ncol(x),byrow=T)
```

We finish the computation as follows:

```
> expec <- rowtot*coltot/sum(x)
```

```
> X2 <- sum((x-expec)^2/expec)
```

An approximate p-value for testing independence can be found by comparing Pearson's χ^2 statistic to a chisquare distribution with degrees of freedom given by:

```
> degf <- (nrow(x)-1)*(ncol(x)-1)
```

(Note: I didn't call my degrees of freedom "df" because that is the name of an Splus function, the probability density for the F-distribution. If you create an object whose name is identical to an existing Splus function, some confusion may result. Typing "help(df)" is a quick way to check whether there is an existing Splus function named "`df()`".) The p-value is the area under the chisquare curve to the right of χ^2, which is the same thing as taking one minus the cumulative distribution (area to the left) of χ^2. The function `pchisq()` calculates the cumulative distribution for a chisquare:

```
> pval <- 1-pchisq(X2,degf)
```

7.4 Lists

One of the nicest features of Splus is that it can create and manipulate lists. A list is a collection of separate objects glued together so that they can be referred to by a single name. The objects glued together need not be of the same type. For example, you can make a list out of a numeric matrix, a character string, and a logical vector. You can make lists with the `list()` function, as in this example:

```
> x <- matrix(1:4,4,7)
> y <- "Your grandmother wears combat boots."
> z <- c(T,F,T,NA)
> mylist <- list(x,y,z)
> mylist
[[1]]:
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    1    1    1    1    1    1
[2,]    2    2    2    2    2    2    2
[3,]    3    3    3    3    3    3    3
[4,]    4    4    4    4    4    4    4

[[2]]:
[1] "Your grandmother wears combat boots."

[[3]]:
[1]  T  F  T NA
```

In the example above, the list "mylist" has three elements. The first element is a matrix, the second element is a character string, and the third element is a logical vector. You can access individual elements of a list using double square brackets:

```
> mylist[[1]]
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    1    1    1    1    1    1
[2,]    2    2    2    2    2    2    2
```

```
[3,]    3    3    3    3    3    3    3
[4,]    4    4    4    4    4    4    4
> mylist[[2]]
[1] "Your grandmother wears combat boots."
> mylist[[3]]
[1]  T  F  T NA
> mylist[[3]][1:2]
[1] T F
```

You can also assign names to each element in the list, and then refer to the elements by their names. For example, let's recreate the same list and name the elements "Tom", "Dick", and "Harry":

```
> mylist <- list(Tom=x,Dick=y,Harry=z)
> mylist
$Tom:
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    1    1    1    1    1    1
[2,]    2    2    2    2    2    2    2
[3,]    3    3    3    3    3    3    3
[4,]    4    4    4    4    4    4    4

$Dick:
[1] "Your grandmother wears combat boots."

$Harry:
[1]  T  F  T NA
```

Now you can refer to the elements of the list by their names. To refer to an element of a list by its name, type the name of the list, a dollar sign ($), and the name of the element:

```
> mylist$Dick
[1] "Your grandmother wears combat boots."
> mylist$Harry
[1]  T  F  T NA
```

You can also supply the name enclosed in quotes and square brackets:

```
> mylist[["Dick"]]       # print out any elements of mylist
                         # named "Dick"
[1] "Your grandmother wears combat boots."
```

The most important attributes of a list are its length and its names. The `length()` of a list is the number of elements in the list. The `names()` of a list is a vector of character strings of the same length as the list, giving the names of the list's elements.

```
> length(mylist)
[1] 3
> names(mylist)
[1] "Tom"   "Dick"  "Harry"
```

Lists are a convenient way to summarize the results of a complex manipulation. For example, Splus has a function called `lsfit()` that performs least-squares regression. When you supply a vector of responses and a matrix of predictors, the function `lsfit()` calculates the regression and returns a list containing a vector of estimated coefficients, a vector of residuals, etc.

7.5 Naming the rows and columns of a matrix

Last week we said that you could assign names to the rows and columns of a matrix, but we didn't describe the procedure because we hadn't yet introduced the idea of a list. Now that we know what a list is, we can describe how to do it.

A matrix in Splus has an attribute called `dimnames()`. If x is a matrix, then dimnames(x) is a list with two elements. The first element of dimnames(x) is a vector of character strings of length nrow(x) containing the row names. The second element of dimnames(x) is a vector of character strings of length ncol(x) containing the column names. We can add row and column names to the matrix x by creating such a list and assigning it to dimnames(x). Look at the following example.

```
> grades <- c(100,99,100,97,54,26)
> grades <- matrix(grades,3,2,byrow=T)
> dimnames(grades) <- list(c("John","Linda","Sylvestre"),
+   c("Midterm","Final"))
> grades
          Midterm Final
     John     100    99
    Linda     100    97
Sylvestre      54    26
```

If you want to give names to the columns but not to the rows, then use "NULL" for the row names. "NULL" is a placeholder to tell Splus that something doesn't exist.

```
>  dimnames(grades) <- list(NULL,c("Midterm","Final"))
> grades
     Midterm Final
[1,]     100    99
```

```
[2,]     100    97
[3,]      54    26
> dimnames(grades) <- list(c("John","Linda","Sylvestre"),NULL)
> grades
          [,1] [,2]
     John  100   99
    Linda  100   97
Sylvestre   54   26
```

If the rows or columns of a matrix have been named, you can then refer to specific rows or columns by supplying their names in square brackets. For example:

```
> dimnames(grades) <- list(c("John","Linda","Sylvestre"),
+  c("Midterm","Final"))
> grades
          Midterm Final
     John     100    99
    Linda     100    97
Sylvestre      54    26
> grades["Linda",]
 Midterm Final
     100    97
> grades[,"Final"]
 John Linda Sylvestre
   99    97        26
```

7.6 Functions

We have already worked with a wide variety of Splus functions. Some of these functions—`sqrt()`, `log()`, `mean()`, etc.—perform computations on numeric objects and return other numeric objects. Other functions—`c()`, `rep()`, `matrix()`, `list()`, etc.—are useful for creating and storing data in a variety of formats. Other functions—`length()`, `dim()`, `nrow()`, `ncol()`, `names()`, `dimnames()`, etc.—allow us to access and modify the attributes of data objects. Yet other functions—`motif()`, `graphics.off()`, `rm()`, `q()`, etc.—tell Splus to perform some specific action, but after execution these functions do not create or return any objects.

It is very easy to write your own functions in Splus. If there is some complicated command or group of commands that you need to execute repeatedly, you can create a function to automatically execute these commands and save yourself from a lot of tedious typing.

You can create your own function like this:

```
> myfunction <- function(argument1,argument2,...)
+     first command here
        +     second command here
        +     etc.
        +     final object to be returned here
```

For example, let's write a function called "getstats" that accepts a data matrix, computes the mean, median, and standard deviation of each column, and returns the result as a matrix with three rows.

```
> getstats <- function(x)
+    means <- apply(x,2,mean)
+    medians <- apply(x,2,median)
```

```
+    stdevs <- sqrt(apply(x,2,var))
+    result <- rbind(means,medians,stdevs)
+    dimnames(result) <-
+       list(c("mean","median","standard dev."),NULL)
+    result
```

Now if we create some matrices–call them x, y, and z–and type

```
> getstats(x)
> getstats(y)
> getstats(z)
```

Splus will print out the summary statistics for the columns of x, y, and z.

Any objects created within a function are local to that function. For example, the function `getstats()` as defined above creates four local objects: "means", "medians", "stdevs", and "result". When you execute the function by typing "getstats(y)", these four objects are created temporarily but then deleted afterward. (These temporary objects are stored in main memory (RAM), not on the disk, so you don't need to worry about these locally defined objects overwriting anything in your .Data directory with the same names.)

Suppose you want to create a function that tells Splus to carry out some commands, but does not return any object at the end. For example, suppose you want to write a function called "plotvec" that accepts a vector, opens a graphics window, and plots the elements of the vector against the integers 1,2,... If you make the final line of your function "NULL", then the function will return nothing. However, this has the annoying effect of printing the word "NULL" after the function executes. To get around this problem, make the final line

"`invisible()`", which generates a null value without printing.

```
> plotvec <- function(x)
+   motif()
+   plot(1:length(x),x)
+   invisible()
```

Finally, let's write a function that carries out the Pearson χ^2 test for independence:

```
> indep.test <- function(x)
+   rowtot <- apply(x,1,sum)
+   rowtot <- matrix(rowtot,nrow(x),ncol(x))
+   coltot <- apply(x,2,sum)
+   coltot <- matrix(coltot,nrow(x),ncol(x),byrow=T)
+   expec <- rowtot*coltot/sum(x)
+   X2 <- sum((x-expec)^2/expec)
+   degf <- (nrow(x)-1)*(ncol(x)-1)
+   pval <- 1-pchisq(X2,degf)
+   list(X2=X2,df=degf,pvalue=pval,expected=expec)
```

This functions returns a list with four elements: the value of the χ^2 statistic (named "X2"), the degrees of freedom (named "df"), the p-value (named "pvalue"), and a matrix of expected counts (called "matrix").