R for Actuarial Science Students

Arun Madappat
Diyana Najwa Nor Azmi
Jimmy Huang
Mentor: David Varodayan

Supported by the Society of Actuaries (SoA) Center of Actuarial Excellence (CAE) Grant

Table of Contents

Chapter 1:Basics of R & R in Actuarial Science

This chapter will focus on three of the many uses of R: data analysis, basic statistical calculations, as well as concepts dealing with the time value of money. All of these uses are highly relevant to those involved in the actuarial field. Being able to take a set of data and draw meaningful conclusions from it is an essential tool for an actuary to have. Calculating and understanding basic statistical derivations is also highly useful in the field of actuarial science. Finally, understanding the concepts surrounding the time value of money is not only useful for the FM exam, but also for working in the actuarial field.

Chapter 1.1: Data Analysis

One of the most basic tools we can use for analyzing a set of data is calculating the expected value, which is the long-run average for a set of data or formula.

To calculate the expected value E(x) of a dice roll, create the vector x with the possible values for the dice roll and the vector y for the corresponding probabilities of those values. Then, take the sum of the product of the two vectors:

```
> x<-c(1,2,3,4,5,6)

> y<-c((1/6),(1/6),(1/6),(1/6),(1/6),(1/6))

> sum(x*y)

[1] 3.5
```

For the expected value of any discrete probability function make a vector for x and f(x), then take the sum of the product of the 2 vectors

Calculate $E(x^2)$ for a dice roll

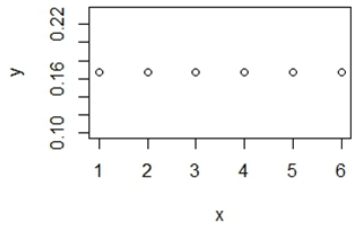
```
> x<-c(1:6)
> y<-c((1/6),(1/6),(1/6),(1/6),(1/6),(1/6))
> sum((x^2)*y)
[1] 15.16667
```

For $E(x^2)$ of any discrete probability function make a vector for x and f(x), then take the sum of the product x^2*y

Easy visualizations with R:

plot the PDF for a dice roll

```
> x<-c(1:6)
> y<-c((1/6),(1/6),(1/6),(1/6),(1/6),(1/6))
> plot(x,y)
```



More advanced data sets:

You can load in your own data set into Rstudio quite easily, tutorial here: https://www.youtube.com/watch?v=I1K3ZijJ3LM

Fortunately, R has a series of data sets preloaded into R for us to learn with. You can view them by typing data() into the console like so:

> data()

For the sake of this tutorial we will use the AirPassengers dataset which analyzes the number of air passengers by month from the year 1949 to 1960
This data set can be accessed by simply typing AirPassengers,
> AirPassengers

```
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949 112 118 132 129 121 135 148 148 136 119 104 118
1950 115 126 141 135 125 149 170 170 158 133 114 140
1951 145 150 178 163 172 178 199 199 184 162 146 166
1952 171 180 193 181 183 218 230 242 209 191 172 194
1953 196 196 236 235 229 243 264 272 237 211 180 201
1954 204 188 235 227 234 264 302 293 259 229 203 229
1955 242 233 267 269 270 315 364 347 312 274 237 278
1956 284 277 317 313 318 374 413 405 355 306 271 306
1957 315 301 356 348 355 422 465 467 404 347 305 336
1958 340 318 362 348 363 435 491 505 404 359 310 337
1959 360 342 406 396 420 472 548 559 463 407 362 405
1960 417 391 419 461 472 535 622 606 508 461 390 432
```

A quick analyses of your data can be done using the summary function like so:
> summary(AirPassengers)

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  104.0   180.0   265.5   280.3   360.5   622.0
```

A more advanced analyses can be done using the psych package which can be installed using the following command:

```
> install.packages("psych")
```

Then simply load the package using the library command:

```
> library(psych)
```

And finally use the following command to find a more meaningful analyses of your data

```
> describe(AirPassengers)
```

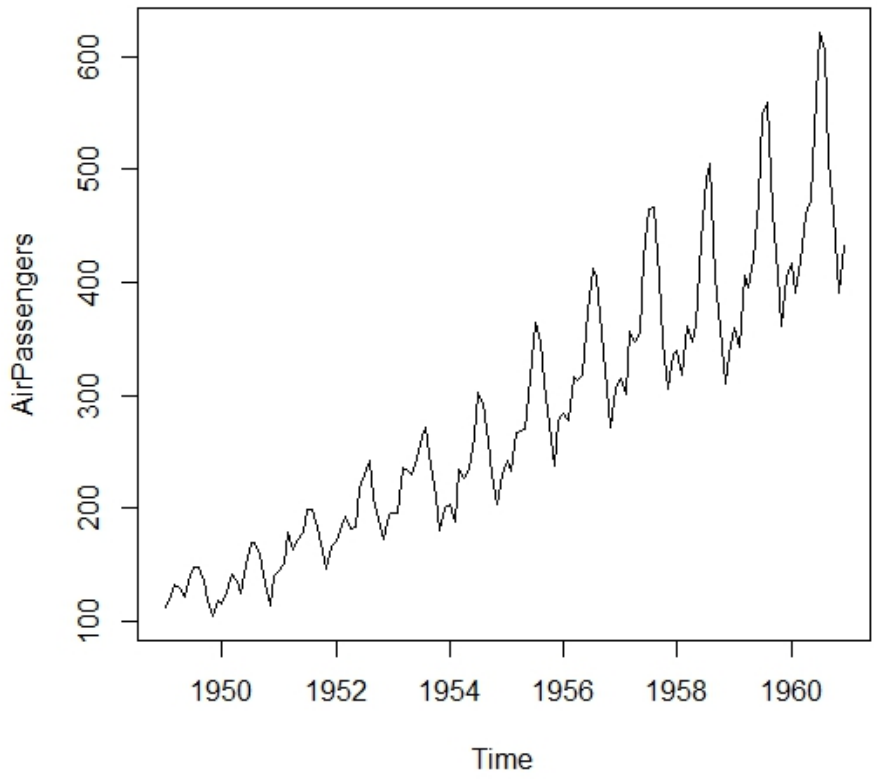
```
  vars   n  mean     sd median trimmed    mad min max range

1    1 144 280.3 119.97  265.5  271.45 133.43 104 622   518

  skew kurtosis se

1 0.57    -0.43 10
```

In order to graph this, we need to change this data frame into a matrix using the following command

```
> apm <- matrix(AirPassengers, ncol = 12, byrow =TRUE,  dimnames = list(
as.character(1949:1960),month.abb))
```

We can then make meaningful graphs such as air passengers vs year using

> plot(AirPassengers)



Chapter 1.2: Stats Package

Now that you've learned the basics of data analysis in R, it is time to learn about the effectiveness of the Stats package in solving many of the tedious calculations found in exam P.

First, let's learn how to calculate the variance of a vector with given data values.

> var(1:10)

[1] 9.166667

This code calculates the variance between a vector filled of values from 1 through 10.

Why is this code useful?
Quickly calculates the variance from a given vector of values
What does this code do?

Takes the given values and calculates the variance using the formula $\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2$
What does the result mean?
It is variance of the given vector values, how spread out the values are.

x=c(1,2,3,4,5)
assigns vector of 1 2 3 4 5 to x
y=c(2,4,6,8,10)
assigns vector of 2 4 6 8 10 to y

We can also find the covariance of 2 vectors using the cov function

Example:

```
> cov(x,y)
[1] 5
```

Why is this code useful?
Quickly calculates the covariance between vectors x and y
What does this code do?
Takes the given vector values for x and y and calculates the covariance using the formula E(xy)-E(x)E(y)

What does the result mean?
Covariance is a measure of the strength of the correlation between vectors x and y

To find the standard deviation of a given vector, we can either take the square root of the variance or use the sd function. In this example we will use the same vector y as above.

Example:

> sd(y)

[1] 3.162278

Why is this code useful?
Quickly calculates the standard deviation of vector y
What does this code do?
Takes the given vector values y and calculates the standard deviation using the formula

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

What does the result mean?
Standard Deviation is another measure of the spread of the values around the mean

We can also find the correlation coefficient between 2 vectors using the cor function as below. We will be using the same vector x as above.

Example:
> y=c(2,5,6,4,88)

> cor(x,y)

[1] 0.72132

Why is this code useful?
Quickly finds the correlation coefficient between x and the new vector y
What does this code do?
Takes the given vector values for x and y and calculates the correlation using the formula (Cov(x,y))/(std(x)*std(y))

What does the result mean?
Correlation describes how x and y are related. correlation coefficient of greater than 0 means x and y are directly related, less than 0 means inversely related, and 0 means not related.

These are some of the basic uses of the stats package that will help you to save some time when calculating these simple, but sometimes tedious calculations. To learn more about the various uses of the stats package visit the link below:

http://stat.ethz.ch/R-manual/R-patched/library/stats/html/00Index.html

Chapter 1.3: TVM Package

Now that you have learned some of the topics found in exam P and the basics of R, it is time to learn about how to master the Time Value of Money(TVM) package. This package deals with many of the concepts found in exam FM and can save time by solving complicated time value of money problems at the click of a mouse.

We will first discuss the cashflow function which can return the cashflows of a loan given the interest rate, maturity, loan amount and type.

How to use it:
cashflow(loan)
Example:

```
cashflow(loan(rate = 0.07, maturity = 30, amt = 1000, type = "bullet"))

 [1]   70   70   70   70   70   70   70   70   70   70   70   70   70   70   70

[16]   70   70   70   70   70   70   70   70   70   70   70   70   70   70 1070
```

Why is this code useful?
Quickly calculates the cashflows of a given loan
What does this code do?
Takes the given input values for the loan and creates a series of cashflows that describe the loan with these input values
What does the result mean?
These values represent all the payments to be made throughout the entire repayment of the loan. A bullet loan means all the principal is paid back on the last payment while all other payments represent interest on the loan.

Now, let's look at the cft function which can calculate the total financial cost of a loan, given the loan amount, maturity, interest rate, the fee the loan payer pays upfront, as well as the fee the loan payer pays per payment period.

How to use it:
cft(amt, maturity, rate, up_fee = 0, per_fee = 0)
Example:

```
cft(amt = 100, maturity = 10, rate = 0.05, up_fee = 1, per_fee = 0.1)

[1] 0.05363473
```

rate:effective interest rate, up_fee:fee that a loan taker takes at the beginning of the loan, per_fee:fee that the loan payer pays every period

Why is this code useful?

Quickly calculates the total financial cost of the loan including fees as well as interest

What does this code do?

Takes the given interest and fee values and determines the total financial cost of the loan.

What does the result mean?

The result represents the actual financial cost of the loan (interest+fees)

Another useful function TVM has to offer is the find_rate function which finds the effective rate of interest on a loan given a vector of discount factors.

How to use it:

find_rate(m, d, loan_type), m=maturity,d=discount factor vector

Example:

```
> find_rate(m = 3, d = c(0.99, 0.98, 0.97), loan_type = "bullet")

[1] 0.01020408
```

Why is this code useful?

Quickly calculates the effective interest rate being paid given a vector of discount factors

What does this code do?

Takes the discount values, loan type, and length to maturity and calculates the effective rate of interest being paid on the loan.

What does the result mean?

It is the effective rate of interest being paid on a loan that is being charged on a discount basis.

With TVM, you can also find the net present value of a series of cashflows given the interest rate, the amount of the cashflows, and when the cashflows are made.

How to use it:

Function: NPV (Net Present Value)

Example

```
> npv(i = 0.01, cf = c(-1, 0.5, 0.9), ts = c(0, 1, 3))

[1] 0.3685806
```

This code finds the net present value of cashflows of -1, 0.5, and 0.9 at times 0, 1, and 3 respectively at an interest rate of 1%

Why is this code useful?
Quickly calculates the Net Present Value of a series of cashflows at a given time at a given interest rate.

What does this code do?
Takes the given cashflows and discounts them at the effective interest rate over the time between the cashflow and the present.
What does the result mean?
The value represents the present value of all cash inflows and outflows summed together.

Using the pmt function, you can also find the payment amount that is necessary to cover the loan with a given interest rate and number of payment periods.

How to use it:
Function:PMT (Payment)
Example:

```
> pmt(amt = 100, maturity = 10, rate = 0.05)

[1] 12.95046
```

This code finds the payment for a $100 loan with 10 payments with a 5% interest rate between payments.

Why is this code useful?
Quickly calculates the scheduled payment for a loan with a given amount, interest rate, and term to maturity.
What does this code do?
Takes the given input values and calculates a payment to pay off the loan at the term to maturity while accounting for the given interest rate.
What does the result mean?
It is the scheduled payment that you would have to pay in order to pay off the loan by the term to maturity.

You can also find the interest rate on a loan given the number of payments, payment amount and loan amount using the rate function.

How to use it:
rate(loan amount, maturity, payment)
Example:

```
> rate(amt = 100, maturity = 10, pmt = 15)

[1] 0.08144239
```

This finds the interest rate on a $100 loan with 10 payments of $15.

Why is this code useful?
Quickly calculates the effective rate of interest on a loan
What does this code do?
Takes the given loan amount, term to maturity, and amount of payment and calculates the effective rate of interest that ensures that the loan is paid off in full at the last payment date.
What does the result mean?
The result represents the effective rate of interest being paid on a loan with these input values.

Now, with this guide you have been taught the absolute basics of data analysis, using the stats package, and understanding the concepts related to the time value of money using R. There is so much more you can learn with R in these topics or in any other topic that may interest you. Here are some links to learn more about these topics, as well as some of the other topics that R is useful for.

actuar Package with lots of actuarial science functionality:
http://cran.r-project.org/web/packages/actuar/actuar.pdf

A comprehensive guide to data analysis with R:
http://cran.r-project.org/doc/contrib/usingR.pdf

R tutorial that covers more comprehensive statistical usages of R as well as programming within R:
http://www.cyclismo.org/tutorial/R/

Chapter 2: Lifecontingencies Package

This chapter mainly covers the lifecontingencies package, which has many of the introductory functions that are used in the MLC or LC exams. It also contains various lifetables for countries such as France, Italy, the United States, and the SOA illustrative lifetable. This is relevant to actuarial science as it has some of the foundations for the MLC and LC exams. It makes it easier to work through extensive calculations on a lifetable as you can just have the machine calculate it. We will go through topics including:

Inputting lifetable
2.1 Basic Annuities
2.2 Basic Life Contingencies Functions
2.3 Miscellaneous Functions
2.4 SOA Illustrative Lifetable
2.5 Example Problem
2.6 Other Country's Lifetables
2.7 Creating an Actuarial Table
2.8 Increasing and Decreasing Life Insurances
2.9 Annuities Immediate and Due
2.10 Conclusion

Chapter 2.1: Inputting Life Table

One of the most fundamental functions of the lifecontingencies package is inputting a life table into R. There are two main classes that have been defined in this particular package, which are shown below. The actuarialtable is defined from the lifetable, but with the addition of interest. To get an even more detailed description of these two, the “showMethods” function offers more information.

Example

```
> showClass("lifetable")
Class "lifetable" [package "lifecontingencies"]

Slots:

Name:         x       lx      name
Class:   numeric   numeric character

Known Subclasses: "actuarialtable"

> showClass("actuarialtable")
Class "actuarialtable" [package "lifecontingencies"]

Slots:

Name:   interest         x       lx      name
Class:   numeric   numeric   numeric character

Extends: "lifetable"
```

In order to input the actual values of the lifetable into R, we use the following code as an example:

Example

```
> x_sample<-seq(from=0, to=9, by=1)
> lx_sample<-c(900, 800, 700, 600, 500, 400, 300, 200, 100, 0)
> lifetable<-new("lifetable", x=x_sample, lx=lx_sample, name = "lifetable")
removing NA and 0s
> print(lifetable)
Life table lifetable

  x  lx       px  ex
1 0 900 0.8888889 4.0
2 1 800 0.8750000 3.5
3 2 700 0.8571429 3.0
4 3 600 0.8333333 2.5
5 4 500 0.8000000 2.0
6 5 400 0.7500000 1.5
7 6 300 0.6666667 1.0
8 7 200 0.5000000 0.5
9 8 100 0.0000000 0.0
```

1. **Why is this code useful?**

This code allows you to input data into a lifetable, which will then allow you to perform a variety of other calculations. It also provides a check in that it won't allow you to input a value of lx that's higher than before because the probability of survival must decrease after each year and the values of x and lx must match.

2. **What does the code do?**

The first line of code creates the x values from 0 to 9 by 1's. The second line of code assigns the values of how many people are alive at each corresponding x term. This is labelled lx. The third line of code creates a lifetable titled, "lifetable" and generates the rest of the table seen. The print line just prints these results into the table shown at the bottom and outputs them.

3. **What does the result mean?**

This creates a life table where x is the year and lx is how many people out of 900 survive at that time in this case. Ex is the number of years they're expected to survive at that point in time and px is the probability they'll survive one year at age x.

Chapter 2.2: Basic Life Contingency Functions

After we've inputted a lifetable into R, we can begin to use the numbers in the table to run functions. Shown here are some of the most basic functions:

Example

```
> DemoEX1<-pxt(lifetable, 5,1)
> c(DemoEX1)
[1] 0.75
> DemoEX2<-qxt(lifetable, 5, 1)
> c(DemoEX2)
[1] 0.25
> DemoEX3<-exn(lifetable, 5, 1, "complete")
> c(DemoEX3)
[1] 0.875
> DemoEX4<-mxt(lifetable, 5, 1)
> c(DemoEX4)
[1] 0.2857143
```

1. **Why is this code useful?**

This table makes use of any lifetable you might've created and performs the basic life contingencies functions on it.

2. **What does the code do?**

The first line of code references the lifetable called, "lifetable" and puts it into a variable called, "DemoEX1". This specific example calculates 1p5 and the second line prints the value this generates. The variable called "Demo EX2" calculate 1q5, which is just 1-1p5 and "Demo EX3" calculates the expected lifetime between times 1 and 5, which is e angle 5:1. "Demo EX4" calculates 1m5, which is the average death rate experienced by the initial lives over the period 5 to 6 in this case.

3. **What does the result mean?**

This uses the data from the lifetable just generated. The first one calculates 1p5, the second one calculates 1q5, and the third one calculates e5:1 and the last one finds 1m5.

Chapter 2.3: Miscellaneous Functions

This is an example of another function possible through the lifecontingencies package. Specifically, the first block of code just calls upon the first lifetable we generated in chapter 2.1, then we create another lifetable and call upon the same functions to confirm the validity of the head and tail functions.

Example

```
> head(lifetable)
  x  lx
1 0 900
2 1 800
3 2 700
4 3 600
5 4 500
6 5 400
> tail(lifetable)
   x  lx
5  4 500
6  5 400
7  6 300
8  7 200
9  8 100
10 9   0
> summary(lifetable)
This is lifetable:  lifetable
 Omega age is:  9
 Expected curtated lifetime at birth is:  4

> x_sample<-seq(from=0, to=15, by = 1)
> lx_sample<-c(1500, 1400, 1300, 1200, 1100, 1000, 900, 800, 700, 600, 500, 400, 300, 200,
100, 0)
> lifetable<-new("lifetable", x=x_sample, lx=lx_sample, name = "lifetable")
removing NA and 0s
> print(lifetable)
Life table lifetable

    x   lx        px  ex
1   0 1500 0.9333333 7.0
2   1 1400 0.9285714 6.5
3   2 1300 0.9230769 6.0
4   3 1200 0.9166667 5.5
```

```
5   4 1100 0.9090909 5.0
6   5 1000 0.9000000 4.5
7   6  900 0.8888889 4.0
8   7  800 0.8750000 3.5
9   8  700 0.8571429 3.0
10  9  600 0.8333333 2.5
11 10  500 0.8000000 2.0
12 11  400 0.7500000 1.5
13 12  300 0.6666667 1.0
14 13  200 0.5000000 0.5
15 14  100 0.0000000 0.0

> head(lifetable)
  x   lx
1 0 1500
2 1 1400
3 2 1300
4 3 1200
5 4 1100
6 5 1000
> tail(lifetable)
    x  lx
11 10 500
12 11 400
13 12 300
14 13 200
15 14 100
16 15   0
```

Why is it useful?

The head and tail function invoke the first 6 values of the lifetable and the last 6 values of it respectively. If for some reason you needed just those values this function would do it for you.

What does the code do?

The head function takes the first 6 values of x and lx that you inputted and prints them and the tail function takes the last 6 values of x and lx that you printed and prints them. If there aren't 6 values total, then it just prints all of them as seen in the last life table I made.

The summary function gives you the limiting age, which is the longest anyone can survive and the expected curtate lifetime at birth.

What does the result mean?

This result gives you the first 6 values and the last 6 values through the head and tail functions and how long someone can survive and expected curtate lifetime at birth with the summary function on a specific lifetable.

Chapter 2.4: SOA Illustrative Lifetable

One of the more important and commonly referenced lifetables, especially while working through practice problems is the SOA illustative lifetable. Luckily, instead of having to input the entire thing, it is possible to just call upon it as all the values have already been inputted as shown below:

Example

```
> data("soa08Act")
> pxtLin<-pxt(soa08Act, 80, 0.5, "linear")
> c(pxtLin)
[1] 0.9598496
>
```

```
> pxtLin<-pxt(soa08Act, 80, 0.5, "linear")
> c(pxtLin)
[1] 0.9598496
> pxtConst<-pxt(soa08Act, 80, 0.5, "constant force")
> c(pxtConst)
[1] 0.9590094
```

1. **Why is this code useful?**

This code makes use of a previously created lifetable called soa08Act and it gives a more advanced function which allows for calculations of fractional survival probabilities. Soa08Act is the SOA illustrative life table at 6% interest.

2. **What does the code do?**

The first line calls for the dataset of the soa08Act and the second line calculates 0.5p80 and this is calculation through linear interpolation. This value is printed in the 3rd line. The next block of code calculates 0.5p80 through a constant force of mortality and prints it.

3. **What does the result mean?**

The first result is the calculation of 0.5p80 through linear interpolation and the second results is the calculation of 0.5p80 through a constant force of mortality based on the SOA lifetable at 6%.

Chapter 2.5: Miscellaneous Functions

This is a basic example problem that could confirm some of the various uses of the functions that we have worked through to this point.

Table 3.1 gives an extract from a life table. Calculate 5p0, q5, 5q0 and how many people died between time 0 and 3

Example

```
> x_problem<-seq(from = 0, to=9, by=1)
> lx_sample<-c(10000, 9965.22, 9927.12, 9885.35, 9839.55, 9789.29, 9734.12, 9673.56,
9607.07, 9534.08)
> lifetable<-new("lifetable",x=x_problem, lx=lx_sample, name = "lifetable")
> print(lifetable)

Life table lifetable

  x       lx        px        ex
1 0 10000.00 0.9965220 8.7955360
2 1  9965.22 0.9961767 7.8262336
3 2  9927.12 0.9957923 6.8562705
4 3  9885.35 0.9953669 5.8852413
5 4  9839.55 0.9948920 4.9126352
6 5  9789.29 0.9943642 3.9378576
7 6  9734.12 0.9937786 2.9601762
8 7  9673.56 0.9931266 1.9787079
9 8  9607.07 0.9924025 0.9924025

> problem1<-pxt(lifetable,0,5)
> c(problem1)
[1] 0.978929

> problem2<-qxt(lifetable,5,1)
> c(problem2)
[1] 0.005635751

> problem3<-qxt(lifetable, 0, 5)
> c(problem3)
[1] 0.021071
```

Just to confirm that this follows the rules of a life table:

```
> sum = problem1+ problem3
```

```
> c(sum)
[1] 1

> problem4<-dxt(lifetable, 0,3)
> c(problem4)
[1] 114.65
```
For people age 3, this is how many years they're expected to live out of the next 2
```
> curtate<-exn(lifetable,3,2)
> c(curtate)
[1] 1.985649
```

1. **Why is this code useful?**

This is an example problem that was introduced relatively early on in the MLC class. This code inputs the lifetable into R and calculates everything the problem asks for. This shows that while it may be some trouble learning these functions, it can make doing multiple calculations much easier.

2. **What does the code do?**

The first section inputs this lifetable that we're given into R as shown before. The second section calculates 5p0 and prints it, the third block calculates 1q5 and the fourth calculates 5q0. The next section is a reasonability check to make sure that 5p0 and 5q0 sum to be 1 and the section called problem 4 calculates how many people died from times 0 to 3.

3. **What does the result mean?**

This result just tells us what the problem asked and it's kind of a culmination of the things that have been shown so far but with more realistic numbers.

Chapter 2.6: Other Country's Lifetables

Here is some of the basic information for actual lifetables that can be referenced. This particular one uses the USA lifetable, but there is data for other countries, as mentioned below.

Example

```
> data("demoUsa")
> usaMale07<-demoUsa[,c("age", "USSS2007M")]
> names(usaMale07)<-c("x","lx")
> usaMale07Lt<-as(usaMale07,"lifetable")
> usaMale07Lt@name<-"USA MALES 2007"
> c(usaMale07)
$x
  [1]   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24
25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48
 [50]  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96
97
 [99]  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113

$lx
  [1] 100000  99262  99213  99182  99158  99138  99120  99104  99089  99075  99065
99056  99047  99032  99007  98968  98912  98841  98754  98654  98541  98414  98275
98128  97980  97834  97693  97555
 [29]  97419  97284  97147  97009  96868  96724  96576  96423  96264  96096  95919
95729  95525  95303  95062  94800  94517  94209  93875  93513  93120  92691  92224
91716  91165  90572  89940  89270
 [57]  88558  87800  86995  86138  85227  84254  83217  82111  80935  79684  78351
76929  75411  73792  72066  70223  68254  66161  63947  61612  59147  56545  53811
50951  47974  44882  41683  38401
 [85]  35063  31699  28341  25024  21785  18670  15722  12986  10501   8297   6393   4794
3496   2479   1711   1149    754    481    298    179    104     58     31     16      8      4      2
1
[113]     NA     NA
```

1. Why is this code useful?

You can also reference premade USA, Italian and Chinese lifetables for the data. The $x is the age and the $lx is how many people out of 100,000 would be alive in the United States. This might be useful if you wanted real life data about some topics to get a reasonability check on actual numbers. The table is formatted the same way it's just shown differently when printed and it caps out at 113 for the USA's.

2. What does the code do?

It shows the age of males in the USA in the year 2007 then prints the datatable.

3. What does the result mean?

This result shows how many many males in the US are expected to survive at each age with 100000 starting out at time 0. There are also datasets for the UK, China, Italy, Japan, France and Canada.

Chapter 2.7: Creating an Actuarial Table

This code creates an actuarial table that gives more output and allows you to input an interest rate as well.

Example

```
> x_sample<-seq(from=0, to=15, by = 1)
> lx_sample<-c(1500, 1400, 1300, 1200, 1100, 1000, 900, 800, 700, 600, 500, 400, 300, 200, 100, 0)
> lifetable<-new("lifetable", x=x_sample, lx=lx_sample, name = "lifetable")
removing NA and 0s
> exampleAct<-new("actuarialtable", x=x_sample, lx=lx_sample, interest = 0.03, name = "example actuarialtable")
removing NA and 0s
> c(exampleAct)
[[1]]
Actuarial table  example actuarialtable interest rate  3 %

    x   lx        Dx           Nx           Cx          Mx          Rx
1   0 1500 1500.00000 10513.08954 97.08738 1193.79351 8893.81309
2   1 1400 1359.22330  9013.08954 94.25959 1096.70613 7700.01959
3   2 1300 1225.37468  7653.86623 91.51417 1002.44654 6603.31346
4   3 1200 1098.16999  6428.49155 88.84870  910.93237 5600.86692
5   4 1100  977.33575  5330.32156 86.26088  822.08367 4689.93454
6   5 1000  862.60878  4352.98581 83.74843  735.82279 3867.85088
7   6  900  753.73583  3490.37702 81.30915  652.07436 3132.02809
8   7  800  650.47321  2736.64119 78.94092  570.76521 2479.95372
9   8  700  552.58646  2086.16798 76.64167  491.82429 1909.18851
10  9  600  459.85004  1533.58152 74.40939  415.18262 1417.36422
11 10  500  372.04696  1073.73148 72.24213  340.77323 1002.18160
12 11  400  288.96851   701.68452 70.13799  268.53110  661.40838
13 12  300  210.41396   412.71601 68.09513  198.39311  392.87728
14 13  200  136.19027   202.30205 66.11178  130.29798  194.48417
15 14  100   66.11178    66.11178 64.18619   64.18619   64.18619
16 15    0    0.00000     0.00000  0.00000    0.00000    0.00000
```

Why is this code useful?

This particular code creates an actuarialtable which is much more detailed than a lifetable. It calculates everything based on 3% interest and it just uses the same code from the lifetable.

What does this code do?

The first few lines just input the same data into the actuarialtable as before and the next lines create the table with 3% interest calling it example actuarialtable and putting it into the variable “exampleAct”, then prints it.

What does the result mean?

This prints out the information from the lifetable and the classical commutation functions. More information about these functions can be found on the bottom of page 29 from this link: http://cran.r-project.org/web/packages/lifecontingencies/vignettes/an_introduction_to_lifecontingencies_package.pdf

Chapter 2.8: Increasing and Decreasing Life Insurances

This code once again references a lifetable and performs slightly more advanced functions on it in the form of the increasing and decrease life insurances.

Example

```
> x_sample<-seq(from=0, to=15, by = 1)
> lx_sample<-c(1500, 1400, 1300, 1200, 1100, 1000, 900, 800, 700, 600, 500, 400, 300, 200, 100, 0)
> lifetable<-new("lifetable", x=x_sample, lx=lx_sample, name = "lifetable")
removing NA and 0s
> print(lifetable)
Life table lifetable

   x   lx        px  ex
1   0 1500 0.9333333 7.0
2   1 1400 0.9285714 6.5
3   2 1300 0.9230769 6.0
4   3 1200 0.9166667 5.5
5   4 1100 0.9090909 5.0
6   5 1000 0.9000000 4.5
7   6  900 0.8888889 4.0
8   7  800 0.8750000 3.5
9   8  700 0.8571429 3.0
10  9  600 0.8333333 2.5
11 10  500 0.8000000 2.0
12 11  400 0.7500000 1.5
13 12  300 0.6666667 1.0
14 13  200 0.5000000 0.5
15 14  100 0.0000000 0.0

> DemoEX6<-IAxn(lifetable, 5, 1, 6)
> c(DemoEX6)
[1] 0.01428571
> DemoEX7<-DAxn(lifetable, 5, 1, 6)
> c(DemoEX7)
[1] 0.01428571
```

Why is this code useful?

This is a slightly more complex function of increasing and decreasing life insurances.

What does this code do?

It first creates another lifetable, then assigns the increase and decreasing life insurances to the variables DemoEX6 and DemoEX7. The first field asks for the name of the lifetable you want to use and the second is the x value.The third is the n value and the fourth is the interest rate. You can also put in values for m, k, EV and power if required.

What does the result mean?

It simply outputs the values of these functions based on the lifetable and the values I put into the fields.

Chapter 2.9: Annuities Immediate and Due

This code simply calculates the values of increasing and decreasing annuities both due and immediate as show below:

Example

```
> 100*annuity(i = 0.05, n=6)
[1] 507.5692
> 100*accumulatedValue(i = 0.05, n=6)
[1] 680.1913
> iannuity<-annuity(i=0.03, n = 5, k=1, type = "immediate")
> dannuity<-annuity(i=0.03, n = 5, k = 12, type = "due")
> c(iannuity, dannuity)
[1] 4.579707 4.653791
> idannuity<-increasingAnnuity(i = 0.05, n = 5, type = "due")
> diannuity<-decreasingAnnuity(i=0.05, n = 5, type="immediate")
> c(idannuity, diannuity)
[1] 13.19471 13.41047
```

Why is this code useful?

This code allows you to easily calculate various annuities and shows what variables you need to input to calculate them.

What does this code do?

It calculates a regular annuity, and if due/immediate isn't specifically called, immediate is assumed. Next is just the accumulated value. Then iannuity is a normal annuity with 1 period a year and immediate while dannuity is monthly as k = 12 and the type is due. idannuity is an increasing and due annuity while diannuity is a decreasing and immediate annuity. Then these are printed.

What does the result mean?

It just gives the values of the functions that are called upon.

Chapter 2.10: Conclusion

Summary
This guide outlines some of the basic functions and uses of the lifecontingencies package in R. It gives examples of how each function should be used and how these functions can be used to solve actual example problems.

Links to additional resources
http://artax.karlin.mff.cuni.cz/r-help/library/lifecontingencies/html/00Index.html - this link has all of the functions

http://cran.r-project.org/web/packages/lifecontingencies/lifecontingencies.pdf - this link goes a bit more in depth about some of the functions

Chapter 3: R in Financial Economics

This chapter covers a few packages that are related to financial economics specifically fOptions, RquantLib, NMOF, VarSwapPrice, and AmericanCallOpt. This chapter especially focuses on some of the option pricing functions for American option and European option using two models; Black-Scholes model and Cox, Ross and Rubinstein binomial model. Also included in this chapter are functions to do quantitative analysis as well as step-by-step guide to use the function along with examples from MFE manual and Math 476 notes.

All the functions in this guide can be used to solve some of the basic financial economics related problems which are covered in the syllabus of Exam MFE/3F. Binomial model and Black-Scholes model are also relevant in the actuarial field particularly in risk management.

Below are the topics that will be covered in this chapter:

Chapter 3.1: Put-Call Parity

Package: NMOF

Usage
putCallParity(what, call, put, S, X, tau, r, q = 0, tauD = 0, D = 0)
where S=spot price, X=strike price, tau=time to maturity, r=interest rate, q=dividend rate, tauD=time to dividend, D=dividends

Example
Example 1.9 Chapter 1.4 from Math 476 notes

A non-dividend paying stock has a price of 40 today. A European call option allows buying the stock for 45 for the end of 9 months. The continuosly compounded risk-free rate is 5%. The premium of the call option is 2.84. Determine the premium of a European put option allowing selling the stock for 45 at the end of 9 months.

```
> putCallParity("put",call=2.84,S=40,X=45,tau=9/12,r=0.05,q=0,tauD=0,D=0)
```

```
[1] 6.183749
```

Why is this code useful?
It can quickly calculate the call premium or put premium based on the no-arbitrage assumption.
What does the code do?
The function calculates the put premium, given the values of call premium, spot price, strike price, time to maturity, and risk-free rate.
What does the result mean?
It is the price of a put option which has the same strike price and time to maturity with the call option.

Similarly, you can can calculate the value of one of the parameters by creating a function in RStudio.

Example:
Find the value of risk-free rate

You can create a function in RStudio that calculates the value of risk-free rate given the value of other parameters by typing the following codes in the editor window:

```
> riskfree <- function(putcalldiff,S,K,t){
  value <- c(1)
  value[1] <- (log((S-putcalldiff)/K))/(-t)

  value
   }
```

Save the script as a script file and then call the script in the console with the following command:
source('riskfree.R')

Example
Example from MFE manual

1. Consider a European call option and a European put option on a nondividend-paying stock. You are given:
(i) The current price of the stock is 60.
(ii) The call option currently sells for 0.15 more than the put option.
(iii) Both the call option and put option will expire in 4 years.
(iv) Both the call option and put option have a strike price of 70.

Calculate the continuously compounded risk-free interest rate.

```
> riskfree(0.15,60,70,4)

[1] 0.03916345
```

Chapter 3.2: American Call Option

The following function calculates the price of American call option without the values of annualized cost-of-carry rate and the dividend yield.

Package: AmericanCallOpt

Example:

```
> S<-100
> K<-100
> r<-0.03
> sigma<-0.4
> t<-0.5
> steps<-100
> call_price_am_bin<-am_call_bin(S, K, r, sigma, t, steps)
> call_price_am_bin

[1] 11.89466
```

Why is this code useful?
It can calculate the price of American call option with the most basic parameters using a binomial approximation model.

What does the code do?
The first six lines are basically assigning the values to the parameters. The last two lines is the command to calculate the price of the option.

What does the result mean?

The result is the price of American call option at time zero using a binomial approximation with 100 steps. The price is slightly higher than the price we get from AmericanOption function in the later chapter because dividend yield is not taken into account.

Chapter 3.3: European Call Option

This code specifically calculates the price of European call using binomial approximation.

Package: NMOF

Usage:
EuropeanCall(S0, X, r, tau, sigma, M = 101)

Example:

```
> EuropeanCall( S0 = 40, X = 40, r = 0.04, tau = 0.5, sigma = 0.30, M = 5)
[1] 3.922904

> h <- 1e-8
> C1 <- EuropeanCall(S0 = 40 + h, X = 40, r = 0.04, tau = 0.5, sigma=0.30, M=5)
> C2 <- EuropeanCall(S0 = 40, X = 40, r = 0.04, tau = 0.5, sigma=0.30, M=5)
> (C1-C2)/h
[1] 0.5833806
```

Why is this code useful?
This code allows you to quickly calculate the price of a European call option and the delta of the option.

What does the code do?
The first line of code basically assigns the values to the parameters and calculates the value of the call option. The last four lines calculate the value of the delta.

What does the result mean?

Delta is the rate of change in option price with respect to the change in the stock price. The result tells us that if the stock price change by any amount, the price of the call option will change by 0.58338 of that amount.

Chapter 3.4: American Option and European Option

The following codes are slightly more advanced as we can price both call and put option for American and European. The codes also return Greek letters for more in depth analysis.

Package: RQuantLib

Example:

```
> AmericanOption("call", 100, 100, 0.02, 0.03, 0.5, 0.4)
Concise summary of valuation for AmericanOption
  value   delta   gamma    vega   theta     rho  divRho
11.3648      NA      NA      NA      NA      NA      NA
```

You can change the first parameter to “put” to find the value of put option.

The code below evaluates the implied volatility for this option.

```
> callprice=11.3648
> AmericanOptionImpliedVolatility(type="call", value=callprice,
underlying=100,strike=100, dividendYield=0.02, riskFreeRate=0.03,maturity=0.5,
volatility=0.4)

$impliedVol
[1] 0.400253
```

Similarly, the following code also finds the value of implied volatility.

```
> impVol<-AmericanOptionImpliedVolatility("call", value=11.3648,
strike=100,volatility=0.4,100,0.02,0.03,0.5)
```

```
> print(impVol)

$impliedVol
[1] 0.400253
```

Why is this code useful?
Pricing a European and American options is one of the most basic things we learn in actuarial risk theory and it is also covered in Exam MFE. Options pricing is widely used to help investors anticipate price movements.

What does the code do?
The first line calculates the price of the American call option, given the value of strike price, spot price, dividend, risk-free rate, time to maturity, and volatility. The second line calculates the implied volatility for the option.

What does the result mean?
The value that we get from the first function is the price of the American call option. The implied volatility is the projected value of how far the price of the stock will move up or down in the future. We can calculate the value of the implied volatility based on the price of the option, strike price, spot price, dividend, risk-free rate, time to maturity, and the historical volatilty.

Similarly, you can simply use the following code calculates the value of call and put for European option.

Example:
```
> EuropeanOption(type="call", underlying=100, strike=100, dividendYield=0.01,
+ riskFreeRate=0.03, maturity=0.5, volatility=0.5)
Concise summary of valuation for EuropeanOption
   value    delta    gamma     vega    theta      rho   divRho
 14.3927   0.5783   0.0110  27.4848 -14.4673  21.7206 -28.9169
```

The result returns value of the European call option as well as the Greek letters. The Greek letters estimate the sensitivity of the option value with respect to other parameters; stock price, time to maturity, and volatility. For example, vega of 27.4848 simply means if the volatility of the stock price change by 1%, the price of the call option will change by 27.4848.

Chapter 3.5: Black-Scholes Model

This is the most basic function to quickly calculate values of call price and put price using Black-Scholes model.

Package: VarSwapPrice

Usage:
black_scholes(S, X, r, t, vol)
S = spot price
X = strike price
r = risk-free interest rate
t = time to maturity
v = volatility

Example:
> black_scholes(5, 5.50, 0.06, 1, 0.3)
$CallPrice
[1] 0.521205

$PutPrice
[1] 0.70091

Why is this code useful?
This code allows you to price both call and put option at the same time.
What does the code do?

It takes the value of all five variables and returns the value of call premium and put premium.

What does the result mean?

This is the price of the call option and put option based on all the assumptions under the Black-Scholes model.

Chapter 3.6: Cox, Ross, and Rubinstein Binomial Model

The following lines of code generally functions similarly like the previous codes but it goes more in depth of pricing the option with Cox, Ross, and Rubinstein binomial model.

Package: fOptions

Example:

This code calculates the value of European call option based on all the assumptions under binomial model.

```
> CRRBinomialTreeOption(TypeFlag = "ce", S = 100, X = 100, Time = 1, r = 0.1, b
= 0.1, sigma = 0.25, n = 10)
```

Output

```
Title:
 CRR Binomial Tree Option
Call:
 CRRBinomialTreeOption(TypeFlag = "ce", S = 100, X = 100, Time = 1,
      r = 0.1, b = 0.1, sigma = 0.25, n = 10)
Parameters:
      Value:
 TypeFlag ce
```

```
S        100
X        100
Time       1
r        0.1
b        0.1
sigma    0.25
n        10

Option Price:
 14.72619
```

If you wish to compare the price from using a Black-Scholes model, this code will do it for you.

```
> GBSOption(TypeFlag = "c", S = 100, X = 100, Time = 1, r = 0.1, b = 0.1, sigma =
0.25)
```

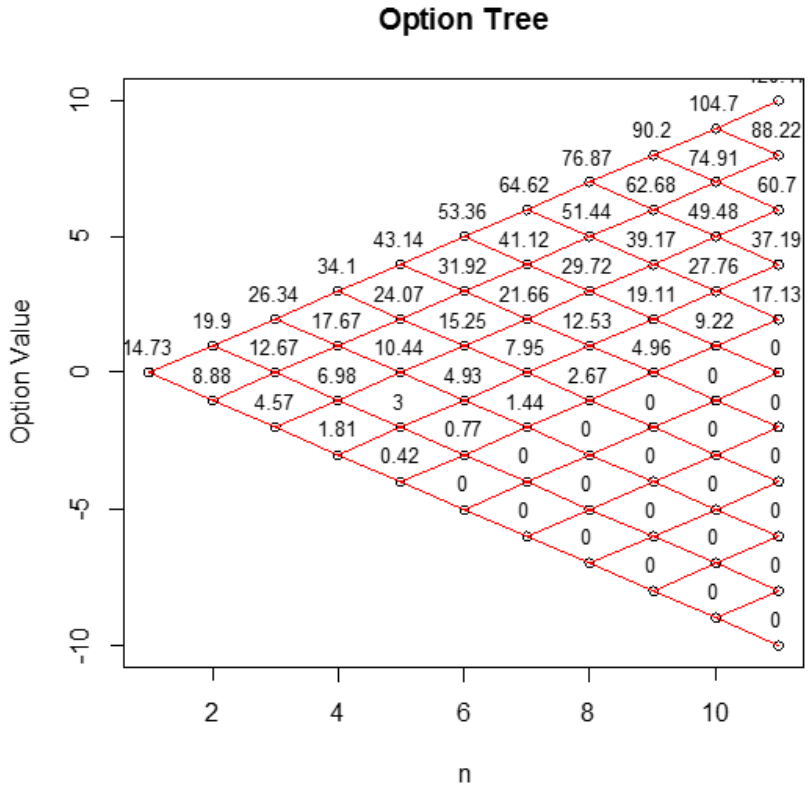
Output:

```
Title:
 Black Scholes Option Valuation
Call:
 GBSOption(TypeFlag = "c", S = 100, X = 100, Time = 1, r = 0.1, b = 0.1, sigma =
0.25)

Parameters:
        Value:
 TypeFlag c
 S        100
 X        100
 Time       1
 r        0.1
 b        0.1
 sigma    0.25

Option Price:
 14.9758
```

You can also plot the binomial tree with the following code and try using different number of steps to see different result.

```
> CRRTree = BinomialTreeOption(TypeFlag = "ce", S = 100, X = 100, Time = 1, r = 0.1, b = 0.1, sigma = 0.25, n = 10)
>  BinomialTreePlot(CRRTree, dy = 1, cex = 0.8, ylim = c(-10, 10),xlab = "n", ylab = "Option Value")
> title(main = "Option Tree")
```

Output:



Why is this code useful?
This code allows you to calculate the price of the European put and call options using a binomial model suggested by Cox, Ross and Rubinstein. We can use the model to break down the time to expiration into any number of intervals and are able to calculate the price of the option at the end of any period. CRR binomial model and Black-Scholes model are the two most frequently used pricing models. Both are in the syllabus of Exam MFE and Math 476. Since both models are based on the same theoretical assumptions, we can use the codes to compare the result that we get from using CRR model and Black-Scholes model.

What does the code do?
The first line is basically a function that we can use to compute the price of the European call option using a binomial model. Similarly, this function can also calculate price of European put, American call, and American put. The second line is the function to calculate price of the option using a Black-Scholes model. The third and fourth lines are the command to plot the binomial tree. The last line is the command to label the binomial tree.

What does the result mean?
Both models give the price for the European call option at time zero. Binomial model yields different result from Black-Scholes model because it takes into account all option prices at the end of every period while Black-Scholes only calculate the price at the expiration. However, as we increase the number of “n”, Binomial model and Black-Scholes model will essentially converge.

Chapter 3.7: Conclusion

This guide covers basic usage of fOptions, RquantLib, NMOF, VarSwapPrice and AmericanCallOpt packages as well as examples of how to use the functions to solve the problems. It also includes plotting of binomial trees from binomial model. Below are some of the useful links to additional resources:

Packages with detailed explanation and guide to other functions:
http://cran.r-project.org/web/packages/fOptions/fOptions.pdf
http://cran.r-project.org/web/packages/RQuantLib/RQuantLib.pdf
http://cran.r-project.org/web/packages/NMOF/NMOF.pdf
http://cran.r-project.org/web/packages/AmericanCallOpt/AmericanCallOpt.pdf
http://cran.r-project.org/web/packages/VarSwapPrice/VarSwapPrice.pdf