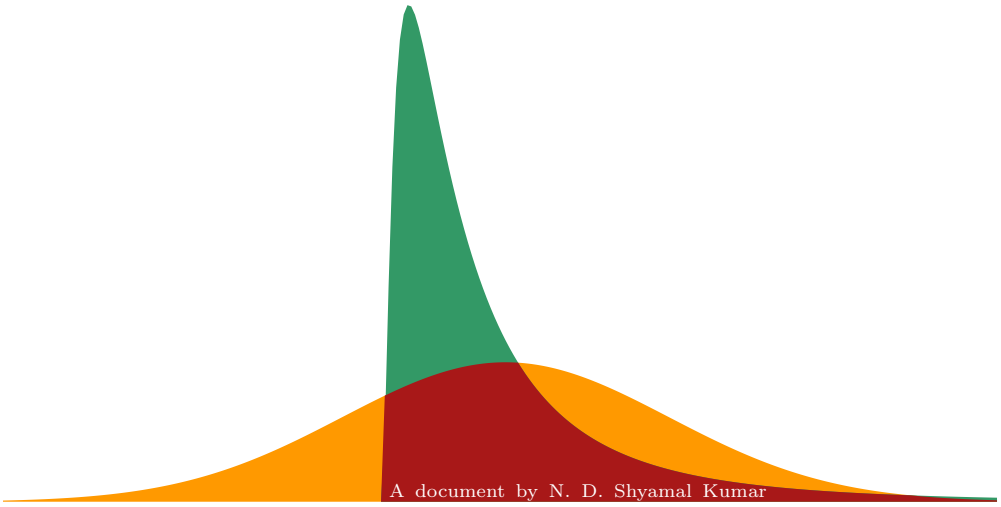
Shyamal Kumar

Actuarial Computing with R

N. D. Shyamal Kumar
The University of Iowa



R you Ready for R?

Actuarial Computing with R

Pre-Print

To Padmalatha

Preface

How to Use This Book

Contents

1 APV 15

2 Premiums 17

3 Reserves 21

4 Installation 37

5 The Manual 41
5.1 Introduction 43
5.2 Mortality DB Tools 45
5.2.1 SoA DB 45
5.2.2 Access to DB 47
5.2.3 Miscellany 52
5.3 General Utilities 53
5.4 Fractional Age Distribution 55
5.4.1 Introduction 55
5.4.2 The Functions 57
5.5 Simulation Tools 65
5.5.1 Introduction 65
5.5.2 The Functions 65

6 The Code 71
6.1 Access to Mortality DB 73
6.2 Utilities for Mortality Tables 79
6.3 Tools for Simulation 81
6.4 Fractional Age Distribution Utilities 85
6.5 General Utilities 93

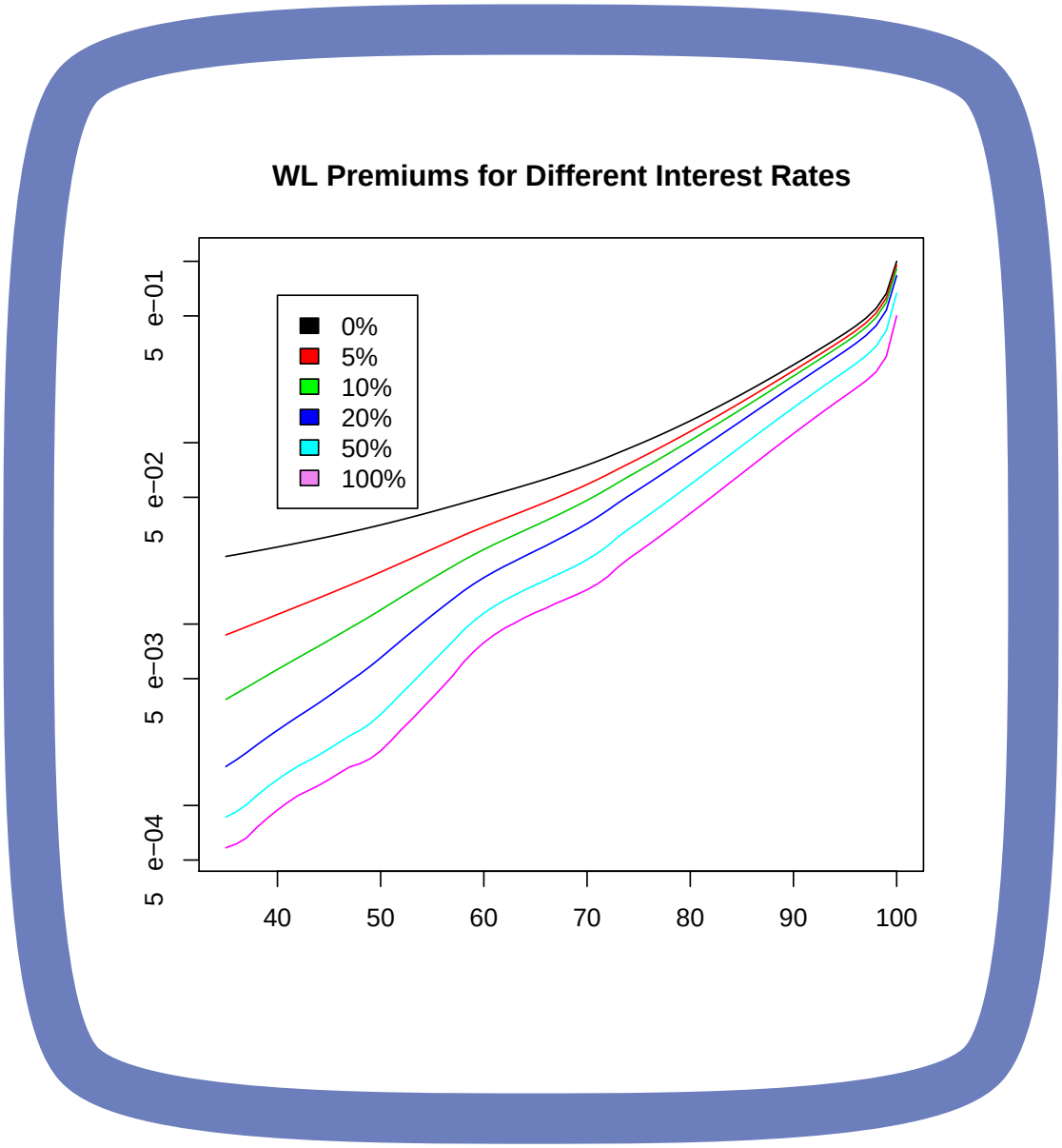
APV

1 Premium Rates v/s Interest Rates 19

Premiums

RCode

```
q<-Tb(15006)$Values$Aggregate[36:101];
f<-SK(q)[1:66]*q;
Premium<-BLFOR(q,(1-q),0)/BLFOR(array(1,length(q)),(1-q),0);
for (i in c(.05,.1,.2,.5,1)){
    Premium=cbind(Premium,BLFOR(q/(1+i),(1-q)/(1+i),0)/
                BLFOR(array(1,length(q)),(1-q)/(1+i),0));
}
matplot(c(35:100),Premium,xlab=NA,ylab=NA,log="y",type="l",lty=1,
        lwd=1,main="WL Premiums for Different Interest Rates");
legend(x=40,y=0.65,c("0%","5%","10%","20%","50%","100%"),col=blue,
        fill=c("black","red","green","blue","cyan","violet"));
```



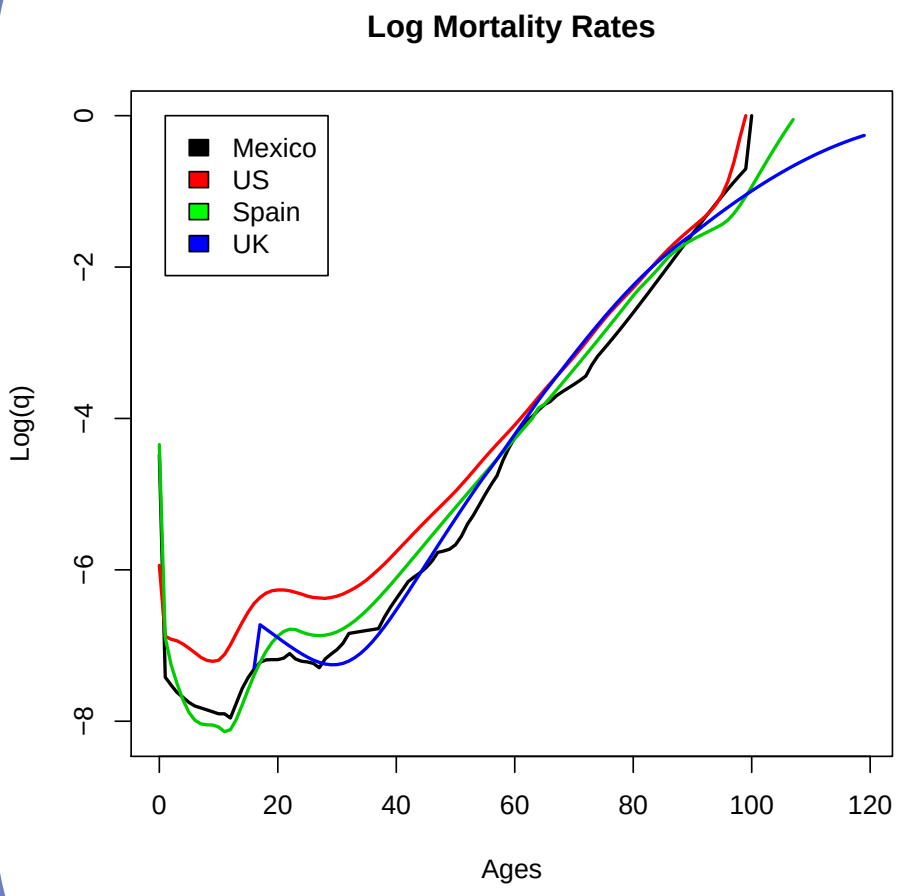
Example 1 Here we plot the whole life premiums, between ages 0 and 100 for the AMA AMIS 2000 Male table, for different interest rates to show that due to the premiums being a weighted average of OYT premiums the premium curves get steeper as the interest rate increases. This phenomenon is not limited only to whole life products though. Can you generalize?

1	Raison D'Être - Mortality Curve Pattern	23
2	Distributions of Loss Random Variables	25
3	Semi-Variance vs Variance	27
4	Comparison of Reserves	29
5	Discounted Annual Loss Variances	31
6	Standard Deviations of Future Loss	33
7	Level NAR Products	35

Reserves

RCode

```
y<-c(log(Tb(15006)$Values$Aggregate),array(NA,19));
y<-cbind(y,c(log(Tb(41)$Values$Aggregate),array(NA,20)));
y<-cbind(y,c(log(Tb(653)$Values$Aggregate),array(NA,12)));
y<-cbind(y,c(array(NA,16),log(Tb(836)$Values$Aggregate)));
par(mfrow=c(1,1));
matplot((0:119),y,type="l",lty=1,lwd=2,xlab="Ages",ylab="Log(q)",
            main="Log Mortality Rates");
legend(x=1,y=0,col=blue,fill=c("black","red","green","blue"),
        c("Mexico","US","Spain","UK"),horiz=FALSE)
```

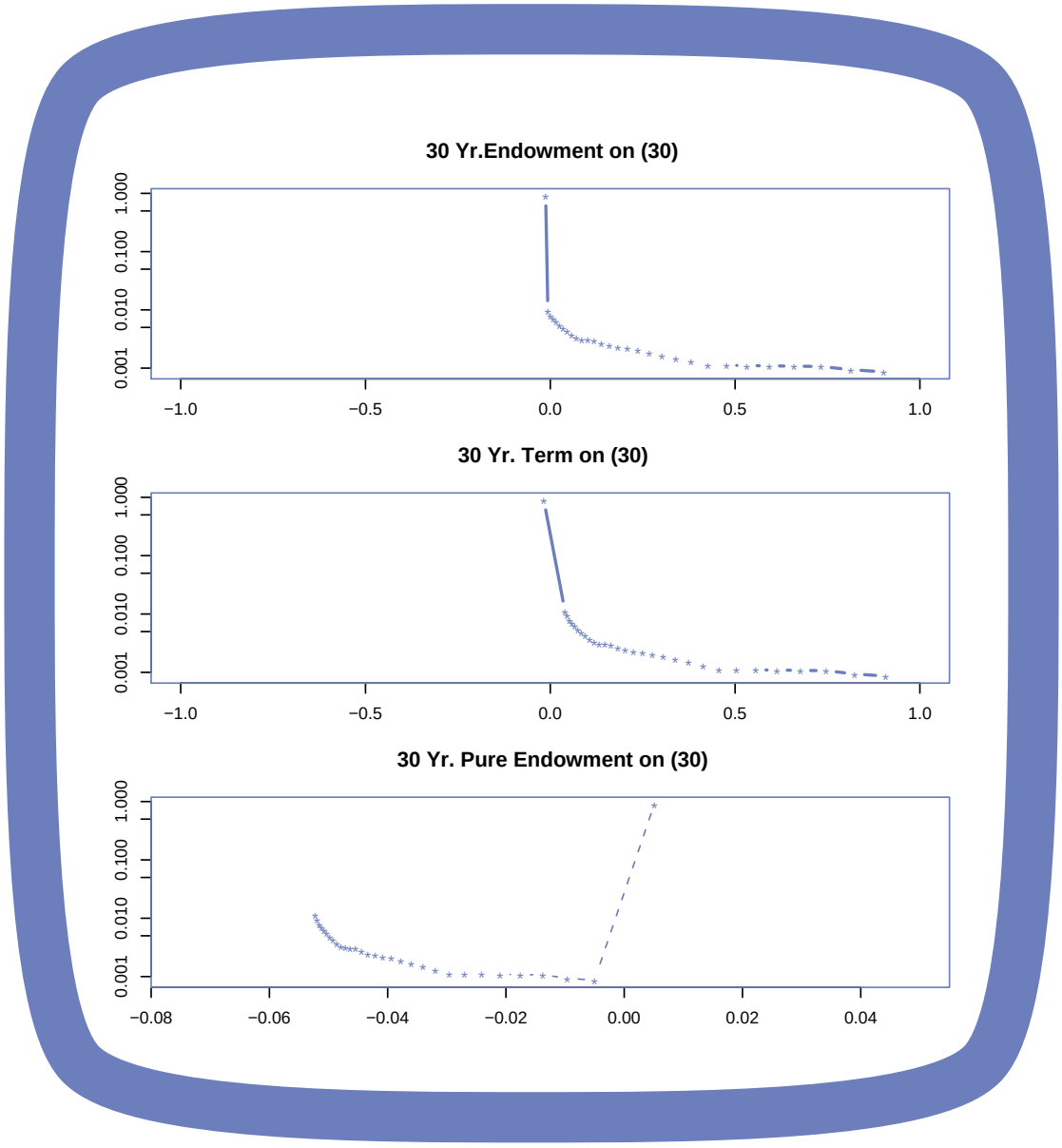


Example 2 We start this chapter by recalling the basic nature of a mortality curve - decreasing at birth due to infant mortality, a hump starting around age 18 due to accidental mortality and then a monotonous increasing pattern after around an age of 35. This predominantly increasing pattern combined with a typical non-increasing income levels of a policyholder (leading to level premium policies) gives rise to reserves.

RCode

```
q<-Tb(15006)$Values$Aggregate;
delta<-log(1.1);x<-30;n<-30;
par(mfrow=c(4,1),col="#6C7EBB",mai=c(0.25,.5,.5,.25),
        lwd=1,lty=2,pch="*");
## Whole Life
    qd<-q[(x+1):101];
    f<-SK(qd)[1:(101-x)]*qd;
    Premium<-BLFOR(qd/1.1,(1-qd)/1.1,0)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/1.1,0)[1];
    l<-exp(-delta*(1:(101-x)))*(1+Premium/.1*1.1)-Premium/.1*1.1;
    plot(l,f,xlab=NA,ylab=NA,xlim=c(-1,1),log="y",type="b",
            main=paste("Whole Life on (",x,")",sep=""));
## Endowment
    qd<-q[(x+1):(x+n)];
    f=SK(qd)[1:n]*qd;
    f[n]=f[n]+SK(qd)[n+1]; ## Survival gets the same benefit
                           ## as death in the last year
    Premium<-BLFOR(qd/1.1,(1-qd)/1.1,1)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/1.1,0)[1];
    l<-exp(-delta*(1:n))*(1+Premium/.1*1.1)-Premium/.1*1.1;
    plot(l,f,xlab=NA,ylab=NA,xlim=c(-1,1),log="y",type="b",
            main=paste(n," Yr.Endowment on (",x,")",sep=""));
## Term Insurance
    f=SK(qd)[1:n]*qd;
    f=c(f,SK(qd)[n+1]); ## Append Survival as another event
                        ## as L takes another unique value
    Premium<-BLFOR(qd/1.1,(1-qd)/1.1,0)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/1.1,0)[1];
    l<-exp(-delta*(1:n))*(1+Premium/.1*1.1)-Premium/.1*1.1;
    l<-c(l,-Premium*(1-exp(-delta*n))/.1*1.1);
    plot(l,f,xlab=NA,ylab=NA,xlim=c(-1,1),log="y",type="b",
            main=paste(n," Yr. Term on (",x,")",sep=""));
## Pure Endowment
    f=SK(qd)[1:n]*qd;
    f=c(SK(qd)[n+1],f); ## Append Survival as another event
                        ## as L takes another unique value
    Premium<-BLFOR(array(0,length(qd)),(1-qd)/1.1,1)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/1.1,0)[1];
    l<- -Premium*(1-exp(-delta*(1:n)))/.1*1.1;
    l<-c(exp(-delta*n)-Premium*(1-exp(-delta*n))/.1*1.1,l);
    plot(l,f,xlab=NA,ylab=NA,xlim=c(-.075,.05),log="y",type="b",
            main=paste(n," Yr. Pure Endowment on (",x,")",sep=""));
```

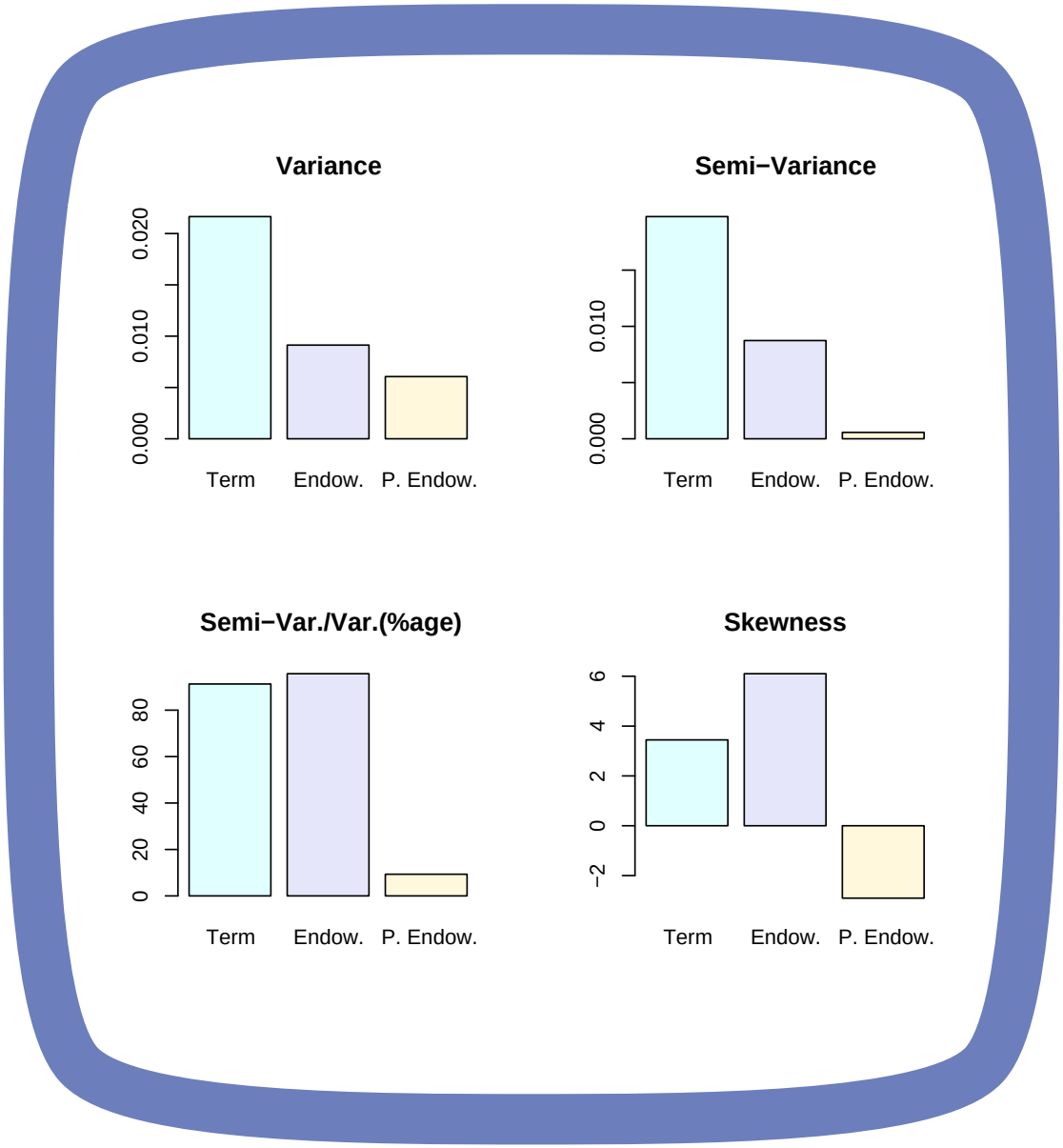
Example 3 Here we plot the probability mass functions of the loss random variable at time zero for the whole life, term and endowment products. Notice the difference between the first three products and the pure endowment - Does it merit further analysis?



RCode

```
q<-Tb(15006)$Values$Aggregate;
x<-30;n<-30;i<-.04;delta<-log(1+i);d<-i/(1+i);
par(mfrow=c(2,2),col="#6C7EBB");
## Endowment
    qd<-q[(x+1):(x+n)];
    f=SK(qd)[1:n]*qd;
    f[n]=f[n]+SK(qd)[n+1];
    Premium<-BLFOR(qd/(1+i),(1-qd)/(1+i),1)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    l<-exp(-delta*(1:n))*(1+Premium/d)-Premium/d;
    DS<-as.array(c(sum(l^2*f),sum(pmax(l,0)^2*f),sum(l^3*f)/
                    sum(l^2*f)^1.5))
## Term Insurance
    f=SK(qd)[1:n]*qd;
    f=c(f,SK(qd)[n+1]);
    Premium<-BLFOR(qd/(1+i),(1-qd)/(1+i),0)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    l<-exp(-delta*(1:n))*(1+Premium/d)-Premium/d;
    l<-c(l,-Premium*(1-exp(-delta*n))/d);
    DS<-cbind(DS,as.array(c(sum(l^2*f),sum(pmax(l,0)^2*f),sum(l^3*f)/
                            sum(l^2*f)^1.5)));
## Pure Endowment
    f=SK(qd)[1:n]*qd;
    f=c(SK(qd)[n+1],f);
    Premium<-BLFOR(array(0,length(qd)),(1-qd)/(1+i),1)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    l<- -Premium*(1-exp(-delta*(1:n)))/d;
    l<-c(exp(-delta*n)-Premium*(1-exp(-delta*n))/d,l);
    DS<-cbind(DS,as.array(c(sum(l^2*f),sum(pmax(l,0)^2*f),sum(l^3*f)/
                            sum(l^2*f)^1.5)));
## Bar Chart
    DS<-t(cbind(DS[,2],DS[,1],DS[,3]));
    DS<-cbind(DS[,1:2],round(DS[,2]/DS[,1]*100,2),DS[,3]);
    rownames(DS)<-c("Term","Endow.","P. Endow.");
    colnames(DS)<-c("Variance","Semi-Variance",
                    "Semi-Var./Var.(%age)","Skewness");
    for (i in 1:4){
        barplot(DS[,i],col=c("lightcyan","lavender",
                "cornsilk"),main=colnames(DS)[i]);
    }
```

Example 4 Semi-Variance is shown here to be a more relevant as a measure of risk than variance.

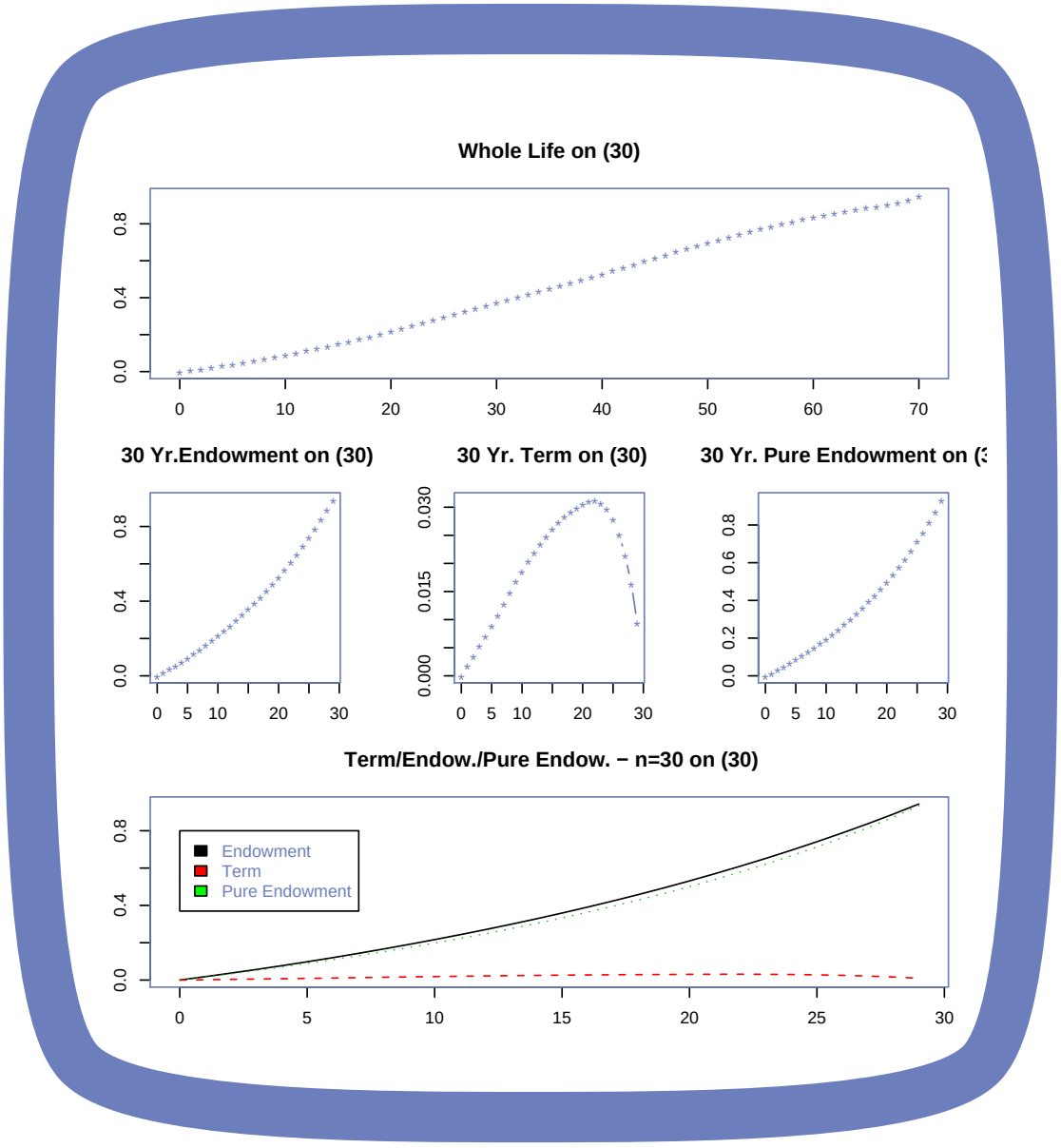


RCode

```
q<-Tb(15006)$Values$Aggregate;
x<-30;n<-30;i<-.04;delta<-log(1+i);d<-i/(1+i);
par(col="#6C7EBB",mai=c(0.25,.5,.5,.25),lwd=1,lty=1,pch="*");
layout(matrix(c(1,1,1:5,5,5),3,3,byrow=TRUE))
## Whole Life
    qd<-q[(x+1):101];
    Premium<-BLFOR(qd/(1+i),(1-qd)/(1+i),0)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    V<-BLFOR(qd/(1+i)-Premium,(1-qd)/(1+i),0);
    plot((0:(100-x)),V,xlab=NA,ylab=NA,type="b",
            main=paste("Whole Life on (",x,")",sep=""));
## Endowment
    qd<-q[(x+1):(x+n)];
    Premium<-BLFOR(qd/(1+i),(1-qd)/(1+i),1)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    V<-BLFOR(qd/(1+i)-Premium,(1-qd)/(1+i),1);
    MATV<-V;
    plot((0:(n-1)),V,xlab=NA,ylab=NA,type="b",
            main=paste(n," Yr.Endowment on (",x,")",sep=""));
## Term Insurance
    Premium<-BLFOR(qd/(1+i),(1-qd)/(1+i),0)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    V<-BLFOR(qd/(1+i)-Premium,(1-qd)/(1+i),0);
    MATV<-cbind(MATV,V);
    plot((0:(n-1)),V,xlab=NA,ylab=NA,type="b",
            main=paste(n," Yr. Term on (",x,")",sep=""));
## Pure Endowment
    Premium<-BLFOR(array(0,length(qd)),(1-qd)/(1+i),1)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    V<-BLFOR(array(-Premium,length(qd)),(1-qd)/(1+i),1);
    MATV<-cbind(MATV,V);
    plot((0:(n-1)),V,xlab=NA,ylab=NA,type="b",
            main=paste(n," Yr. Pure Endowment on (",x,")",sep=""));
    matplot((0:(n-1)),MATV,xlab=NA,ylab=NA,type="l",
            main=paste("Term/Endow./Pure Endow. - n=",n," on (",
                        x,")",sep=""));
    legend(x=0,y=0.8,col=blue,fill=c("black","red","green"),
        c("Endowment","Term","Pure Endowment"),horiz=FALSE)
```

Example 5 Here we compare the reserves for the fully discrete case between an endowment, pure endowment, whole life and a term insurance with the same issue age (30) and excepting the whole life the same term (30). Notice the following:

- The reserves for the endowment is the sum of the reserves corresponding to the term and pure endowment
- The increasing pattern of the reserves to the sum assured in the case of the endowment and pure endowment (also whole life as $q_\omega = 1$)
- Hump pattern of a term insurance reserves (Why?)

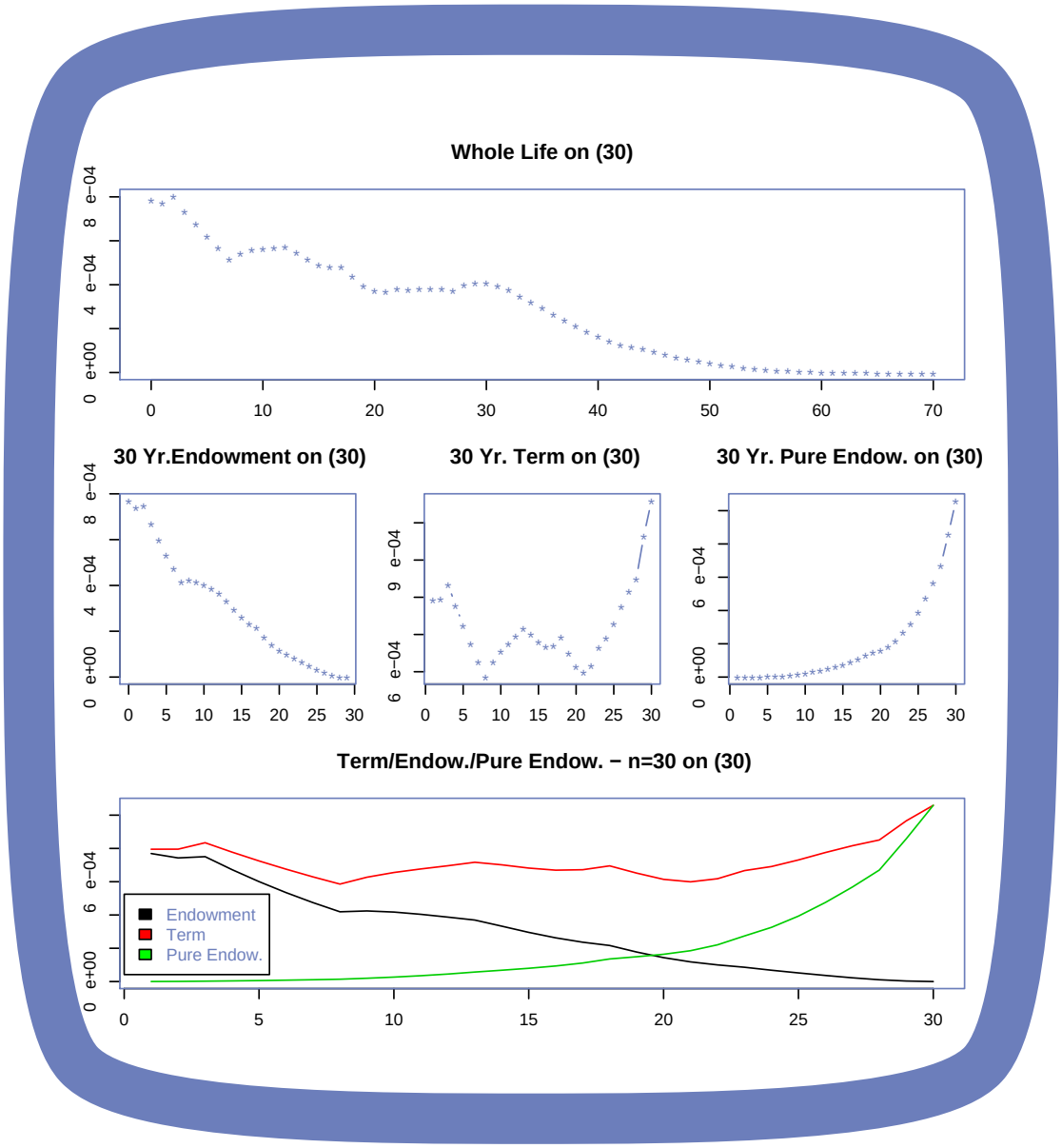


RCode

```
q<-Tb(15006)$Values$Aggregate;
k<-Tb(15006)$Values$Meta$MaxAge-Tb(15006)$Values$Meta$MinAge+1;
x<-30;n<-30;i<-.04;delta<-log(1+i);d<-i/(1+i);v<-(1+i)^(-1);
par(col="#6C7EBB",mai=c(0.25,.3,.5,.15),lwd=1,lty=1,pch="*");
layout(matrix(c(1,1,1:5,5,5),3,3,byrow=TRUE))
## Whole Life
    qd<-q[(x+1):k];z<-(1:(k-x));
    Premium<-BLFOR(qd/(1+i),(1-qd)/(1+i),0)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    V<-as.array(c(BLFOR(qd/(1+i)-Premium,(1-qd)/(1+i),1)[2:(k-x)],1));
    plot(z-1,SK(qd)[x-k]*(1-qd)*qd*v^(2*z)*(1-V)^2,xlab=NA,ylab=NA,
        type="b",main=paste("Whole Life on (",x,")",sep=""));
## Endowment
    qd<-q[(x+1):(x+n)];z<-(1:n);
    Premium<-BLFOR(qd/(1+i),(1-qd)/(1+i),1)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    V<-as.array(c(BLFOR(qd/(1+i)-Premium,(1-qd)/(1+i),1)[2:n],1));
    MATV<-SK(qd)[-n-1]*(1-qd)*qd*v^(2*z)*(1-V)^2;
    plot(z-1,MATV,xlab=NA,ylab=NA,type="b",lty=1,
            main=paste(n," Yr.Endowment on (",x,")",sep=""));
## Term Insurance
    Premium<-BLFOR(qd/(1+i),(1-qd)/(1+i),0)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    V<-as.array(c(BLFOR(qd/(1+i)-Premium,(1-qd)/(1+i),0)[2:n],0));
    MATV<-cbind(MATV,SK(qd)[-1-n]*(1-qd)*qd*v^(2*z)*(1-V)^2);
    plot(z,MATV[,2],xlab=NA,ylab=NA,type="b",lty=1,
            main=paste(n," Yr. Term on (",x,")",sep=""));
## Pure Endowment
    Premium<-BLFOR(array(0,length(qd)),(1-qd)/(1+i),1)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    V<-as.array(c(BLFOR(array(-Premium,length(qd)),
                        (1-qd)/(1+i),1)[2:n],1));
    MATV<-cbind(MATV,SK(qd)[-1-n]*(1-qd)*qd*v^(2*z)*V^2);
    plot(z,MATV[,3],xlab=NA,ylab=NA,type="b",lty=1,
            main=paste(n," Yr. Pure Endow. on (",x,")",sep=""));
    matplot(z,MATV,xlab=NA,ylab=NA,type="l",lty=1,main=paste(
            "Term/Endow./Pure Endow. - n=",n," on (",x,")",sep=""));
    legend(x=0,y=0.000525,col=blue,fill=c("black","red","green"),
        c("Endowment","Term","Pure Endow."),horiz=FALSE)
```

Example 6 Here we compare the discounted annual loss variances for the fully discrete case between an endowment, pure endowment, whole life and a term insurance with the same issue age (30) and excepting the whole life the same term (30). Notice the following:

- The annual loss variances in the last year for the pure endowment and the term insurance are the same.
- Decreasing pattern of annual loss variance for the endowment
- Increasing pattern for the pure endowment and term insurance
- Hump pattern for the whole life

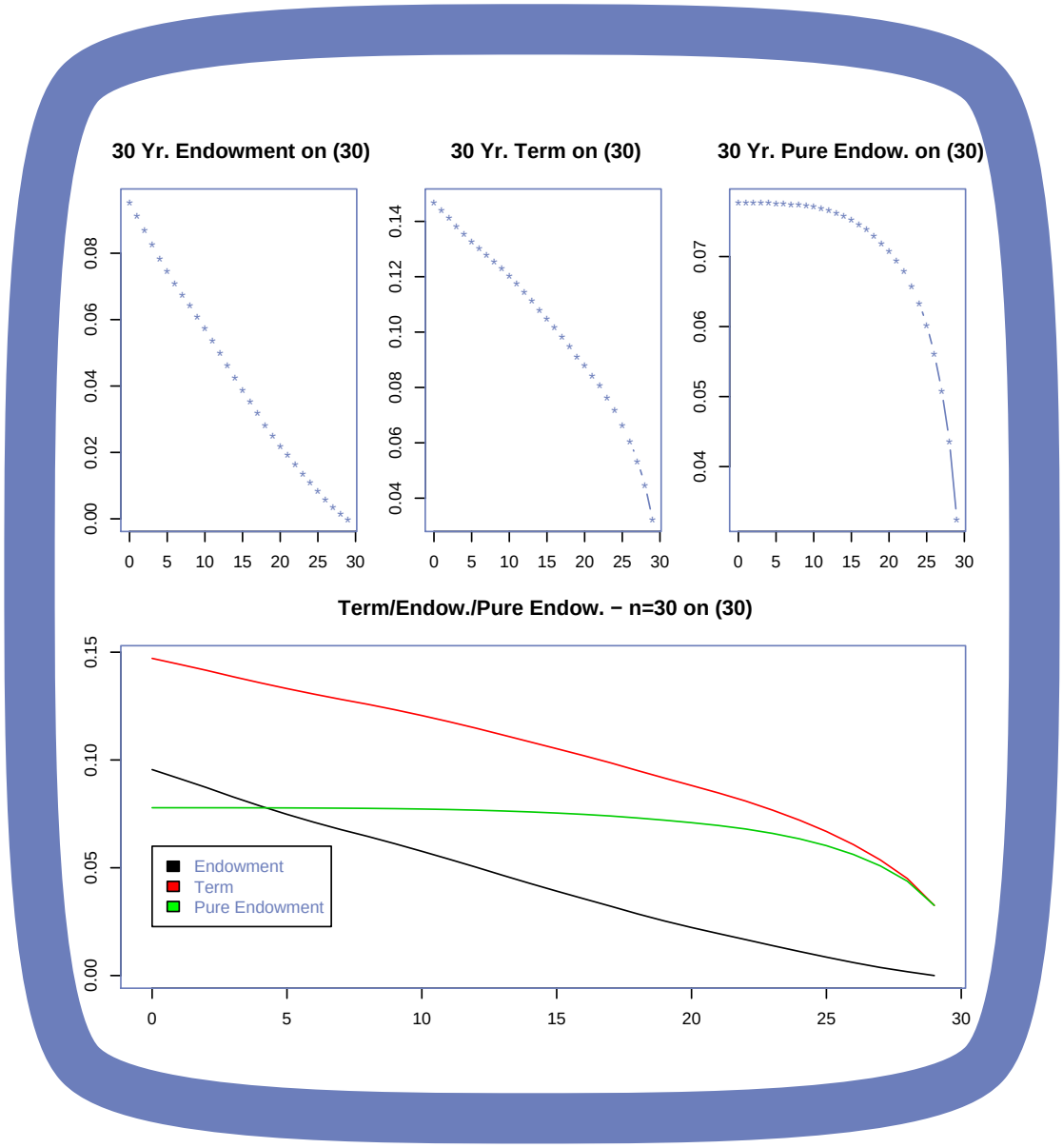


RCode

```
q<-Tb(15006)$Values$Aggregate;
k<-Tb(15006)$Values$Meta$MaxAge-Tb(15006)$Values$Meta$MinAge+1;
x<-30;n<-30;i<-.04;delta<-log(1+i);d<-i/(1+i);v<-(1+i)^(-1);
par(col="#6C7EBB",mai=c(0.25,.3,.5,.15),lwd=1,lty=1,pch="*");
layout(matrix(c(1:4,4,4),2,3,byrow=TRUE))
## Endowment
    qd<-q[(x+1):(x+n)];z<-(1:n);
    Premium<-BLFOR(qd/(1+i),(1-qd)/(1+i),1)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    V<-as.array(c(BLFOR(qd/(1+i)-Premium,(1-qd)/(1+i),1)[2:n],1));
    MATV<-SK(qd)[-n-1]*(1-qd)*qd*v^(2*z)*(1-V)^2;
## Term Insurance
    Premium<-BLFOR(qd/(1+i),(1-qd)/(1+i),0)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    V<-as.array(c(BLFOR(qd/(1+i)-Premium,(1-qd)/(1+i),0)[2:n],0));
    MATV<-cbind(MATV,SK(qd)[-1-n]*(1-qd)*qd*v^(2*z)*(1-V)^2);
## Pure Endowment
    Premium<-BLFOR(array(0,length(qd)),(1-qd)/(1+i),1)[1]/
                BLFOR(array(1,length(qd)),(1-qd)/(1+i),0)[1];
    V<-as.array(c(BLFOR(array(-Premium,length(qd)),
                        (1-qd)/(1+i),1)[2:n],1));
    MATV<-cbind(MATV,SK(qd)[-1-n]*(1-qd)*qd*v^(2*z)*V^2);
## Variance of Loss Random Variables
    MATV<-t((t(MATV)%*%outer(z,z,function(x,y) (x>=y)*1))^0.5);
## Graphical Commands
    plot(z-1,MATV[,1],xlab=NA,ylab=NA,type="b",lty=1,
            main=paste(n," Yr. Endowment on (",x,")",sep=""));
    plot(z-1,MATV[,2],xlab=NA,ylab=NA,type="b",lty=1,
            main=paste(n," Yr. Term on (",x,")",sep=""));
    plot(z-1,MATV[,3],xlab=NA,ylab=NA,type="b",lty=1,
            main=paste(n," Yr. Pure Endow. on (",x,")",sep=""));
    matplot(z-1,MATV,xlab=NA,ylab=NA,type="l",lty=1,main=paste(
            "Term/Endow./Pure Endow. - n=",n," on (",x,")",sep=""));
    legend(x=0,y=0.06,col=blue,fill=c("black","red","green"),
        c("Endowment","Term","Pure Endowment"),horiz=FALSE)
```

Example 7 Here we compare the standard deviations of the future loss for the fully discrete case between an endowment, pure endowment and a term insurance with the same issue age (30) and the same term (30). This gives us a measure of the amount of capital required by the portfolio - except in the case of a pure endowment. Notice the following:

- Same standard deviations in the last year for the pure endowment and the term insurance.
- Similar slopes for the endowment and the term insurance.
- Decrease to zero of the required capital for the endowment
- A three fold decrease in required capital for term insurance
- Pretty flat standard deviations for the pure endowment except in the final policy years.

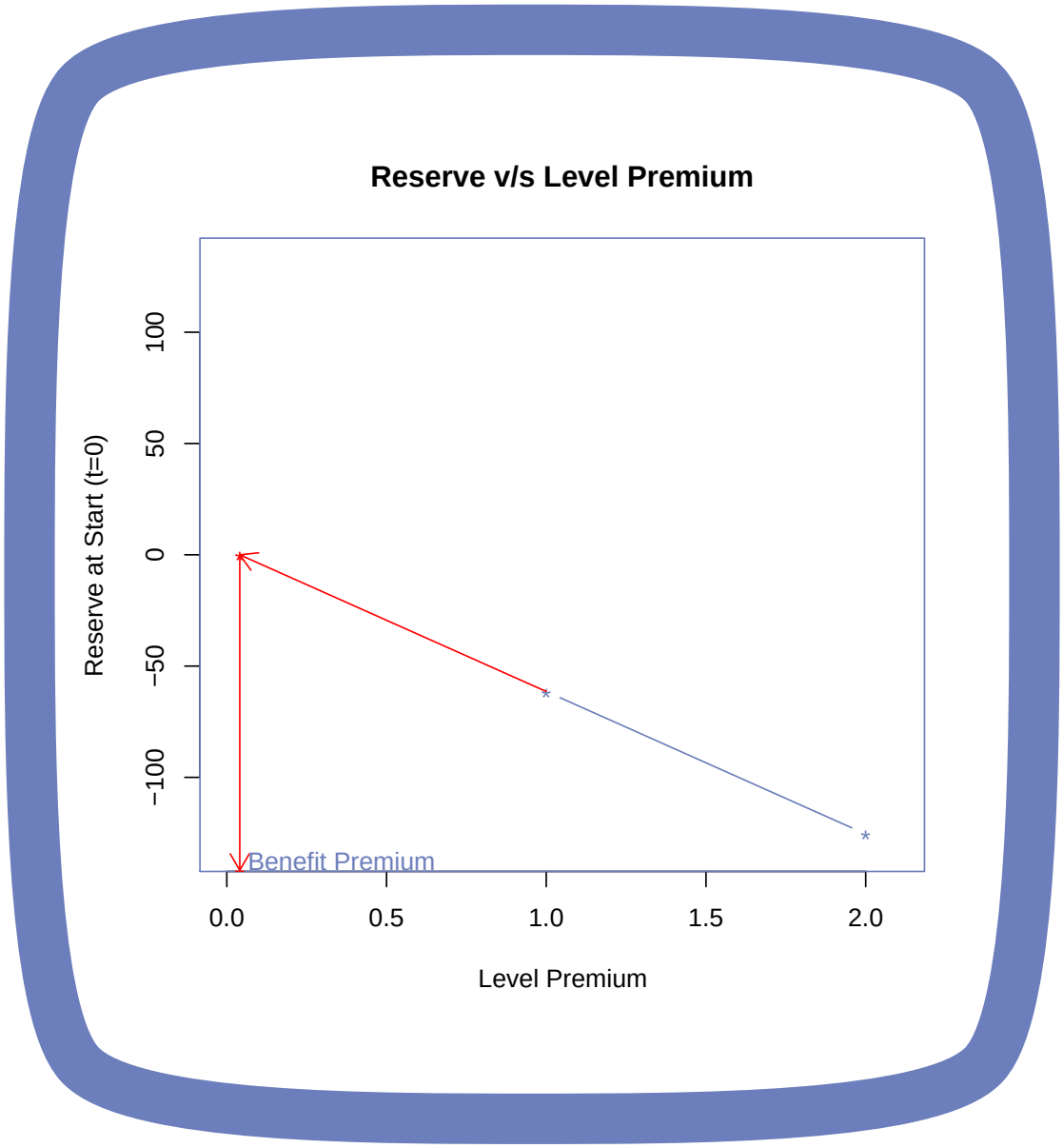


RCode

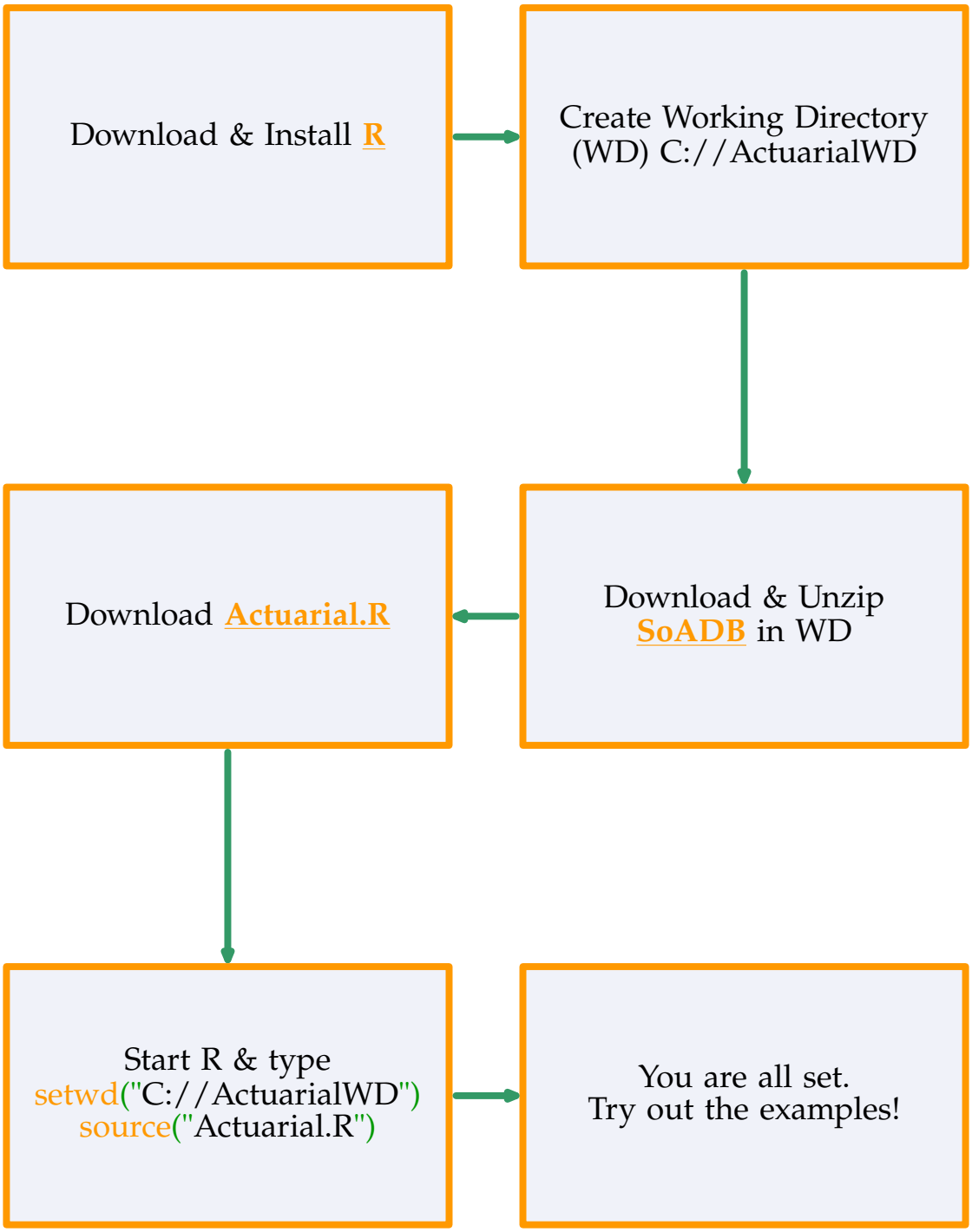
```
## Premium Calculation of a Constant Net Amount at Risk Insurance
## Concept - Reserve as a Linear function of the Premium

## Basic Data

    q<-Tb(15006)$Values$Aggregate;
    i<-.01;
    v<-1/(1+i); # For Presentation Purposes

## Calculations

    V2<-BLFOR(q*v-2,array(v,length(q)),0)[1];
    V1<-BLFOR(q*v-1,array(v,length(q)),0)[1];
    P<-(V2-2*V1)/(V2-V1);
    V<-BLFOR(q*v-P,array(v,length(q)),0);

## Premium via the Lottery Interpretation
## Notice the Weights of this Weighted Average

    CP<-sum(q*(v^(1:101)))/sum(v^(0:100));

## Graphing
    par(mfrow=c(1,1),col="#6C7EBB",lwd=1,lty=1,pch="*");
    plot(c(1,2),c(V1,V2),xlab="Level Premium",
         ylab="Reserve at Start (t=0)",xlim=c(0,2.1),
         ylim=c(V2*1.05,-V2*1.05),type="b",
         main="Reserve v/s Level Premium");
    arrows(1,V1,P,0,length=0.125,col="red");
    arrows(P,0,P,V2*1.13,length=0.125,col="red");
    points(P,0,col="red");
    points(P,V2*1.13,col="red");
    text(P+.025,V2*1.13,labels="Benefit Premium",adj=c(0,0));
```

Example 8 We consider here a Level NAR Whole Life Insurance. There are two concepts at play here:

i. That the Reserves as a function of the Level Premium is linear and this can be used to calculate the benefit premium
ii. The lottery interpretation of a Level NAR Insurance



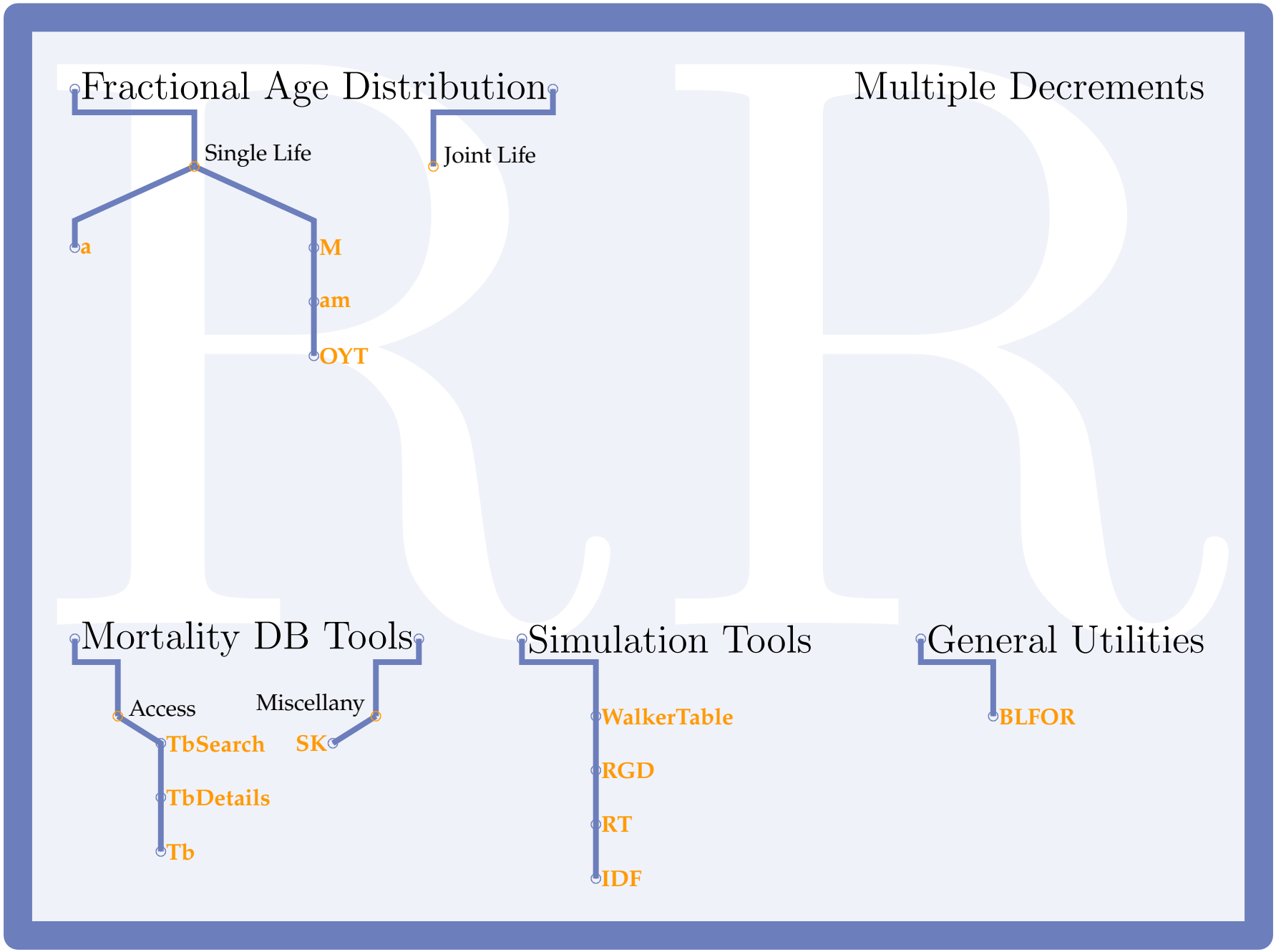
Installation



A Way to Get Started

5.1 Introduction 43
5.2 Mortality DB Tools 45
5.3 General Utilities 53
5.4 Fractional Age Distribution 55
5.5 Simulation Tools 65

The Manual


Fractional Age Distribution
Single Life
a
M
am
OYT
Joint Life
Multiple Decrements
Mortality DB Tools
Access
TbSearch
TbDetails
Tb
Miscellany
SK
Simulation Tools
WalkerTable
RGD
RT
IDF
General Utilities
BLFOR

5.1 Introduction

This *package* in R has been primarily developed to facilitate development of actuarial insight among actuarial science students through computing. It has been developed using the experience gained in developing and using an ActiveX/Excel Addin based application with similar, but a subset of the current set of, features. This is a work in progress. Rapid development is expected in the next few months with two choices for the final goal. First, end after implementation of sufficient tools - both on the life as well as the P&C side - to facilitate computing in undergraduate/graduate Actuarial Science courses. Second, concentrate only on the life side but go the full distance to implementing a professional pricing/risk management system. In any case, tools on the life side for didactic use will be completed before fall 2004.

The shift from the ActiveX/Excel platform to that of R needs explanation. As any two platforms there are pro's and con's - we feel that for our purposes the benefits of the move outweigh the drawbacks. First, on availability both score well - one can hardly doubt the ubiquity of Excel whereas R is open source. It should be added though that ActiveX ties you to the Windows operating system whereas R is available on many operating systems, for e.g. on both Linux and Windows. Second, the ease of implementation of the utilities is similar although R would outscore Excel, for e.g. the function integrate on R has been useful whereas we are unaware of any equivalents on Excel. One advantage of Excel over R is that it always keeps the numbers in sight and this has an accompanying disadvantage that one spends a lot of time formatting it, even if inadvertently. The latter is a significant disadvantage from the point of view of implementation. Third, the graphic capability of R surpasses that of Excel even though the latter is quite competitive. Fourth, Excel is definitely more user friendly but given that our software is meant for actuaries and actuarial students we feel that R will not lag far behind. Moreover, R is quite akin to APL which has been widely embraced by actuaries. Fifth, it is comparatively easy to use the R based package from Excel than it is vice-versa especially since ActiveX is a windows based technology and Excel is not zero-cost.

The figure on the left describes the package as it stands as of this writing. In what follows, we go from section to section describing the utilities under each part of the package - their raison d'être and how to put them to use them. In a later version, we will describe the mathematics behind their implementation.

5.2 Mortality DB Tools

This part of the package primarily deals with providing access to the tables stored in the SoA, binary format, database of tables. The database owes its existence to the superb effort put by the Computer Science Section of the SoA, the International Section of the SoA, and other volunteers. A caveat from the web site of the SoA - 'Users are advised to check the published reference given with each table before relying on the values. Many of the tables have not been double-checked. Discrepancies should be reported to webmaster@soa.org'.

5.2.1 SoA DB

In order to use the SoA DB one needs to download two files - tables.dat, the data file, and tables.ndx the index file. The package assumes that these are in the same directory as the package. A description of the storage format for these files can be found in the online help of TableManager, a software copyrighted by Steve Strommen, FSA, MAAA, and distributed freely by the SoA. We summarize the information contained therein below, even though the users need not be familiar with it. It is useful only for those who want to understand the code in-detail.

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 as shown below.

Figure 1 A Generic Record

There are a total of 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 is to be taken care of when the data is presented to the user as list as some of the records may be *empty*. The character strings neither have a terminating NULL byte (like C) nor a leading byte containing the length (like Pascal) - but note that the length can be read of the third and fourth bytes of the generic record. R as a platform is more convenient to read binary files compared to writing code in Visual Basic as in the latter one has to implement routines to convert bytes to IEEE floating point whereas in R it is implemented as part of the core support.

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

Record Types for tables.dat

Type	Content	Storage Format	Decoding Information
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

Figure 2 Record Types for tables.dat

tables.ndx

The primary purpose of this 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 the given table begins. This is a binary file with a sequence of fixed size records with an initial offset of 58 bytes (for some descriptive information). Each record has five fields as depicted in the figure below.

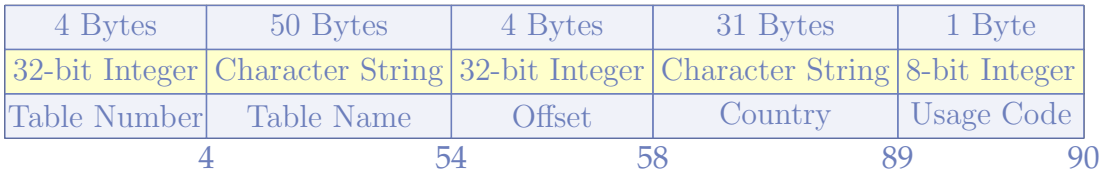


Figure 3 An Index File Record

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.

One can use the TableManager program to create many smaller data and their corresponding index files containing sub-groups of tables - but given the speed of access we feel that it is rather unnecessary.

5.2.2 Access to DB

TbSearch

Description:

'TbSearch' is used to search the SoA database for pertinent tables and their corresponding table numbers. The table numbers are used as arguments to the associated functions '**TbDetails**' and '**Tb**'.

Usage:

```
TbSearch("(Female|2000).{0,}(Female|2000)","Mexico","1");
TbSearch("[^ e](male|2000).{0,}[^ e](male|2000)","Mexico","1");
```

Arguments:

NameStr

This is a PERL regular expression which is matched against the 'Table Name' field of the records from the tables.ndx file.

CountryStr

This is a PERL regular expression which is matched against the 'Country of origin' field of the records from the tables.ndx file.

UsageStr

This is a PERL regular expression which is matched against the 'Usage Code' field of the records from the tables.ndx file.

Values:

'TbSearch' returns a data frame containing the fields 'Country of Origin', 'Usage Code', 'Table Name' and 'Table Number' of the records that match all the three regular expressions passed as the argument. In the case that the search results in zero records, a message conveying the same is returned instead of a data frame.

References:

`Perl 5.8.0 Documentation` : This and the links mentioned here are a good source to learn about the PERL version of regular expressions.

`SoA Database` : The best place to look for information regarding the SoA tables database is here.

See Also:

TbDetails and **Tb**.

Examples:

Example 1

In order to search for all Mexican female insured mortality tables of the year 2000 one could use the following command:
TbSearch("(Female|2000).{0,}(Female|2000)","Mexico","1");
And for Mexican male insured mortality tables of the year 2000 one could use the following command:
TbSearch("[^ e](male|2000).{0,}[^ e](male|2000)","Mexico","1");

TbDetails

Description:

'TbDetails' is used to find the details of a table in the SoA database, as stored in the file tables.ndx, given its table number. In particular, it is used by the function '**Tb**' to find the offset at which the data of a given table begins in the file tables.dat.

Usage:

```
TbDetails(15006);
```

Arguments:

TableNo

This is the table number of the desired table as defined by the SoA database.

Values:

'TbDetails' returns a list with the fields 'No', 'Offset', 'Usage', 'Name' and 'Country' respectively.

References:

`SoA Database` : The best place to look for information regarding the SoA tables database is here.

See Also:

TbSearch and **Tb**.

Examples:

Example 1

In order to find the details of the AMA-AMIS Mexican female insured mortality table of the year 2000 one could use the following command:

```
TbDetails(15007);
```

Tb

Description:

'Tb' is used to retrieve all the data stored on a certain table specified by its table number in the SoA database, as stored in the file tables.dat.

Usage:

```
q<-Tb(15006)$Values$Aggregate;
MinAge<-Tb(15006)$Values$Meta$MinAge;
q<-Tb(428)$Values$Select;
```

Arguments:

TableNo

This is the table number of the desired table as defined by the SoA database.

Values:

'Tb' returns a list of lists whose structure is depicted pictorially in **Figure 4**. The non-list field names correspond to those used in the record types of the file tables.dat, see **Figure 2**, excepting the floating point arrays which we describe below. In the case of a table of type 'A' or 'D', the array 'Aggregate' is an array of floating points with MaxAge - MinAge + 1 elements; in the case of type 'S', the array 'Select' is a two-dimensional array of floating points with dimension (MaxSAge - MinAge + 1) × SelPrd and the array 'Ultimate' is an array of floating points with MaxAge - MinAge - SelPrd + 1 elements. Given the above dimensions and that the above arrays are arranged in the natural order, the manner in which the data is stored in these arrays is obvious.

References:

`SoA Database` : The best place to look for information regarding the SoA tables database is here.

See Also:

TbSearch and **TbDetails**.

Examples:

Example 1

Here we define a function 'Aggregate' which for a given aggregate table, specified by its table number, and age, say x, returns q_x. Note that a well written such function must use error handlers - here we have refrained from doing so in the interest of maintaining clarity.

```
Aggregate<-function(TableNo,Age){
  Tb(TableNo)$Values$Aggregate[Age-Tb(TableNo)$Values$Meta$MinAge+1]
}
```

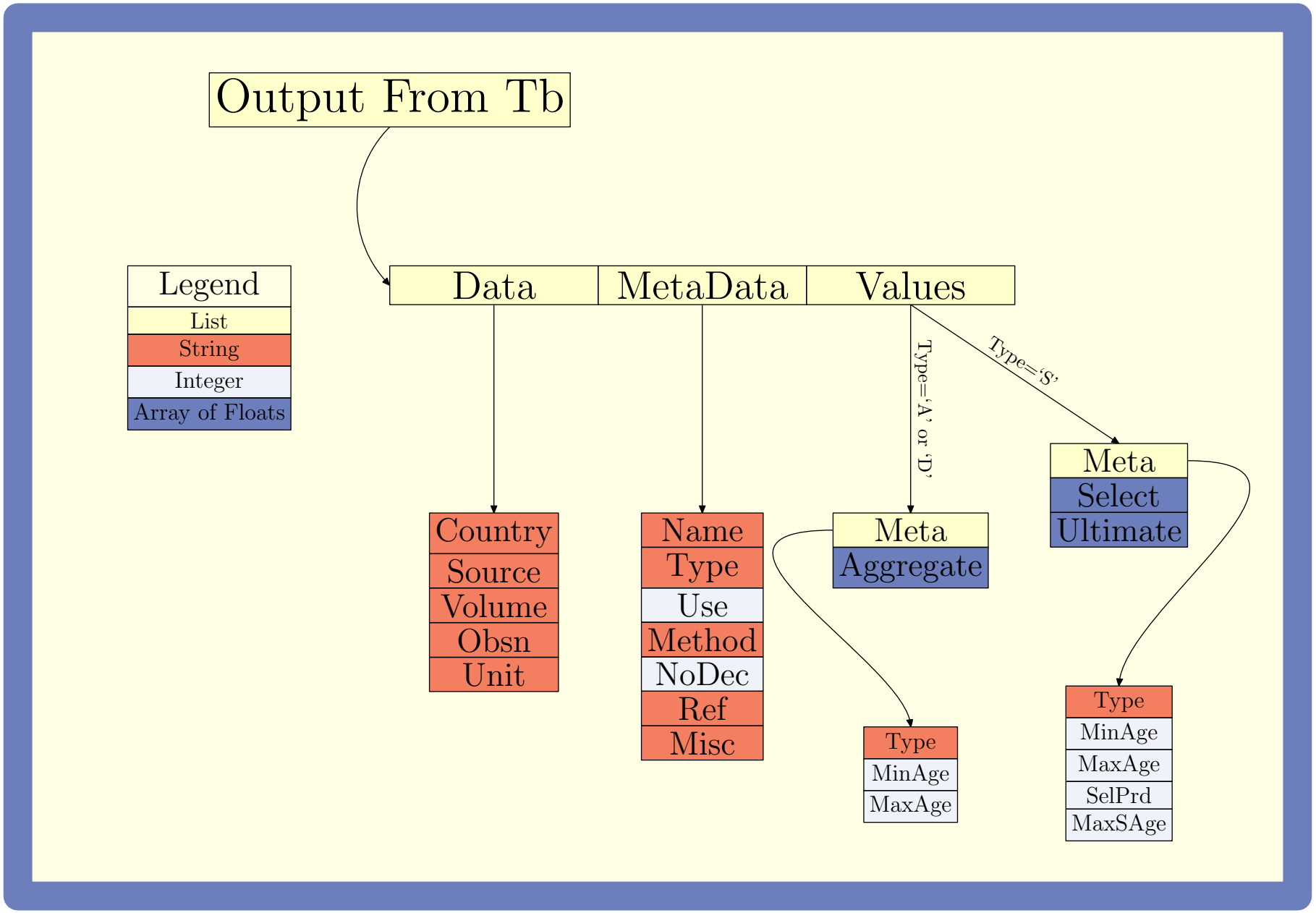


Figure 4 Structure of Tb Output

5.2.3 Miscellany

SK

Description:

'SK' is a simple function to arrive at

$$\left\{{}_k p_{x_i}\right\}_{(x_f - x_i + 1) \geq k \geq 0}$$

from

$$\left\{q_y\right\}_{x_f \geq y \geq x_i}$$

Usage:

```
p<-SK(q);
```

Arguments:

q

An array of mortality rates. Does not require that the terminal mortality rate is 1.

Values:

'SK' returns an array of floating points with a dimension of one more that the input (q) as described above.

References:

Any book on Actuarial Mathematics should be a good reference for the above actuarial notations; see for e.g.

Bowers, N., Gerber, H., Hickman, J., Jones, D., and Nesbitt, C. 1997. *Actuarial Mathematics*. Second Edition. Itasca, Ill.:Society of Actuaries.

Examples:

Example 1

The goal is to calculate the probability mass function ('f' below) of K(0) for the AMA-AMIS Mexican male insured mortality table of the year 2000.

```
q<-Tb(15006)$Values$Aggregate;
f<-SK(q)[1:101]*q;
```

5.3 General Utilities

BLFOR

Description:

'BLFOR' is a function which solves a **B**ackward **L**inear **F**irst **O**rder **R**ecurrence given the coefficients and the boundary condition (or value). In the following the arrays **r** and **s** and the real bv are such that the solution of the backward linear first order recurrence, **x** satisfies

$$x_n = r_n + s_n * x_{n+1}, \qquad 1 \le n \le \omega - 1, \qquad \text{with} \quad x_\omega = \text{bv}.$$

Usage:

BLFOR(q/1.1,(1-q)/1.1,1)

Arguments:

r

An array of reals representing the *constant* factor in the recurrence.

s

An array of reals representing the scale factor in the recurrence.

bv

A real representing the boundary value.

Values:

'BLFOR' returns an array of floating points with the same dimension as that of the input (**r**). Note that $x_{\omega+1}$ is not returned as it equals the given boundary value bv.

References:

Any book on Actuarial Mathematics should be a good reference for the above actuarial notations; see for e.g.

Bowers, N., Gerber, H., Hickman, J., Jones, D., and Nesbitt, C. 1997. *Actuarial Mathematics*. Second Edition. Itasca, Ill.:Society of Actuaries.

For recurrence relations, also see

Shyamal Kumar, N. D. 2003a. 'Recurrence Relations in Life Contingencies'. *Technical Report* No. DAS-03.10. Mexico:ITAM.

Examples:

Example 1

The goal is to calculate the array $\{A_x\}_{0\leq x\leq 100}$ ('WL' below) for the AMA-AMIS Mexican male insured mortality table of the year 2000 using an annual interest rate of 10%.

```
q<-Tb(15006)$Values$Aggregate;
WL<-BLFOR(q/1.1,(1-q)/1.1,1);
```

5.4 Fractional Age Distribution

5.4.1 Introduction

Mortality tables specify the distribution of the curtate future life time random variable. In order to arrive at the distribution of the complete future life time one either has to use company data or make a suitable assumption. This assumption, i.e. of the distribution of $T(x)$ given $K(x) = 0$ is known as the fractional age assumption and the distribution the fractional age distribution.

Fractional Age Assumptions

No	Name	Prob. Mass Fn. / Density Fn.
0	Degenerate at 0	$\Pr(\{0\}) = 1$
1	Degenerate at 1	$\Pr(\{1\}) = 1$
2	Uniform on $(0,1)$	$f(t) = \begin{cases} 1 & \text{if } 0 \le t \le 1; \\ 0 & \text{otherwise.} \end{cases}$
3	Exponential	$f_\lambda(t) = \begin{cases} \frac{\lambda \exp(-\lambda x)}{1-\exp(-\lambda)} & \text{if } 0 \le t \le 1; \\ 0 & \text{otherwise.} \end{cases}$, with $\lambda = -\log(p_x)$
4	Hyperbolic	$f_\alpha(t) = \begin{cases} \frac{\alpha}{(\alpha+(1-\alpha)*x)^2} & \text{if } 0 \le t \le 1; \\ 0 & \text{otherwise.} \end{cases}$, with $\alpha = p_x$

Figure 5 Fractional Age Distributions

The popular assumptions, sometimes for wrong reasons, are those of the continuous uniform random variable on the interval $(0,1)$, truncated (at 1) exponential distribution with parameter

$-\log(p_x)$ and the *Balducci* or hyperbolic assumption. We also consider the degenerate distributions at 0 and at 1 as these form two extremities and are useful for variational analysis. For purposes of the software, we have assigned a number to each of the above assumptions as shown in **Table 5**. The mentioned table also specifies the probability mass function or the density function for each of the assumptions. This is a probabilistic way at looking at the assumptions. An analytic way would be to look at the uniform, exponential and hyperbolic as linear, geometric or harmonic interpolation of the survival function to arrive at one which is continuous (and also differentiable).

Figure 6 exhibits an interesting property of the exponential and the hyperbolic distributions - both converge to the uniform when $q_x \to 0$ and both converge to the degenerate at 0 when $q_x \to 1$. Another interesting property, which follows easily from the analytic viewpoint, is that uniform, exponential and the hyperbolic are ordered in the first order for distributions in the descending order.

This section is about functions provided to facilitate computing of actuarial quantities that depend on the fractional age assumption. We will describe each such function in detail.

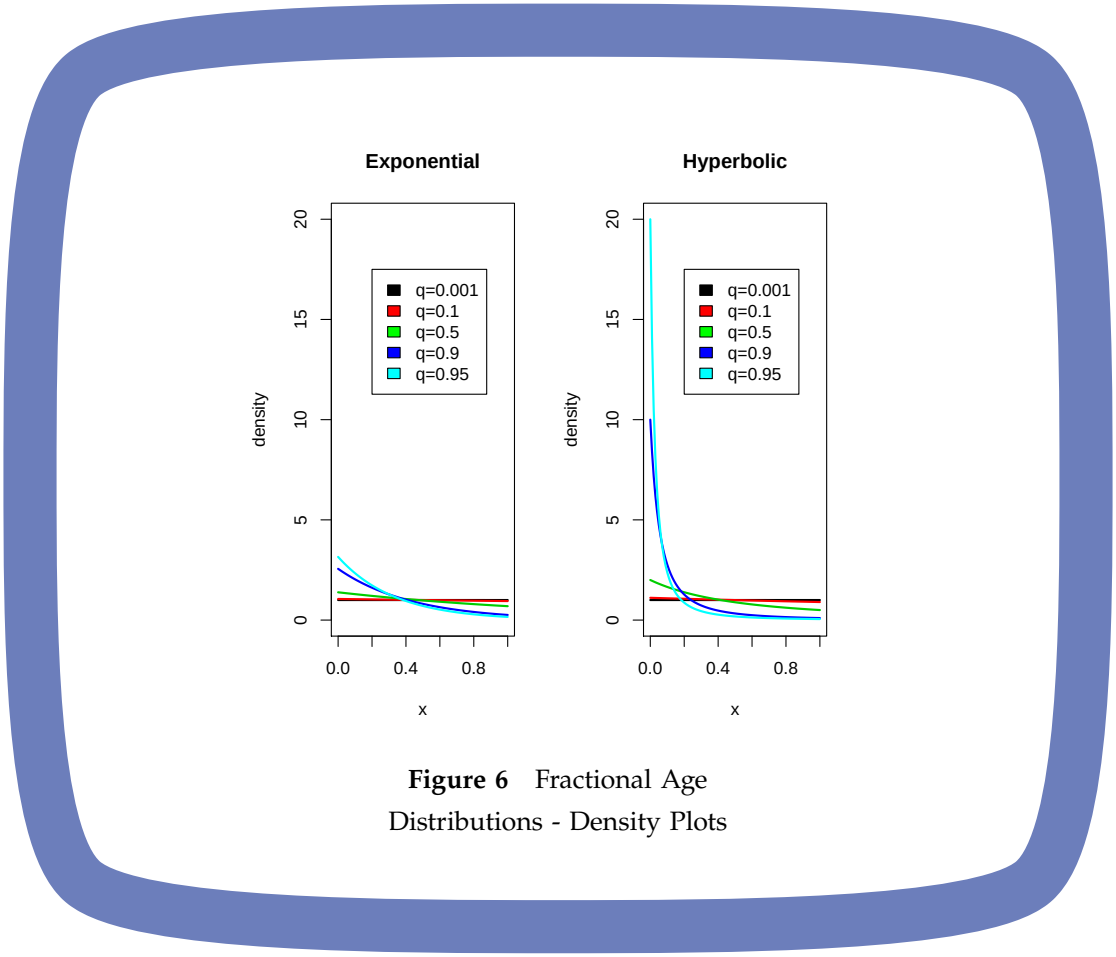


Figure 6 Fractional Age Distributions - Density Plots

5.4.2 The Functions

a

Description:

'a' is a function which for a given value of q_x and chosen fractional age distribution returns the value of

$$\mathbb{E}(T(x)|T(x) < 1).$$

A point to note is that for the degenerate distribution at 1 we have to interpret the above expectation in limit or as $\mathbb{E}(T(x)|T(x) \leq 1)$.

Usage:

```
ce1<-q*a(q,3)+(1-q);
```

Arguments:

q

An array of reals representing the mortality rates.

FracAgeAssmpn

The number of the fractional age assumption as appearing in the **Table 5**.

Details:

'a' is primarily useful in recurrence relations involving complete life expectations. A listing of the primary such recurrences is given in **Figure 7**

Values:

'a' returns an array of floating points with the same dimension as that of the input (**q**).

References:

Bowers, N., Gerber, H., Hickman, J., Jones, D., and Nesbitt, C. 1997. *Actuarial Mathematics*. Second Edition. Itasca, Ill.:Society of Actuaries.

Shyamal Kumar, N. D. 2003a. 'Recurrence Relations in Life Contingencies'. *Technical Report* No. DAS-03.10. Mexico:ITAM.

Examples:

Example 1

The goal is to calculate the array $\{\dot{e}_x\}_{0\le x\le 100}$ ('ce' below) for the AMA-AMIS Mexican male insured mortality table of the year 2000 using an annual interest rate of 10%.

```
q<-Tb(15006)$Values$Aggregate;
ce<-BLFOR(q*a(q,3)+(1-q),(1-q),1)
```

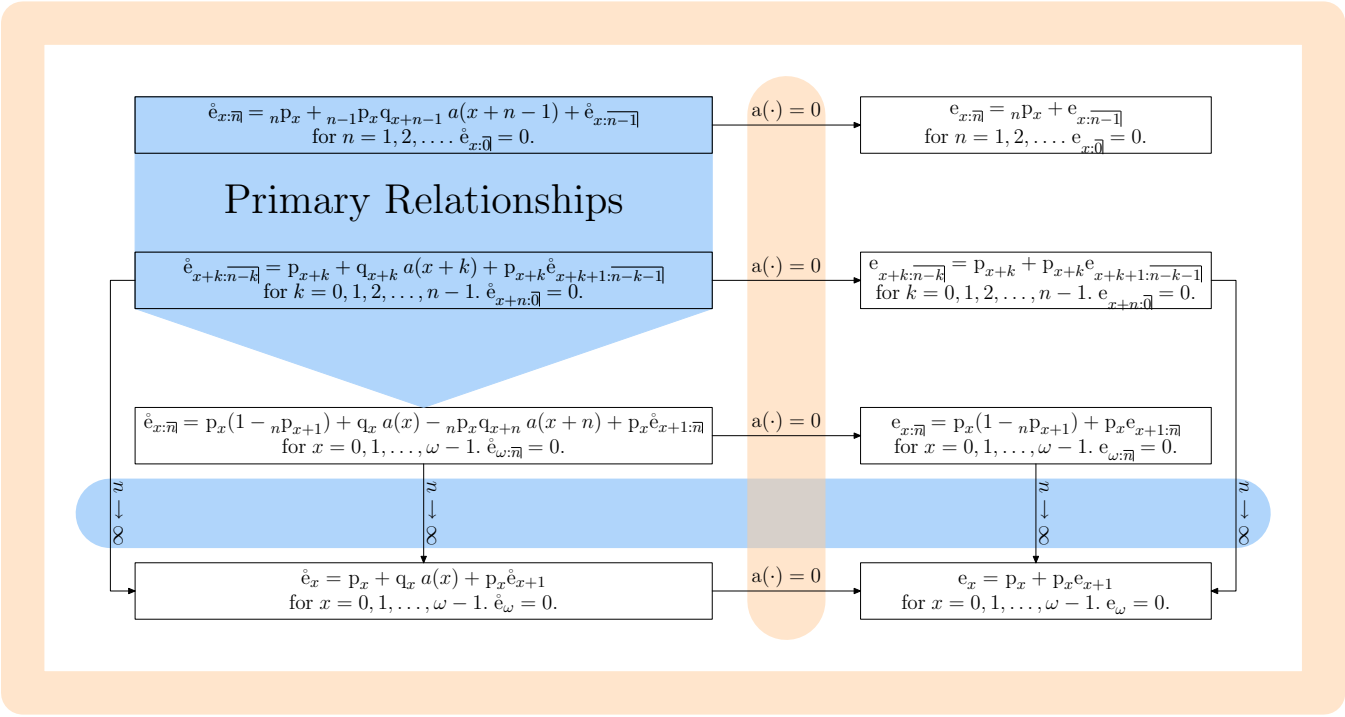


Figure 7 Expectation of Life

M

Description:

'M' is a function which for a given value of q_x, annual interest rate and chosen fractional age distribution returns the value of

$$\mathbb{E}(\exp(-\delta T(x))\,|T(x) < 1).$$

A point to note is that for the degenerate distribution at 1 we have to interpret the above expectation in limit or as $\mathbb{E}(\exp(-\delta T(x))\,|T(x) \le 1)$.

Usage:

```
A1<-q*M(q,3,0.1);
```

Arguments:

q

An array of reals representing the mortality rates.

FracAgeAssmpn

The number of the fractional age assumption as appearing in the **Table 5**.

i

The annual interest rate.

Details:

'M' is primarily useful in recurrence relations involving insurance products like whole life, term insurances and endowments. A listing of the primary such recurrences is given in **Figure 8** and **Figure 9**. The function $m_\delta(\cdot)$ in the figures is the same as 'M'. Also note that the mortality rate multiplied by the value of 'M' gives one the value of a one year term insurance.

Values:

'M' returns an array of floating points with the same dimension as that of the input (**q**).

References:

Bowers, N., Gerber, H., Hickman, J., Jones, D., and Nesbitt, C. 1997. *Actuarial Mathematics*. Second Edition. Itasca, Ill.:Society of Actuaries.

Shyamal Kumar, N. D. 2003a. 'Recurrence Relations in Life Contingencies'. *Technical Report* No. DAS-03.10. Mexico:ITAM.

Examples:

Example 1

The goal is to calculate the array $\{\bar{A}_x\}_{0\le x\le 100}$ ('A' below) for the AMA-AMIS Mexican male insured mortality table of the year 2000 using an annual interest rate of 10%. The fractional age assumption is that of constant force.

```
q<-Tb(15006)$Values$Aggregate;
i<-0.1;
A<-BLFOR(q*M(q,3,i),(1-q)/(1+i),1)
```

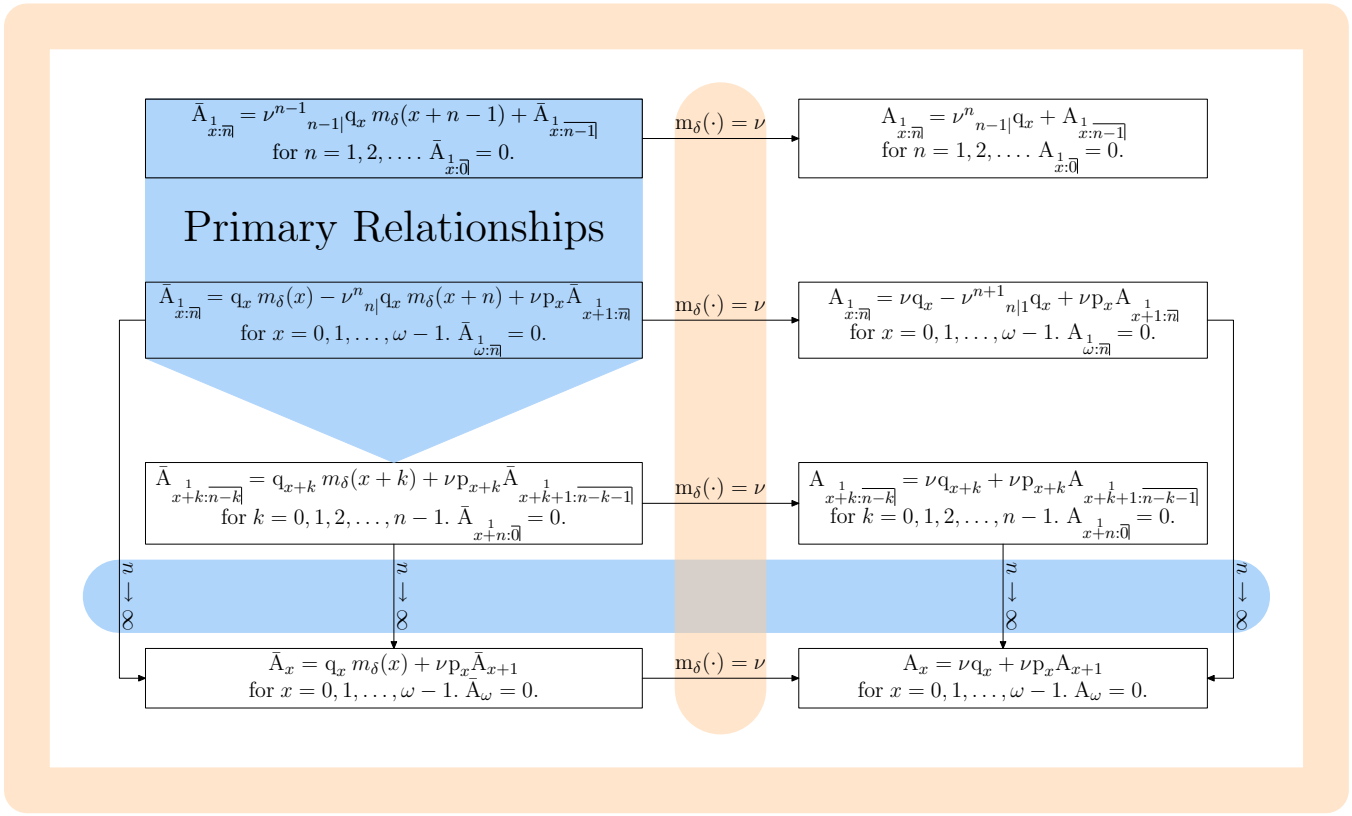


Figure 8 Term Insurances

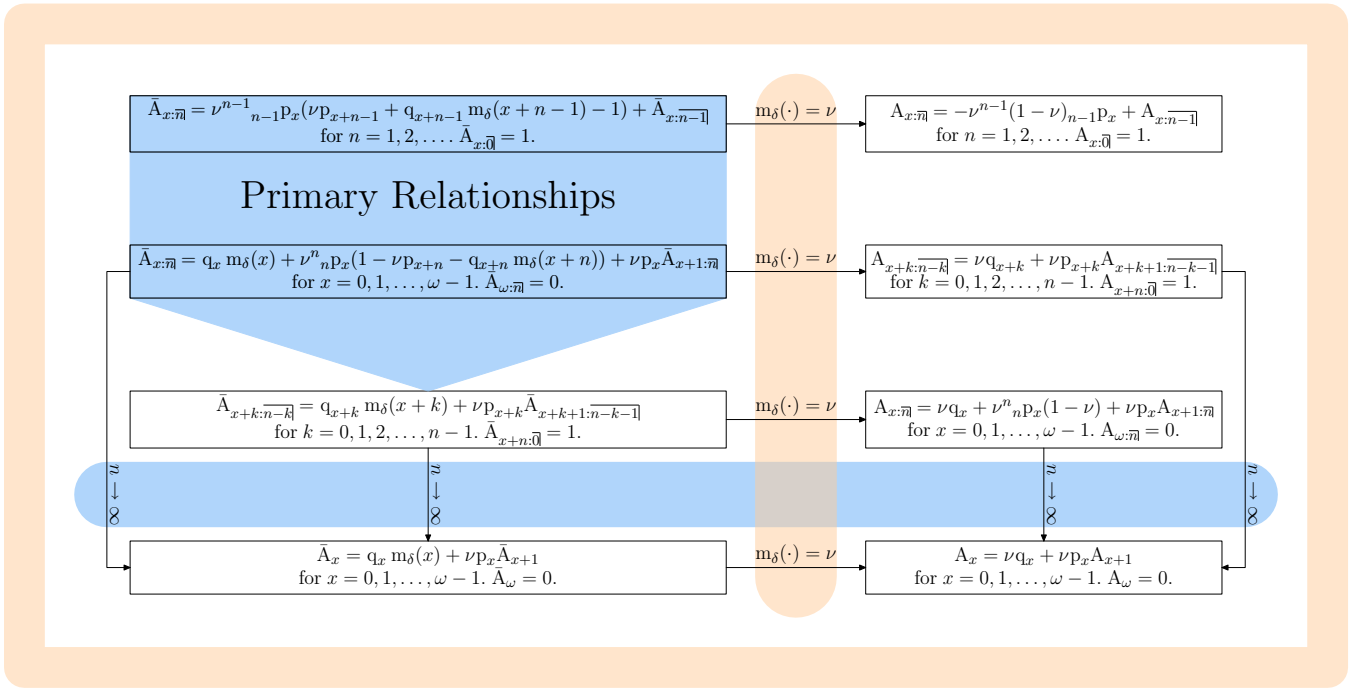


Figure 9 Endowments

am

Description:

'am' is a function which for a given value of q_x, number of fractional payments in a year (zero for continuous payment stream), annual interest rate and chosen fractional age distribution returns the value of a one year temporary life annuity.

Usage:

am(q,4,3,0.1);

Arguments:

q

An array of reals representing the mortality rates.

m

m value of zero signifies a continuous payment stream. An integral value signifies that there are *m* fractional payments each year.

FracAgeAssmpn

The number of the fractional age assumption as appearing in the **Table 5**.

i

The annual interest rate.

Details:

'am' is primarily useful in recurrence relations involving annuities though it is used by the function ''OYT''.

Values:

'am' returns an array of floating points with the same dimension as that of the input (**q**).

References:

Bowers, N., Gerber, H., Hickman, J., Jones, D., and Nesbitt, C. 1997. *Actuarial Mathematics*. Second Edition. Itasca, Ill.:Society of Actuaries.

Shyamal Kumar, N. D. 2003a. 'Recurrence Relations in Life Contingencies'. *Technical Report* No. DAS-03.10. Mexico:ITAM.

Shyamal Kumar, N. D. 2003b. 'Probabilistic Coupling in Life Contingencies'. *Technical Report* No. DAS-03.11. Mexico:ITAM.

Examples:

Example 1

The goal is to calculate the array $\{\ddot{a}_x^{(4)}\}_{0\le x\le 100}$ ('a4' below) for the AMA-AMIS Mexican male insured mortality table of the year 2000 using an annual interest rate of 10%. The fractional age assumption is that of the uniform distribution on the unit interval.

```
q<-Tb(15006)$Values$Aggregate;
i<-0.1;
a4<-BLFOR(am(q,4,2,i),(1-q)/(1+i),1)
```

OYT

Description:

'OYT' is a function which for a given value of q_x, annual interest rate and chosen fractional age distribution returns the value of a one year term insurance, i.e.

$$\mathbb{E}\Big(\exp\left(-\delta \mathrm{T}(x)\right) \mathrm{I}_{\{\mathrm{T}(x)<1\}}\Big).$$

A point to note is that for the degenerate distribution at 1 we have to interpret the above expectation in limit or as $\mathbb{E}\Big(\exp\left(-\delta \mathrm{T}(x)\right) \mathrm{I}_{\{\mathrm{T}(x)\le 1\}}\Big)$

Usage:

```
OYT(q,0,3,0.1);
```

Arguments:

q

An array of reals representing the mortality rates.

m

m value of zero signifies sum assured payable at the moment of death and a integral value signifies that the sum assured is payable at the end of the *m*-th period of death.

FracAgeAssmpn

The number of the fractional age assumption as appearing in the **Table 5**.

i

The annual interest rate.

Details:

'OYT' is primarily useful in recurrence relations involving insurance products like whole life, term insurances and endowments. A listing of the primary such recurrences is given in **Figure 8** and **Figure 9**. The function $m_\delta(\cdot)$ in the figures is the same as 'M'. When $m = 0$ one could easily use the function 'M' as the mortality rate multiplied by the value of 'M' is the value of a one year term insurance. That for $m > 1$ one has to resort to 'am' is the raison d'être. Note that the latter case is equivalent to assuming that the fractional age distribution is discrete uniform on the set $\left\{\frac{1}{m}, \frac{2}{m}, \ldots, \frac{m}{m}\right\}$.

Values:

'OYT' returns an array of floating points with the same dimension as that of the input (**q**).

References:

Bowers, N., Gerber, H., Hickman, J., Jones, D., and Nesbitt, C. 1997. *Actuarial Mathematics*. Second Edition. Itasca, Ill.:Society of Actuaries.

Shyamal Kumar, N. D. 2003a. 'Recurrence Relations in Life Contingencies'. *Technical Report* No. DAS-03.10. Mexico:ITAM.

Shyamal Kumar, N. D. 2003b. 'Probabilistic Coupling in Life Contingencies'. *Technical Report* No. DAS-03.11. Mexico:ITAM.

Examples:

Example 1

The goal is to calculate the array $\{\bar{A}_x^{(4)}\}_{0\le x\le 100}$ ('A4' below) for the AMA-AMIS Mexican male insured mortality table of the year 2000 using an annual interest rate of 10%. The fractional age assumption is that of the uniform distribution on the unit interval.

```
q<-Tb(15006)$Values$Aggregate;
i<-0.1;
A4<-BLFOR(OYT(q,4,2,i),(1-q)/(1+i),1)
```

5.5 Simulation Tools

5.5.1 Introduction

This section is about facilitating simulation of actuarial quantities. This is useful for e.g. as Actuarial present values can be estimated by sample mean of the present value random variable, variance of a loss random variable by its sample variance etc.. The basic building block then is one that generates random numbers from an arbitrary discrete distribution - this is implemented using the alias method of Walker (1977). Also provided are tools to generate T(x) for one of the five fractional age distributions mentioned in **Table 5**.

5.5.2 The Functions

WalkerTable

Description:

'WalkerTable' is a function which for a given discrete distribution constructs the table used for random number generation employing the alias method of Walker (1977). The table returned is actually a slight modification of that recommended by Walker (1977) - it can be found in Kronmal and Peterson (1979).

Usage:

WalkerTable(f);

Arguments:

f

An array of reals representing the probabilities (and hence summing to 1).

Details:

'WalkerTable' is primarily useful to generate the input required for the function **RGD** and is also used by the function **RT** internally.

Values:

'WalkerTable' returns a bi-dimensional array of floating points with the same number of rows as that of the input **f** and two columns.

References:

KRONMAL, R. A. and PETERSON, A. V. 1979. 'On the Alias Method for Generating Random Variables from a Discrete Distribution'. *American Statistician* 33,No. 4:214-218.

WALKER, A. J. 1977. 'An Efficient Method for Generating Discrete Random Variables with General Distributions'. *ACM Transactions on Mathematical Software* 3:253-256.

Examples:

Example 1

The goal is to simulate a 1000 realizations of K(35) for the AMA-AMIS Mexican male insured mortality table of the year 2000.

```
q<-Tb(15006)$Values$Aggregate[36:101];
f<-SK(q)[1:66]*q;
r<-RGD(WalkerTable(f),1000)-1
```

RGD

Description:

'RGD' is a function which for a given discrete distribution generates random number employing the alias method of Walker (1977). The table used for the generation is part of the arguments and can be generated using **WalkerTable**.

Usage:

```
RGD(T,k);
```

Arguments:

T

A bi-dimensional array of floating points with the same number of rows as the size of the support of the discrete distribution and can be generated using **WalkerTable**.

k

The desired size of the random vector.

Details:

'RGD' is useful to generate random variables from a general finite discrete distribution. The values are assumed to be the first n natural numbers - for a general support one could use an array to map from $1 : n$ to the support.

Values:

'RGD' returns an uni-dimensional array of floating points with the same number of rows as that of the input **k**.

References:

KRONMAL, R. A. and PETERSON, A. V. 1979. 'On the Alias Method for Generating Random Variables from a Discrete Distribution'. *American Statistician* 33,No. 4:214-218.

WALKER, A. J. 1977. 'An Efficient Method for Generating Discrete Random Variables with General Distributions'. *ACM Transactions on Mathematical Software* 3:253-256.

Examples:

Example 1

The goal is to simulate a 1000 realizations of K(35) for the AMA-AMIS Mexican male insured mortality table of the year 2000.

```
q<-Tb(15006)$Values$Aggregate[36:101];
f<-SK(q)[1:66]*q;
r<-RGD(WalkerTable(f),1000)-1
```

IDF

Description:

'IDF' is a function which provides the inverse distribution function for any of the fractional age distribution list in **Table 5**. It has obvious use for simulation.

Usage:

```
IDF(q,2,x);
```

Arguments:

q

The mortality rate for the interval under consideration. To generate T(x) one would input the value of $q_{K(x)}$, where K(x) would first be random generated.

FracAgeAssmpn

The reference number for the chosen fractional age assumption as given in **Table 5**.

x

The point at which the inverse is to be evaluated.

Details:

'IDF' is primarily useful to generate T(x). It is internally used by the function **RT**.

Values:

'IDF' returns a uni-dimensional array of floating points with the same number of rows as that of the input **x**.

References:

Bowers, N., Gerber, H., Hickman, J., Jones, D., and Nesbitt, C. 1997. *Actuarial Mathematics*. Second Edition. Itasca, Ill.:Society of Actuaries.

Examples:

Example 1

The goal is to simulate a 1000 realizations of T(35) for the AMA-AMIS Mexican male insured mortality table of the year 2000. The fractional age distribution is assumed to be that of the constant force. A more efficient way is to use the function **RT**.

```
q<-Tb(15006)$Values$Aggregate[36:101];
f<-SK(q)[1:66]*q;
r<-RGD(WalkerTable(f),1000)-1;
r<-r+unlist(lapply(q[r+1],g<-function(q){ IDF(q,3,runif(1))}));
```

RT

Description:

'RT' generates a random sample of T(x) given mortality rates and a fractional age distribution from **Table 5**.

Usage:

```
RT(q,3,1000);
```

Arguments:

q

The mortality rates array pertinent to the distribution of T(x).

FracAgeAssmpn

The reference number for the chosen fractional age assumption as given in **Table 5**.

k

The desired size of the random sample.

Details:

'RT' generates random sample of T(x) by using the alias method of Walker (1977). It internally uses the functions **WalkerTable** and **IDF**.

Values:

'RT' returns a uni-dimensional array of floating points containing the random sample.

References:

Bowers, N., Gerber, H., Hickman, J., Jones, D., and Nesbitt, C. 1997. *Actuarial Mathematics*. Second Edition. Itasca, Ill.:Society of Actuaries.

Kronmal, R. A. and Peterson, A. V. 1979. 'On the Alias Method for Generating Random Variables from a Discrete Distribution'. *American Statistician* 33,No. 4:214-218.

Walker, A. J. 1977. 'An Efficient Method for Generating Discrete Random Variables with General Distributions'. *ACM Transactions on Mathematical Software* 3:253-256.

Examples:

Example 1

The goal is to simulate a 1000 realizations of T(35) for the AMA-AMIS Mexican male insured mortality table of the year 2000. The fractional age distribution is assumed to be that of the constant force. This is an efficient way to do this even though there exist other ways (see Example 1 in the help section of **IDF**).

```
r<-RT(Tb(15006)$Values$Aggregate[36:101],3,1000);
```

6.1 Access to Mortality DB 73
6.2 Utilities for Mortality Tables 79
6.3 Tools for Simulation 81
6.4 Fractional Age Distribution Utilities 85
6.5 General Utilities 93

The Code

6.1 Access to Mortality DB

TbSearch

```
"TbSearch" <-
function (NameStr, CountryStr, UsageStr)
{
    zz <- file("tables.ndx", "rb")
    readChar(zz, 58)
    k <- 1
    for (i in 1:((file.info("tables.ndx")$size - 58)/90)) {
        TbNo = readBin(zz, integer(), size = 4)
        TbName = readChar(zz, 50)
        TbOffset = readBin(zz, integer(), size = 4)
        TbCountry = readChar(zz, 31)
        TbUsage = readBin(zz, integer(), size = 1)
        if (any(grep(NameStr, TbName, ignore.case = TRUE,
                perl = TRUE)) &&
            any(grep(CountryStr, TbCountry, ignore.case = TRUE,
                perl = TRUE)) &&
            any(grep(UsageStr, as.character(TbUsage),
                ignore.case = TRUE, perl = TRUE))) {
            if (k == 1) {
                No <- as.vector(TbNo)
                Name <- as.vector(TbName)
                Country <- as.vector(TbCountry)
                Usage <- as.vector(TbUsage)
            }
            else {
                No[k] = TbNo
                Name[k] = TbName
                Country[k] = TbCountry
                Usage[k] = TbUsage
            }
            k = k + 1
        }
    }
    close(zz)
```

TbSearch - Contd.

```
    if (k != 1) {
        dummy <- data.frame(No = No, Name = Name, Country = Country,
                            Usage = Usage)
        dummy <- dummy[order(dummy$Country, dummy$Usage, dummy$No),
                       , , ]
        dummy <- list(Country = dummy$Country, Usage = dummy$Usage,
                      Name = dummy$Name, No = dummy$No)
    }
    else {
        dummy <- list(Messages = "Search Resulted in Zero Entries")
    }
    data.frame(dummy)
}
```

TbDetails

```
"TbDetails" <-
function (TbSearch)
{
    zz <- file("tables.ndx", "rb")
    readChar(zz, 58)
    TbNo = 0
    while (TbNo != TbSearch) {
        TbNo = readBin(zz, integer(), size = 4)
        if (TbNo != TbSearch) {
            readChar(zz, 86)
        }
        else {
            TbName = readChar(zz, 50)
            TbOffset = readBin(zz, integer(), size = 4)
            TbCountry = readChar(zz, 31)
            TbUsage = readBin(zz, integer(), size = 1)
        }
    }
    close(zz)
    list(No = TbNo, Offset = TbOffset, Usage = TbUsage,
         Name = TbName, Country = TbCountry)
}
```

Tb

```
"Tb" <-
function (TbSearch)
{
    TbOffset <- TbDetails(TbSearch)["Offset"]
    RNo <- 0
    zz <- file("tables.dat", "rb")
    readChar(zz, TbOffset)
    while (RNo != 9999) {
        RNo = readBin(zz, integer(), size = 2)
        if (RNo != 9999) {
            RLen = readBin(zz, integer(), size = 2)
            if (RNo == 2) {
                TbNo = readBin(zz, integer(), size = 4)
            }
            else if ((RNo <= 11) || (RNo == 19)) {
                if (RNo <= 11) {
                  switch(RNo, TbName <- readChar(zz, RLen), ,
                    TbType <- readChar(zz, RLen),
                    TbContributor <- readChar(zz, RLen),
                    TbSource <- readChar(zz, RLen),
                    TbVolume <- readChar(zz, RLen),
                    TbObsn <- readChar(zz, RLen),
                    TbUnit <- readChar(zz, RLen),
                    TbMethod <- readChar(zz, RLen),
                    TbRef <- readChar(zz, RLen),
                    TbMisc <- readChar(zz, RLen))
                  }
                else {
                  TbCountry <- readChar(zz, RLen)
                }
            }
            else if (RNo <= 16) {
                switch((RNo - 11),
                       MinAge <- readBin(zz, integer(), size = 2),
                       MaxAge <- readBin(zz, integer(), size = 2),
                       SelPrd <- readBin(zz, integer(), size = 2),
                       MaxSAge <- readBin(zz, integer(), size = 2),
                       NoDec <- readBin(zz, integer(), = 2))
            }
```

Tb Contd.

```
            else if (RNo == 17) {
                qa <- as.array(readBin(zz, double(), n = RLen/8,
                  size = 8))
            }
            else if (RNo == 18) {
                TbHash = readBin(zz, integer(), size = 4)
            }
            else {
                TbUse = readBin(zz, integer(), size = 1)
            }
        }
        else {
            if (SelPrd > 0) {
                if (MaxSAge < MaxAge) {
                  qs <- t(array(qa[c(1:(SelPrd * (MaxSAge - MinAge +
                    1))) + trunc(c(0:(SelPrd * (MaxSAge - MinAge +
                    1) - 1))/SelPrd)], c(SelPrd, (MaxSAge - MinAge +
                    1))))
                  qu <- array(qa[c((SelPrd + 1) * c(1:(MaxSAge -
                    MinAge + 1)), (SelPrd + 1) * (MaxSAge - MinAge +
                    1) + c(1:(MaxAge - MaxSAge)))], c(MaxAge -
                    (MinAge + SelPrd) + 1))
                }
                else {
                  qs <- t(array(qa[c(1:(SelPrd * (MaxSAge - MinAge +
                    1))) + trunc(c(0:(SelPrd * (MaxSAge - MinAge +
                    1) - 1))/SelPrd)], c(SelPrd, (MaxSAge - MinAge +
                    1))))
                  qu <- array(qa[(SelPrd + 1) * c(1:(MaxSAge -
                    MinAge + 1))], c(MaxAge - (MinAge + SelPrd) +
                    1))
                }
            }
            else {
            }
        }
    }
```

Tb Contd.

```
    if (inherits(try(TbSource, TRUE), "try-error")) {
        TbSource <- "Missing Field"
    }
    if (inherits(try(TbVolume, TRUE), "try-error")) {
        TbVolume <- "Missing Field"
    }
    if (inherits(try(TbObsn, TRUE), "try-error")) {
        TbObsn <- "Missing Field"
    }
    if (inherits(try(TbUnit, TRUE), "try-error")) {
        TbUnit <- "Missing Field"
    }
    TbData <- list(Country = TbCountry, Source = TbSource,
            Volume = TbVolume, Obsn = TbObsn,
            Unit = TbUnit)
    TbMetaData <- list(Name = TbName, Type = TbType, Use = TbUse,
        Method = TbMethod, NoDec = NoDec, Ref = TbRef, Misc = TbMisc)
    if (TbType == "S") {
        TbValues <- list(Meta = list(Type = TbType, MinAge = MinAge,
            MaxAge = MaxAge, SelPrd = SelPrd, MaxSAge = MaxSAge),
            Select = qs, Ultimate = qu)
    }
    else {
        TbValues <- list(Meta = list(Type = TbType, MinAge = MinAge,
            MaxAge = MaxAge), Aggregate = qa)
    }
    close(zz)
    list(Data = TbData, MetaData = TbMetaData, Values = TbValues)
}
```

6.2 Utilities for Mortality Tables

SK

```
"SK" <-
function (q)
{
    temp <- 1
    dum <- function(x) {
        temp <<- (1 - x) * temp
    }
    unlist(lapply(c(0, q), FUN = dum))
}
```

6.3 Tools for Simulation

RGD

```
"RGD" <-
function (T, k)
{
    n <- length(T[, 1])
    r <- as.array(runif(k))
    for (l in 1:k) {
        u <- r[l]
        i <- ceiling(n * u)
        if (u >= T[i, 1]) {
            r[l] = i
        }
        else {
            r[l] = T[i, 2]
        }
    }
    r
}
```

WalkerTable

```
"WalkerTable" <-
function (f)
{
    F <- cbind(length(f) * f, 1:length(f))
    L <- array(0, length(f))
    while (length(F[(F[, 1] < 1) & (F[, 1] > 0), 1]) > 0) {
        j <- F[(F[, 1] < 1) & (F[, 1] > 0), 2][1]
        k <- F[F[, 1] >= 1, 2][1]
        L[j] = k
        F[k, 1] = F[k, 1] - 1 + F[j, 1]
        F[j, 1] = -F[j, 1]
    }
    cbind((F[, 2] - abs(F[, 1]))/length(f), L)
}
```

IDF

```
"IDF" <-
function (q, FracAgeAssmpn, x)
{
    if (FracAgeAssmpn == 0) {
        0
    }
    else if (FracAgeAssmpn == 1) {
        1
    }
    else if (FracAgeAssmpn == 2) {
        x
    }
    else if (FracAgeAssmpn == 3) {
        if (q < 1e-06) {
            x + 0.5 * q * x * (x - 1)
        }
        else if (q == 1) {
            0
        }
        else {
            log(1 - q * x)/log(1 - q)
        }
    }
    else {
        (1 - q) * x/(1 - q * x)
    }
}
```

RT

```
"RT" <-
function (q, FracAgeAssmpn, k)
{
    n <- length(q)
    T <- WalkerTable(SK(q)[1:n] * q)
    r <- as.array(runif(k))
    for (l in 1:k) {
        u <- r[l]
        i <- ceiling(n * u)
        if (u >= T[i, 1]) {
            r[l] = i - 1 + IDF(q[i], FracAgeAssmpn, (u - T[i,
                1])/(1 - T[i, 1]))
        }
        else {
            r[l] = T[i, 2] - 1 + IDF(q[T[i, 2]], FracAgeAssmpn,
                u/T[i, 1])
        }
    }
    r
}
```

6.4 Fractional Age Distribution Utilities

a

```
"a" <-
function (q, FracAgeAssmpn)
{
    if (FracAgeAssmpn == 0) {
        dum <- function(q) {
            0
        }
    }
    else if (FracAgeAssmpn == 1) {
        dum <- function(q) {
            1
        }
    }
    else if (FracAgeAssmpn == 2) {
        dum <- function(q) {
            0.5
        }
    }
    else if (FracAgeAssmpn == 3) {
        dum <- function(q) {
            if (q == 1) {
                Soln <- 0
            }
            else if (q < 1e-06) {
                Soln <- 0.5 - q/12
            }
            else {
                Soln <- -((1 - q)/q + 1/log(1 - q))
            }
            Soln
        }
    }
    else if (FracAgeAssmpn == 4) {
        dum <- function(q) {
            if (q == 1) {
                Soln <- 0
            }
```

a Contd.

```
            else if (q < 1e-06) {
                Soln <- 0.5 - q/6
            }
            else {
                Soln <- -((1 - q)/q^ 2) * (q + log(1 - q))
            }
            Soln
        }
    }
    else {
        dum <- function(q) {
            NA
        }
    }
    unlist(lapply(q, dum))
}
```

am

```
"am" <-
function (q, m, FracAgeAssmpn, i)
{
    delta <- log(1 + i)
    disc <- i/(1 + i)
    if (m != 0) {
        im <- m * ((1 + i)^ (1/m) - 1)
        dm <- m * im/(m + im)
    }
    if (FracAgeAssmpn == 0) {
        if (m == 0) {
            dum <- function(q) {
                (1 - q) * disc/delta
            }
        }
        else {
            dum <- function(q) {
                q/m + (1 - q) * disc/dm
            }
        }
    }
```

am Contd.

```
    else if (FracAgeAssmpn == 1) {
        if (m == 0) {
            dum <- function(q) {
                disc/delta
            }
        }
        else {
            dum <- function(q) {
                disc/dm
            }
        }
    }
    else if (FracAgeAssmpn == 2) {
        if (m == 0) {
            dum <- function(q) {
                q/delta * (1 - disc/delta) + (1 - q) * disc/delta
            }
        }
        else {
            dum <- function(q) {
                q/dm * (1 - disc/im) + (1 - q) * disc/dm
            }
        }
    }
    else if (FracAgeAssmpn == 3) {
        if (m == 0) {
            dum <- function(q) {
                Soln <- 0
                if ((1 - q)/(1 + i) != 1) {
                  Soln <- ((1 - q)/(1 + i) - 1)/(log((1 - q)/(1 +
                    i)))
                }
                Soln
            }
        }
```

am Contd.

```
        else {
            dum <- function(q) {
                Soln <- 1/m
                if ((1 - q)/(1 + i) != 1) {
                  Soln = (1 - (1 - q)/(1 + i))/(m * (1 - ((1 -
                    q)/(1 + i))^ (1/m)))
                }
                Soln
            }
        }
    }
    else if (FracAgeAssmpn == 4) {
        if (m == 0) {
            dum <- function(q) {
                if (q > 0.99999) {
                  if (q == 1) {
                    Soln <- 0
                  }
                  else {
                    f1 <- function(x) {
                      (1 - exp(-delta * x) - delta * x)/(1 -
                        q + q * x)^ 2
                    }
                    Soln <- integrate(f1, 0, 1)$value * (1 -
                      q)/delta - ((1 - q)/q^ 2) * (q + log(1 -
                      q))
                  }
                }
                else {
                  f2 <- function(x) {
                    exp(-delta * x)/(1 - q + q * x)^ 2
                  }
                  Soln <- (1 - (1 - q)/(1 + i) - q * (1 - q) *
                    integrate(f2, 0, 1)$value)/delta
                }
                Soln
            }
        }
```

am Contd.

```
        else {
            dum <- function(q) {
                Soln <- 1
                for (k in 1:(m - 1)) {
                  Soln = Soln + (1 - q)/((1 - (1 - k/m) * q) *
                    (1 + im/m)^ k)
                }
                Soln/m
            }
        }
    }
    else {
        dum <- function(q) {
            NA
        }
    }
    unlist(lapply(q, dum))
}
```

M

```
"M" <-
function (q, FracAgeAssmpn, i)
{
    if (FracAgeAssmpn == 0) {
        dum <- function(q) {
            1
        }
    }
    else if (FracAgeAssmpn == 1) {
        dum <- function(q) {
            1/(1 + i)
        }
    }
    else if (FracAgeAssmpn == 2) {
        if (i < 1e-06) {
            dum <- function(q) {
                1 - i/2
            }
        }
```

M Contd.

```
        else {
            dum <- function(q) {
                (1 - 1/(1 + i))/log(1 + i)
            }
        }
    }
    else if (FracAgeAssmpn == 3) {
        dum <- function(q) {
            if (q > 1e-06 && q != 1) {
                Soln <- -(log(1 - q) * (1 - (1 - q)/(1 + i)))/(q *
                  (-log(1 - q) + log(1 + i)))
            }
            else if (q < 1 - 0.999999 * (1 + i)) {
                Soln <- (1 + q/2)/(1.5 - 0.5 * (1 - q)/(1 + i))
            }
            else if (q < 1e-06) {
                Soln <- (1 + q/2) * (1 - (1 - q)/(1 + i))
            }
            else {
                Soln <- 1
            }
            Soln
        }
    }
    else if (FracAgeAssmpn == 4) {
        delta <- log(1 + i)
        dum <- function(q) {
            if (q > 0.99999) {
                if (q == 1) {
                  Soln <- 1
                }
                else {
                  f1 <- function(x) {
                    (exp(-delta * x) - 1 + delta * x)/(1 - q +
                      q * x)^ 2
                  }
                  Soln < 1 + integrate(f1, 0, 1)$value * (1 -
                    q) + delta * ((1 - q)/q^ 2) * (q + log(1 -
                    q))
                }
            }
```

M Contd.

```
            else {
                f2 <- function(x) {
                  exp(-delta * x)/(1 - q + q * x)^ 2
                }
                Soln <- (1 - q) * integrate(f2, 0, 1)$value
            }
            Soln
        }
    }
    else {
        dum <- function(q) {
            NA
        }
    }
    unlist(lapply(q, dum)) }
```

OYT

```
"OYT" <-
function (q, m, FracAgeAssmpn, i)
{
    if (m != 0) {
        dm <- m * (1 - (1 + i)^ (-1/m))
        Soln <- 1 - dm * am(q, m, FracAgeAssmpn, i) - (1 - q)/(1 +
            i)
    }
    else {
        Soln <- q * M(q, FracAgeAssmpn, i)
    }
    Soln
}
```

6.5 General Utilities

BLFOR

```
"BLFOR" <-
function (r, s, bv)
{
    temp <- bv
    dum <- function(x) {
        temp <<- x[1] + x[2] * temp
    }
    as.array(apply(cbind(r, s)[dim(r):1, ], 1, FUN = dum)[dim(r):1])
}
```