ISSUE 34 | JANUARY 2010

SOCIETY OF ACTUARIES Technology Section

CompAct

ELECTRONIC NEWSLETTER

CONTENTS

QUICK LINKS

Technology Section Web site

Council

Links of Interest

Share Print Article Search Back issues

R CORNER–FUNCTIONS

by Steve Craighead

Editor's note: R Corner[1] is a series by Steve Craighead introducing readers to the "R" language used for statistics and modeling of data. The first column was published in the October 2008 issue, and explains how to download and install the package, as well as providing a basic introduction to the language. Refer to each CompAct issue since then for additional articles in the series. The introductory article can be found on p. 24 of the October 2008 issue on the SOA Web site.

To design your own functions in R, you will need to follow a simple format:

```
functionname <- function( param1=default1,
param2=default2,...,paramN=defaultN)
{
# your function's calculations
return_value
}
```

For instance, a function of a single variable (with no defaults) that squares the input would be:

```
Square<-function(x)
{
x^2
}
```

Fiction Contest

Howard Callif, Editor

SOA Staff
Meg Weber, Staff Partner

Sue Martz,
Section Specialist

Sam Phillips, Staff Editor

You would use this function as

```
Square(4)
[1] 16
```

What is interesting about this function is that it can be processed against vectors or matrices (and other data structures).

Vector:

```
Square(1:10)
[1] 1 4 9 16 25 36 49 64 81 100
```

Matrix:

```
(M <- matrix(c(1:9),nrow=3))
```

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

and

```
Square(M)
```

	[,1]	[,2]	[,3]
[1,]	1	16	49
[2,]	4	25	64
[3,]	9	36	81

So in a sense, what appears to be a function in a single variable, actually can be treated as having both multivariate input and output.

Now, if you wanted the function to have a default value, you would write your function as so:

```
Square2(x=0)
{
x^2
}
```

Now, if you invoke the function Square2 as follows, the default value will be used.

```
Square2()
[1] 0
```

Let's say that you need to do a large number of log–log plots of various data, You could create your own plot function by

```
LLplot<-function(x,y){plot(log(x),log(y))}
```

LLplot(1:10,101:110) would produce this graph:

I have been using the command area of R to create all of these functions, but as the function size grows, you will want to use a full screen editor on them. Just use the fix() function. For instance fix(LLplot), would display the function within the fix editor. I've already mentioned the fix() editor when we were examining how to manipulate data frames.

Let's revise your function so that your function would place a heading on the graph.

```
LLplot <- function(x,y,main="My Plot"){plot(log(x),log(y),main=main)}
```

Now if you just use the LLplot(1:10,101:110) as before, you will obtain the default header of "My Plot." However, if you invoke LLplot(1:10,101:110,"My Log Log Plot"), the header will use your specified header.

You have seen how to expand the input of a function to allow for multiple parameters. Now you need to see how to control the output of the function. Returning a list object as output in the last statement of the function does this. You specify the list object with both the name and the content of each output.

For instance, the function bellows squares your x values and returns the values in alpha and cubes the y values and returns them in beta.

```
Test<- function(x,y)
{
a<-x^2
b<-y^3
list( alpha = a, beta = b)
}
```

Now:

```
(answer <- Test(1:10,4:9))
$alpha [1] 1 4 9 16 25 36 49 64 81 100

$beta
```

```
[1] 64 125 216 343 512 729
```

To reaccess the values in beta, you would type answer$beta, or answer[[2]], which are basic access techniques for list objects.

Notice how the data format of x and y was maintained within the list object. If you used the above function on a vector for x and a matrix for y, the output list would contain a vector and a matrix.

All of the above examples are very simple, but you can enhance your functions by using conditional statements in them. For example, let's construct a function that squares all values less than 10 and cubes all values above 10.

```
Test2<-function(x)
{
if (x < 10) y <- x^2 else y<- x^3
y
}
```

If you test on a single value for x, the obvious transformation will be returned. However, if you used a vector input for x, with one value for x beginning greater or equal to 10, you will get the following result with a warning:

```
Test2(1:10)
[1] 1 4 9 16 25 36 49 64 81 100
Warning message:
In if (x < 10) y <- x^2 else y <- x^3 :
the condition has length > 1 and only the first element will be used
```

Also, note how the return value for 10 is 10^2 and not 10^3!

Now, let's change the function so that it maintains the same structure. Now, a conditional structure such as (x<10), is a structure that is the same as x, but the values of x are replaced by values of TRUE and FALSE.

```
Test2<-function(x)
{
(x<10)*^2+(x>=10)*x^3
}
```

The above uses conditionals on the index of x. When the conditional index value (e.g., (x<10)) is true, all values will be TRUE (effectively = 1) or FALSE (effectively = 0).

Testing Test2 on a matrix, you will see:

(x<-matrix(c(5:22),nrow=3))

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	5	8	11	14	17	20
[2,]	6	9	12	15	18	21
[3,]	7	10	13&	16	19	22

Test2(x)

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	25	64	1331	2744	4913	8000
[2,]	36	81	1728	3375	5832	9261
[3,]	49	1000	2197	4096	6859	10648

I have barely scratched the surface on how to use functions. If you would like to learn more, please read Chapter 10 in "An introduction to R" in the R environment. Access this by choosing the "Manuals (in PDF)" feature under the "Help" option dropdown list.

1Craighead, S. (2000), "Insolvency Testing: An Empirical Analysis of the Generalized Beta Type 2 Distribution, Quantile Regression, and a Resampled Extreme Value Technique," ARCH, pp. 13–149.