Using R for Actuarial Science

Shyamal Kumar

1 Introduction

The prestigious 1998 Association for Computing Machinery Award for software systems was awarded to John Chambers of Bell Labs for the S system. The **citation** reads, *For the S system, which has forever altered how people analyze, visualize, and manipulate data*. R is an open source implementation of S and S-PLUS its commercial implementation.

R is an extensible, well documented language and environment with a core group of developers spread out over many countries. The size of its global user group and the diversity in its applications are some of its many strengths. Aimed at actuaries and especially actuarial science students who are pondering over the choice of a computing platform to complement MS Excel, the insurance industry defacto standard, this article makes a case for R.

Designed for statistical computing and embraced by scores of statisticians, most if not all statistical needs of an actuary should be ably served. Hence discussion of its statistical prowess will be conspicuously absent. The next section will highlight some features of potential interest to actuaries. Life contingent computations use mortality tables and most of the important tables are found in the SOA tables database.The third section discusses an implementation of access to this binary database with the intention of providing the readers with a good starting point for computing on the life side. Vectorization, the subject of the penultimate section, is an important concept for computing on R, like on APL and other interpreted vector languages . There we discuss a vectorized solution for an important class of actuarial algorithms.

2 Some Features of R

First, R has an effective programming language with a simple syntax. ***The R Language Definition***, describes the syntax as having *"a superficial similarity with C, but the semantics are of the FPL (functional programming language) variety with stronger affinities with Lisp and APL."*. ***A Brief History of S*** by Richard A. Becker is a worth a read to know of the influences of other languages on the development of S. For example, the following is taken from it; *"... the basic interactive operation of S, the parse/eval/print loop, was a well-explored concept, occurring in APL, Troll, LISP, and PPL, among others. From APL we borrowed the concept of the multi-way array (although we did not make it our basic data structure), and the overall consistency of operations. The notion of a typeless language (with no declarations) was also present in APL and PPL."*.

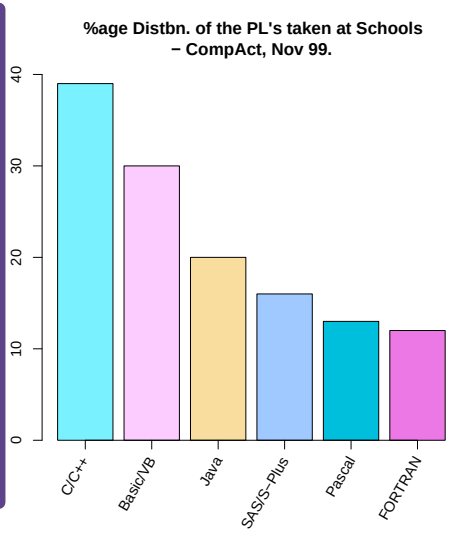
Second, R is highly extensible and seamless extensions are carried out using packages. Importantly, C and FORTRAN code can be linked and called at run time from R not only enabling

N. D. Shyamal Kumar ASA, an assistant professor of Actuarial Science at the University of Iowa, is a member of the computer science section council. He can be contacted at shyamal-kumar@uiowa.edu.

use of existing code but also providing avenues for accelerating computationally intensive tasks. And in the other direction, R objects can be manipulated in C. Talking about speed, support for BLAS (Basic Linear Algebra Subroutines) is provided in R. Also, there is a package called snow which implements a simple mechanism for computing on workstation clusters. For more details consult **Writing R Extensions**.

Third, its reputed ability to produce high quality graphics with a minimal amount of code. The base graphics system contains both high and low level commands. For a demo of its capabilities, type `demo(graphics)` on the R command prompt. Especially important is its ability to generate graphics for publication in many formats including postscript, pdf, jpeg and png. Also, there is a separate graphics sub-system called grid which is considered to be more powerful and another package called gridBase for combining grid and base graphics output. Below is an example of the use of gridBase, similar to one in **Murrell (2003)**, to combine the ease of a highlevel base graphics system command to draw a barplot with the power of grid used here to rotate the x-axis labels.

```
midpts <- barplot(c(39,30,20,16,13,12), axes = FALSE,
  col=c("#7AF1FE","#FCCBFF","#F8DD9E","#9FC9FF",
  "#00BFDD","#EB78E4"),ylim=c(0,40));
axis(2); axis(1, at = midpts, labels = FALSE);
vps<-baseViewports();
pushViewport(vps$inner, vps$figure,vps$plot);
grid.text(c("C/C++","Basic/VB","Java","SAS/S-Plus",
  "Pascal","FORTRAN"),x=unit(midpts,"native"),
  y=unit(-1,"lines"),just="right",rot=60);
popViewport(3);
```



And last but not the least, R has many utilities for accessing data. Access to data resident in a Relational Database Management Systems is provided by several packages. Any system providing an ODBC interface can be accessed using RODBC. This list not only includes most important RDBMSs but also databases like MS Access. Also, on Windows, ODBC drivers for text files, Excel files and Dbase files are available. System specific packages are available for Oracle and MySQL. R provides tools for accessing data in a binary format which is used in the implementation of the next section. There is a package to parse XML files using either the DOM or the SAX mechanism. Of course, there is a rich support for reading text files. Also, there are packages which make R a DCOM client (and server too) which help in accessing for example, live (financial) data. ***R Data Import/Export*** is the relevant manual for details.

Since R is well documented, the above description has been kept brief. Moreover, the above list is just a small subset of its wide ranging capabilities. Observe that some of the described

features are not part of the *core* of R but of one of the many packages which seamlessly extend it. Access to such features requires installation of relevant package(s), a matter of just a few *clicks*. At this point, a tour of its official web site at **http://www.r-project.org** and a browse of **An Introduction to R** would be a good idea. And if already convinced to try it out, go ahead and install your free download!

3 SOA Tables Database

Here we discuss an implementation of access to the SOA tables database. To keep the discussion self contained, the first sub-section describes the SOA tables database files (binary format) `tables.dat` and `tables.ndx`. The next sub-section discusses the implementation along with some examples of its use.

3.1 Binary Files of SOA DB

The database owes its existence to a joint project of the Technology Section of the SOA, the International Section of the SOA and other volunteers. It consists of two binary files - `tables.dat`, the data file, and `tables.ndx`, the index file. The description of the storage format for these files can be found in the help of **TableManager**, a software copyrighted by Steve Strommen, FSA, MAAA, and distributed freely by the SOA. The description is repeated below to keep the article self contained. The **Table Manager web page** also has a MS Excel addin available for free download for those interested in importing table values directly into Excel. A visit to the page is highly recommended.

`tables.ndx`

The primary purpose of this file, as any index file, is to facilitate fast access to a table from the `tables.dat` file. This is done by providing the offset, i.e. the number of bytes from the start of the file, at which the data for a table begins. This is a binary file with a sequence of fixed size records with an initial offset of 58 bytes (due to descriptive information). Each record has five fields as depicted in the figure below. The character strings in this file, unlike those of the `tables.dat`, are NULL byte terminated (as in C). The usage code is the same as that of the data file.

4 Bytes	50 Bytes	4 Bytes	31 Bytes	1 Byte
32-bit Integer	Character String	32-bit Integer	Character String	8-bit Integer
Table Number	Table Name	Offset	Country	Usage Code
4	54	58	89	90

Figure 1 An Index File Record

Record Types for tables.dat

Type	Content	Storage Format	Possible Values
1	Table Name	Character String	
2	Table Number	32-bit Integer	
3	Table Type	Single Character	• S - Select • A - Aggregate by Age • D - Aggregate by Duration
4	Contributor	Character String	
5	Source	Character String	
6	Volume	Character String	
7	Observation Period	Character String	
8	Unit of Observation	Character String	
9	Construction Method	Character String	
10	Reference	Character String	
11	Comments	Character String	
12	Minimum Age	16-bit Integer	
13	Maximum Age	16-bit Integer	
14	Select Period	16-bit Integer	
15	Maximum Select Age	16-bit Integer	
16	No. of Decimal Places	16-bit Integer	
17	Table Values	Sequence of 8-byte IEEE floating types	
18	Hash Value	32-bit Unsigned Integer	
19	Country	Character String	
20	Usage	8-bit Integer	• 0 - No Data • 1 - Insured Mortality • 2 - Annuitant Mortality • 3 - Population Mortality • 4 - Selection Factors • 5 - Projection Scale • 6 - Lapse Rates • 99 - Miscellaneous

tables.dat

This binary file is a sequence of generic records with a single consecutive sub-sequence of such records (terminating with a record of type 9999 with missing length and data fields) pertaining to a single table. The storage format of a generic record is shown below.

Record Type Code - 16 bit Integer	Length of the Data - 16 bit Integer	Data - Variable Length

Figure 2 A Generic Record

There are twenty record types as listed in the following inset. A caveat is that not all the record types are relevant to a table and moreover not all the relevant types are necessarily present. The latter becomes important as the data packaged as list may contain some *empty* components. The character strings neither have a terminating NULL byte (like C) nor a leading byte containing its length (like Pascal) - but note that the length can be read off the third and fourth bytes of the generic record. R as a platform is convenient to read binary files as it has support for reading IEEE floating point numbers.

The table values are stored in the ascending order of the index (Age or Duration) for types "A" and "D". For type "S", the order is issue age wise in the ascending order of the select duration until we hit the maximum issue age and thereafter in the ascending order of the age.

3.2 Implementation

RCode for TblSearch

```
"TblSearch" <-
function (Na = "", C = "", U = "", No = "")
{
    rec <- function(...) {
        c(readBin(z, integer(), size = 4), sub("", "", readChar(z,
            50)), readBin(z, integer(), size = 4), sub("", "",
            readChar(z, 31)), readBin(z, integer(), size = 1))
    }
    readChar(z <- file("tables.ndx", "rb"), 58)
    x <- sapply(1:((file.info("tables.ndx")$size - 58)/90), rec)
    close(z)
    apply(matrix(c(Na, C, U, No, 2, 4, 5, 1), 4, 2), 1, function(y) {
        x <<- x[, grep(y[1], x[as.integer(y[2]), ], perl = TRUE),
            drop = FALSE]
    })
    data.frame(No = as.integer(x[1, ]), Name = x[2, ], Country = x[4,
        ], Usage = as.integer(x[5, ]), Offset = as.integer(x[3,
        ]))
}
```

The first function, `TblSearch`, provides a query facility for the file `tables.ndx`. The query has to be in the form of a **PERL *regular expression*** and can be over each of the fields excepting offset. The result is the set of records satisfying all of the individual constraints (intersection) packaged as a dataframe object. This function not only helps a user search for tables in the database but also is used by the function Tbl. The following are some comments on the code.

i. A user may leave unspecified the trailing arguments which would then assume the default null value..
ii. The function `rec` reads the five fields for each record and is used by the `sapply` function. `sapply`, used to avoid an explicit loop, returns an array containing the fields for all of the records in the file.
iii. Once the records are read, successively the regular expressions are used to filter the records using the `apply` function.
iv. The last statement packages the array as a dataframe object.

Below are some examples of its use.

1. To list the US insured mortality age last tables for male smokers, one could use `TblSearch("(?i)[^e]male.*[^n]smoker.*last","US")`

2. For US insured mortality age last basic tables of 1980, one could use `TblSearch("(?i)1980.*basic.*nearest","US","1")`

3. To list just the names of all US annuitant mortality tables, one could use `TblSearch("","US","2")$Name`

The second function, `Tbl`, given a vector of either table numbers or offsets returns a list of lists containing all the fields for each table requested. If the input is a vector of table numbers then `TblSearch` is used to get the offsets. The code could have been a lot shorter but for the need to have this function vectorized - this way there is just a single read of the file for all the tables combined. It does not return a dataframe like the earlier function because the mortality rates data is of varying sizes. To facilitate recursive indexing of the list of lists, the following function has proved useful. It is left uncommented as the logic is rather straightforward.

```
"RI" <-
function (z, Name)
{
    sapply(1:length(z), function(x) {
        z[[c(x, grep(paste("^", Name, "$", sep = ""), c("Name",
            "Number", "Type", "Contributor", "Source", "Volume",
            "ObsnPeriod", "ObsnUnit", "Method", "Reference",
            "Comments", "MinAge", "MaxAge", "SelPeriod", "MaxSelAge",
            "NumDec", "Rates", "HashValue", "Country", "Usage"),
            perl = TRUE))]]
    })
}
```

RCode for Tbl

```
"Tbl" <-
function (Offset, Num = TRUE)
{
    FT <- c(1, 2, array(1, 9), array(2, 5), 3, 2, 1, 2)
    Table <- vector(length(Offset), mode = "list")
    Len <- 0
    z <- file("tables.dat", "rb")
    if (Num) {
        Offset <- TblSearch("", "", "", paste("^", as.character(Offset),
            "$", sep = "", collapse = "|"))$Offset
    }
    Offset <- sort(Offset)
    Offset <- Offset - c(0, Offset[-length(Offset)])
    for (i in 1:length(Offset)) {
        readChar(z, Offset[i] - Len)
        Len <- 0
        F <- 0
        T <- vector(20, mode = "list")
        attributes(T) <- list(names = c("Name", "Number", "Type",
            "Contributor", "Source", "Volume", "ObsnPeriod",
            "ObsnUnit", "Method", "Reference", "Comments", "MinAge",
            "MaxAge", "SelPeriod", "MaxSelAge", "NumDec", "Rates",
            "HashValue", "Country", "Usage"))
        while (F != 9999) {
            F <- readBin(z, integer(), size = 2)
            Len <- Len + 2
            if (F != 9999) {
                l <- readBin(z, integer(), size = 2)
                Len <- Len + 2 + l
                T[[F]] <- switch(FT[F], readChar(z, l), readBin(z,
                  integer(), size = l), as.array(readBin(z, double(),
                  n = l/8, size = 8)))
            }
        }
        Table[[i]] <- T
    }
    close(z)
    Table
}
```

Below are some queries, similar to the ones above, using the additional fields from `tables.dat`. Examples of life contingency calculations will be part of the next section.

1. To find the names of all select and ultimate US insured mortality tables, one could use
```
RI((y<-Tbl(TblSearch("","US","1")$Offset,FALSE))[sapply(1:length(y),function(x){
  y[[x]]$Type})=="S"],"Name")
```

2. The following lists the names of insured aggregate mortality tables whose rates do not satisfy the monotonicity beyond the age 35.
```
x<-(y<-Tbl(sort(TblSearch("","","1")$Offset),FALSE))[sapply(1:length(y),function(x){
  y[[x]]$Type})=="A"]
RI(x[as.logical(1-sapply(x,function(z) {min((z$Rates[-length(z$Rates)]<=z$Rates[-1])
  [-(1:(35-z$MinAge))])}))],"Name")
```

4 Vectorization

Most actuarial quantities on the life side satisfy a linear difference equation of the first order. On compiled languages, a simple loop is the way to go and the direction in most actuarial problems happens to be backward, in some forward and in the rest one could choose either one. This is the way one computes on any spread sheet platform too - the loop becoming relative references to the preceding or succeeding cell.

When working with an interpreted vector language, explicit loops are inefficient in the presence of an algorithm which is able to *vectorize* the problem. It is not that vectorization removes the loop but rather that it makes the loop implicit i.e. it is executed internally. In the actuarial literature, see Shiu (1987), Shiu & Seah (1987) and Giles (1993), it has been noted that the usual *closed form* solution of the linear difference equation translates to elegant (concise) APL code. More than the brevity of the APL code that the closed form yields to a vectorized solution is important. In other words, as in Shiu (1987), that the closed form solution can be translated to a code which avoids explicit loops is the key. Hence this solution will lead to not only concise but an efficient solution to the problem on all vector languages where explicit loops are inefficient. Below are the details.

Life contingencies abounds with difference equations of the first order. Some are listed below.

1. The curtate future lifetime of a life aged (x) satisfies,

$$e_x = p_x + p_x e_{x+1}$$

2. The actuarial present value of a whole life insurance with benefits payable at the end of the year of death satisfies,

$$A_x = \nu q_x + \nu p_x A_{x+1}$$

3. Fackler's formula for the reserves of a fully discrete annual whole life on (x) satisfies,

$$ {}_nV_x = \nu q_{x+n} - P_x + \nu p_{x+n}\ {}_{n+1}V_x$$

4. Hattendorff's formula for the variance of the loss random variable underlying a fully discrete annual whole life on (x) satisfies,

$$\mathrm{Var}\left({}_nL|K(x) \geq n\right) = \nu^2 q_{x+n} p_{x+n} \left(1 - {}_{n+1}V_x\right)^2 + \nu^2 p_{x+n} \mathrm{Var}\left({}_{n+1}L|K(x) \geq n+1\right)$$

All of the above are backward in nature as the boundary value at the right end is known. Moreover, they are each just a particular case of the equation

$$x_n = a_n + b_n x_{n+1}, \quad n = 1, 2, \ldots, k \text{ with } x_{k+1} \text{ known}$$

It is easy to show that the solution is given by,

$$x_n = \frac{x_{k+1} \prod_0^k b_i + \sum_{l=n}^k a_l \prod_0^{l-1} b_i}{\prod_0^{n-1} b_i}, \quad \text{where } b_0 = 1 \text{ and } n = 1, 2, \ldots, k$$

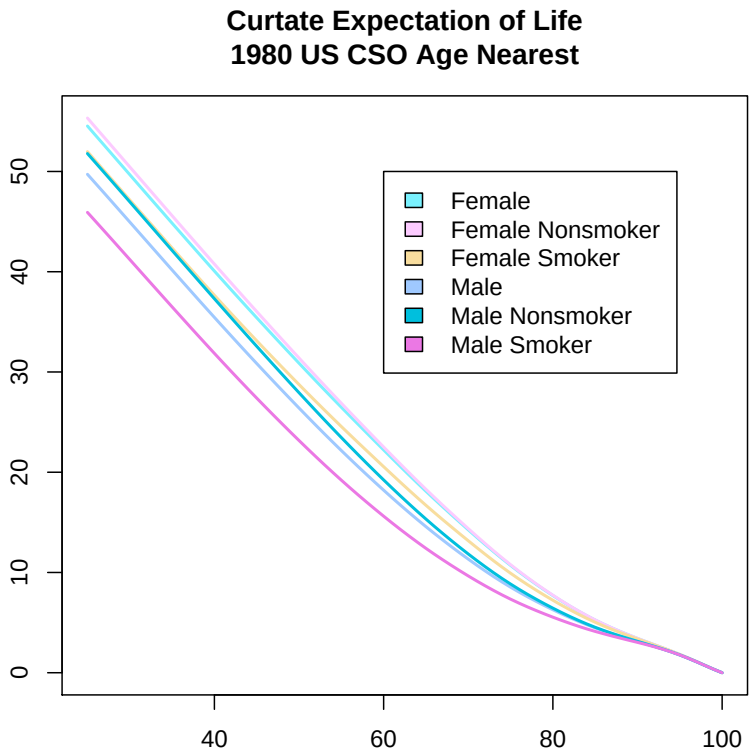
The R code for the above, which for matter of style has not been compressed into a single line, is encapsulated by the function `BD`.

```
"BD" <-
function (a, b, bv)
{
    s <- rev(cumprod(c(1, b)));
    (rev(cumsum(s[-1] * rev(a))) + s[1] * bv)/rev(s[-1])
}
```

As an example, below is an R code for a plot of the curve of curtate life expectation for each of the CSO 1980 basic age nearest tables. Note that the curve for female smokers is higher than that of male non-smokers.

```
y<-c("#7AF1FE","#FCCBFF","#F8DD9E","#9FC9FF","#00BFDD","#EB78E4")
x<-Tbl(TblSearch("(?i)1980.*basic.*nearest","US","1")$Offset,FALSE)
matplot(25:100,sapply(x,function(z) {BD(a,a<-(1-z$Rates[-c(1:(25-z$MinAge))]),0)}),type="l",
  lwd=2,ylab="",xlab="x",lty=1,col=y,main="Curtate Expectation of Life\n 1980 US CSO Age
  Nearest")
legend(x=70,y=50,gsub("(1980 US CSO Basic|Age nearest)","",RI(x,"Name")),fill=y,
  horiz=FALSE)
```

5 Conclusion



As some of the other interpreted vector languages, R is a fantastic tool for rapid prototyping; besides, it offers a large coherent set of utilities. This combination of a prototyping language with an environment could be particularly desirable by those who compute as part of their job while not being responsible for development of software systems. In fact, R allows one to smoothly move from a prototype to a compiled code as it allows for one to access such code from within R.

For an actuarial science program, a powerful case can be made for adoption of R as the standard software across courses.

Acknowledgement

I thank Elias Shiu and Luke Tierney for valuable discussions. All opinions and errors are mine alone. This article has been written using mainly ConTEXt (a TEX macro-package), Metapost, PERL and R. I am greatly indebted to all involved in their development.

References

Giles, T. L. 1993. Life Insurance Application of Recursive Formulas. *Journal of Actuarial Practice* 1, No. 2:141-151..

Shiu, E. S. W. 1987. Discussion of "Life Insurance Transformations" by Douglas A. Eckley. *Transaction of the Society of Actuaries* XXXIX, 17-18.

Shiu, E. S. W. and Seah, E. 1987. Discussion of "Financial Accounting Standards No. 87: Recursion Formulas and other Related Matters" by Barnet N. Berin and Eric P. Lofgren Douglas A. Eckley. *Transaction of the Society of Actuaries* XXXIX, 37-40.