Central Washington University

CS 567

Computational Statistics

Life Insurance using R Technical Documentation

Chao Huang
41716747
Hermann Yepdjio
40917845
Lubna Alzamil
41442408

Instructor:
Dr.Donald Davendra

March 7, 2019

Contents

Listings

List of Figures

1 Introduction

This project implements the formulas used by life insurance companies using the R programming language version 3.5.1. Applying the equations into the computer will make it easier and more efficient to calculate and visualize several types of products offered by the insurance companies.

2 Overview

The data used is a mortality data taken from the Society of Actuaries Mortality Tables. The mortality data was downloaded and saved as a CSV file called compare.csv. All the equations were coded in an R file called life-table.R. The user can specify the inputs by editing life-table-inputs.txt. The life-table .R reads the data from the life-table-inputs.txt file and outputs the result. The project also simulates the production of life insurance policies for 10000 clients. The process takes place in project1-updated.R file. Additionally, the code calculates the profit and the fund value that the company earns depending on the interests and the benefits paid. Furthermore, the code generates plots of a variety of variables that will help the company to visualize the results of the calculations.

3 The Code

3.1 Life-table.R

```
1 startTime <- Sys.time()
2 setwd("/media/hermann/Tonpi/tonpi/Collegecourses/CWU/Graduate
      School/Winter 2019/CS 567/Projects/Project1/3R")
3 inputs <- read.table("life_table_inputs.txt", header = TRUE, sep = "\t",
      dec = ".", stringsAsFactors=FALSE) #read the inputs values from the
      life_table_inputs.txt file
4 inputMortalityFile <- (inputs[inputs$label == "inputMortalityFile",
      c("value")])
5 print (inputMortalityFile)
6 data <- read.csv(inputMortalityFile, header = TRUE,
      stringsAsFactors=FALSE) #avoid convert string to factor
7 dataTotalRow <- nrow(data)
8 beginRowData = 1
```

Listing 1: Reading The Data Part 1

In Listing 1, Line 1 stores the start time when the code is started to execute. At the end of code execution, the total time will be printed. Line 2 sets the working directory. Line 3 reads the inputs values from the life-table-inputs.txt file and store them in *inputs*. Line 4 gets the value of inputMortalityFile row from *inputs* data frame which is compare.csv and stores it in *inputMortalityFile*.Line 5 prints the value of *inputMortalityFile*. Line 6 reads the csv data of inputMortalityFile and store them in *data*. stringsAsFactors=FALSE means that do not covert the string data to factors. Line 7 counts the number of rows in *data*.

```
1 ages = as.numeric(data[beginRowData:dataTotalRow, 1])
2 mortality = as.numeric(data[beginRowData:dataTotalRow, 2])
3
4 lifeTable <- data.frame(ages,mortality)
5
6 lifeTableTotalRow = nrow(lifeTable)
7 lifeTable$t <- lifeTable$ages + 1
8 lifeTable$q <- lifeTable$mortality
9 lifeTable$p <- 1 - lifeTable$q
```

Listing 2: Reading The Data Part 2

in Listing 2, line 1 starts to loop inside the first column (ages column) of*data* data frame from 1 until the total rows of *data* then convert the ages into numeric values and store them in *ages*. Line 2 does the same with the second column ,which contains the mortality, storing them as numeric values in *mortality*. Line 4 creates a data frame called *lifeTable* which contains for now two columns *ages* and *mortality*. Line 6 counts the number of rows in *lifeTable* and store them in *lifeTableTotalRow*. Line 7 adds new column to the data frame *lifeTable* the column is called t and contains the values ages+1. Line 8 adds new column to the data frame *lifeTable* the column is called q and contains the same data as *mortality* column. Line 9 as well adds new column to the data frame *lifeTable* the column is called p and contains the values of 1-q.

```
1 #calculate v^t
2 interest <- as.numeric(inputs[inputs$label == "interest", c("value")])
3 lifeTable$vt <- 1/(1+interest)^lifeTable$t
4
5 #calculate t_p_x | x=0
6 lifeTable$t_p_x0[1] <- lifeTable$p[1]
7 for (i in 2:lifeTableTotalRow) {
8 lifeTable$t_p_x0[i] <- lifeTable$t_p_x0[i-1] * lifeTable$p[i]
9 }
```

Listing 3: Imeplemening The Equations Part 1

In Listing 3, lines 1 gets the value of interest from *inputs* and stores it in *interest*. Line 2 add new column to the data frame *lifeTable* the column is called v^t which contains the value of equation 1 . Lines 6-8 add new column to the data frame *lifeTable* the column is called ${}_tP_{x0}$ and contains the value of equation 2.

$$v^t = (1+i)^{-t} \tag{1}$$

$${}_tP_{x=0} =_1 P_{t-1} *_{t-1} P_{x=0} \tag{2}$$

```
1 #calculate t_E_x | x=0
2 lifeTable$t_E_x0 <- lifeTable$vt * lifeTable$t_p_x0
3
4 #calculate a_x
5 lifeTable$a_x[1] <- 1 + sum(lifeTable$t_E_x0)
6 for (i in 2:lifeTableTotalRow) {
7 lifeTable$a_x[i] <- 1 +
    sum(lifeTable$t_E_x0[i:lifeTableTotalRow])/lifeTable$t_E_x0[i-1]
8 }
9
10 #calculate A_x
11 d = interest/(1 + interest)
12 lifeTable$A_x <- 1 - d * lifeTable$a_x
```

Listing 4: Imeplemening The Equations Part 2

In Listing 4, line 2 adds new column to the data frame *lifeTable* the column is called ${}_tE_{x0}$ and contains the value ${}_tE_x = v^t * {}_tp_{x0}$. Lines 5-7 add new column to the data frame *lifeTable* the column is called ax and its values are $ax = 1 + \Sigma_t E_x$.The first

value in ax column is= $1 + \Sigma_t E_{x0}$. Lines 11-12 add new column to the data frame *lifeTable* the column is called A_x and contains the value of $A_x = 1 - d * ax$.

3.2 Projec1.R

In this file, we are calling the formulas located in life-table.R using the source function.

```
1  setwd(C://Desktop//Statis//GroupProject1//3R)
2  inputsProject1 <- read.delim("project1_inputs.txt", header = TRUE, sep =
       "\t", dec = ".", stringsAsFactors=FALSE) #read the inputs values from
       the project1_inputs.txt file
3  print (inputsProject1)
4  source("life_table.R")
5  #install library if not exist
6  if (!require(ggplot2)) install.packages('ggplot2')
7  if (!require(reshape)) install.packages('reshape')
8  library(ggplot2)
9  library(reshape)
10 x11()
11 myGraph <- ggplot(lifeTable, aes(ages, A_x))
12 myGraph <- myGraph + geom_point()
13 print(myGraph)
14 ggsave(filename = "images/A_x Vs Ages.png", plot = myGraph)
```

Listing 5: The Relationship between A_x and Ages

In Listing 5, line 1 sets the working directory to the path of where the project and needed files are stored . Line 2 reads the inputs values from the project1-inputs.txt file and stores them in *inputsProject1*. Line 3 prints the values in *inputsProject1*. Line 4 calls Life-table.R file that contains the necessary formulas. Line 5-6 install the needed packages and call them with library function . Here we are calling ggplot2 and reshape libries in lines 8-9. Reshape library is used to cast a data frame into the reshaped form. The library ggplot2 is used for creating plots. Line 10 opens a new graph window before creating a new graph to avoid overwriting of previous plotted graphs. X11() function is used in Unix platform, windows() is used in Windows platform and quartz() is used in Mac platform. Line 11 myGraph contains plots the *ages* and A_x cloumns from *lifeTable* data frame. Line 12 geom point function is used to create scatter points displaying the relationship between *ages* and A_x. Line 13 shows the plot on the screen. Line 14 saves the plot on images folder. Figure 1 shows the result of Line 12.

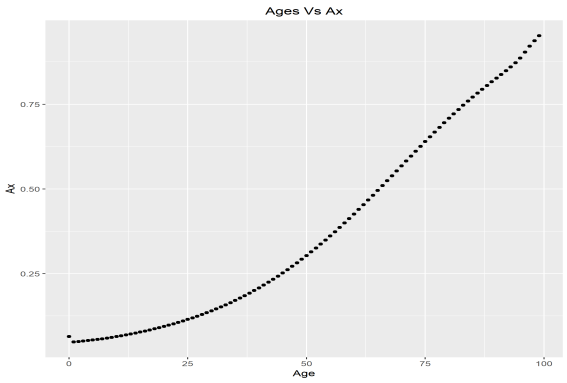


Figure 1: The Relationship between A_x and Ages

```
1 inputAges = as.numeric(inputsProject1[inputsProject1$label ==
    "inputAges", c("value")])
2 inputBenefit = as.numeric(inputsProject1[inputsProject1$label ==
    "inputBenefit", c("value")])
3 nsp <- lifeTable[lifeTable$ages == inputAges, c("A_x")]*inputBenefit
4 print(paste("input Ages: ", inputAges))
5 print(paste("Whole Life Net Single Premium: ", nsp))
```

Listing 6: Life Net Single Premium

In Listing 6, lines 1-2 get the values of inputAges and inputBenefit columns in *inputsProject1* convert them to numeric and store them in *inputAges* and *inputBenefit*. Line 3 extracts the corresponding A_x of the *inputAges* which is 12 and multiples it by the *inputBenefit* which is 1000 storing the result in *nsp*. The formula used is $NetSinglePremium = Benefit * A_x$. Line 4 prints the age. Line 5 prints the result of Net Single Premium.

```r
CalculateFinalAgeSemiPerfectData <- function(inputAge) { #---only for
    testing

#calculate probability of (x) lives t years t_p_x where x = inputAges
previousAgeP <- lifeTable[ages == (inputAge - 1), c("t_p_x0")]
if (inputAge > 0) {
pLives <- lifeTable[ages >= inputAge, c("t_p_x0")]/previousAgeP
} else {
pLives <- lifeTable[ages >= inputAge, c("t_p_x0")]
}

pLivesAges <- lifeTable[ages >= inputAge, c("ages")]
pCurveX <- data.frame(pLivesAges, pLives)

#calculate probability tl1_q_x that x survives t years and dies within 1
    year
pCurveXRow <- nrow(pCurveX)

#probability of dead based on input age
pCurveX$tl1_q_x[1] <- 1 - pCurveX$pLives[1] # firt data important,,, to
    avoid bug
for (j in 2:(pCurveXRow)) {

#small bug which does not consider the 0|1_q_x data..... also add q_x of
    1 in the last year automatically?
pCurveX$tl1_q_x[j] <- pCurveX$pLives[j-1] - pCurveX$pLives[j]
#pCurveX$tl1_q_x[j] <- pCurveX$pLives[j] - pCurveX$pLives[j+1] #i = u in
    the pdf 1 = t in pdf
}
#note: return a vector
bFAge <- sample(pCurveX$pLivesAges, inputNumberClients, replace = T,
    pCurveX$tl1_q_x)


return(bFAge)
}
```

Listing 7: Calculating ${}_tq_x$

Listing 7 calulates the probability of (x) lives t years using the defined function CalculateFinalAgeSemiPerfectData. It finds the probablity ${}_tp_{x0}$ of *inputAges - 1*

and stores the result in *previousAgeP*. Then checks whether the *inputAges* is greater than zero or not. If it is, it will get all the probabilities ${}_tp_{x0}$ of all ages and divide by *previousAgeP* and store the result in *pLives*. If the *inputAges* is not greater than zero, it will get all the probabilities ${}_tp_{x0}$ of all ages and store them in *pLives*. Then stores all the ages that are greater than or equal to *inputAges* in *pLivesAges*. After that, it creates a data frame called *pCurveX* that has two columns *pLivesAges* and *pLives*. After that, calculate ${}_{t1}q_x$ which is the probability that x survives t years and dies within 1 year. The general formula used to calculate this probability is ${}_{u|t}q_x = {}_u p_x$ - ${}_{u+t}p_x$. *pCurveXRow* stores the number of rows in *pCurveX* data frame. It iterates from 1 until *pCurveXRow*-1. Adds new column to the data frame *pCurveX* the column is called ${}_{t1}q_x$ and contains the values of the formula ${}_{u|t}q_x = {}_u p_x$ - ${}_{u+t}p_x$. Finally, it stores the ${}_{u|t}q_x$ of the last row in *pCurveX* data frame.

```
#now creating a block of 10000 people who are in different ages and want
    different benefits
BussinessBlock <- function(rAge,rBenefit) {
netSinglePremium <- lifeTable[lifeTable$ages == rAge, c("A_x")]*rBenefit
return(netSinglePremium)
}

inputNumberClients <- as.numeric(inputsProject1[inputsProject1$label ==
    "inputNumberClients", c("value")])

bAge <- vector(mode="integer", length=inputNumberClients) #initialize
    variables
bBen <- vector(mode="integer", length=inputNumberClients)
bNps <- vector(mode="numeric", length=inputNumberClients)
bFAge <- vector(mode="integer", length=inputNumberClients)
maxAges <- max(lifeTable$ages)
lifeTableAges <- lifeTable$ages[ages < maxAges]# avoid picking max ages

#looping 10000
lifeTableAges <- data.frame(Age=ages) # creating a data frame that
    contains the ages
print("---------Calculating Lifetimes----------")
print(paste("Number of clients: ", inputNumberClients))
lifeTableAsVector = as.vector(lifeTable[1,], mode = 'numeric') #create a
    vector representation of lifeTable so I can pass it to the c function
for (i in 2:lifeTableTotalRow)
{
lifeTableAsVector = c(lifeTableAsVector, as.vector(lifeTable[i,],
```

```
24 mode = 'numeric'))
25 }
26 dim = c(length(lifeTableAges$Age), 10, length(lifeTableAges$Age)) #bug if
       only put lifeTableAges -> length =1, 99 instead of 100
27 dyn.load("c_code.dll")
28 res = .C("for_loop", lifeTable=as.numeric(lifeTableAsVector), dim =
       as.integer(dim), bAge = as.numeric(bAge), bBen=as.integer(bBen), bNps
       = as.numeric(bNps), bFAge = as.numeric(bFAge), lifeTableAges =
       as.integer(lifeTableAges), inputNumberClients
       =as.integer(inputNumberClients))
29 bFAge = res[6]$bFAge
30 bAge = res[3]$bAge
31 bBen= res[4]$bBen
32 bNps= res[5]$bNps
```

Listing 8: 10000 Simulation

Listing 8 creates a function named ***BussinessBlock*** that receives two parameters -age and benefit- and calculates and returns the Net Single Premium according to the formula $NetSinglePremium = Benefit * A_x$. Line 7 gets the population value which is 10000 from *inputsProject1* and stores it in *inputNumberClients*. Lines 9-12 creates new variables (arrays) and set them to zeros with same length of *inputNumberClients* for later use. Line 11 gets the maximum age from *lifeTable* data frame and stores in *maxAges*. Line 14 stores all the ages that are less than *maxAges* in *lifeTableAges*. Line 17 stores all the ages in *lifeTableAges*. The loop is made in a c code and called within R to reduce the time needed to loop over all 10000 clients.

```
1 bDataFrame <- data.frame(Age = bAge, Benefit = bBen, NetSinglePremium =
      bNps, Die = bFAge) #Creating the final dataframe
2 bDataFrame$SurviveYears <- bDataFrame$Die - bDataFrame$Age
3 # creating the CSV file, each time we run the code new file will be
      created and the older file will be replaced
4 write.table(bDataFrame, file="BusinessData.csv", row.names=F,
      col.names=T, append=F, sep=",")
5 paymentTable <- bDataFrame[c("SurviveYears", "Benefit")]
6 paymentTable <- aggregate(. ~SurviveYears, data=paymentTable, sum, na.rm
      = TRUE)
7
8 investmentInterest <- as.numeric(inputsProject1[inputsProject1$label ==
      "investmentInterest", c("value")])
9
```

```
10 surviveYears <- 0
11 money <- sum(bDataFrame$NetSinglePremium)
12 earnInterest <- money*investmentInterest
13
14 benefitPayment <- 0
15 #find payment of the year
16 payment <- paymentTable[paymentTable$SurviveYears == 0, c("Benefit")]
17 if(length(payment) == 0){ #sometime there is no payment for specific
       year, so convert numeric(0) to 0
18 benefitPayment <- 0
19 }else{
20 benefitPayment <- payment
21 }
22 nYears <- max(paymentTable$SurviveYears)
23 for (i in 1:nYears) {
24 surviveYears[i+1] <- i
25 money[i+1] <- money[i] + earnInterest[i] - benefitPayment[i]
26 if(money[i+1] > 0){ #only calculate earnInterest when money > 0
27 earnInterest[i+1] <- money[i+1]*investmentInterest
28 }else {
29 earnInterest[i+1] <- 0
30 }
31
32 #find payment of next year
33 payment <- paymentTable[paymentTable$SurviveYears == i, c("Benefit")]
34 if(length(payment) == 0){ #sometime there is no payment for specific
       year, so convert numeric(0) to 0
35 benefitPayment[i+1] <- 0
36 }else{
37 benefitPayment[i+1] <- payment
38 }
39 }
40 #benefitPayment[i+1] <- 0 #last year payment = 0
41 profitTable <- data.frame(surviveYears, money, earnInterest,
       benefitPayment)
```

Listing 9: Business Data Frame

In Listing 9, line 1 creates a data frame named *bDataFrame* that contains Age, Benefit, NetSinglePremium and Die columns. Line 2 adds new column to *bDataFrame* named *SurviveYears* that contains the difference between Die and Age column. Line 4 creates the CSV file, each time we run the code new file will be created and the older file will be replaced. Lines 5-6 creates a data frame named *paymentTable* that

contains SurviveYears and Benefit columns from *bDataFrame* data frame. After that, we are gonna calculate and see whether the insurance company makes money or loses money with interest rate of 5.00%. Line 8 gets the interest value which is 0.05 from *inputsProject1* and stores it in *investmentInterest*. Line 10 sets the SurviveYear to zero which indicates the current year. Line 11 calculates the sum of NetSinglePremium column in *bDataFrame* data frame and stores the result in *money*. Line 12 multiplies *money* with *investmentInterest* and stores the result in *earnInterest*. Lines 16-29 calculates the payment of benefits in current year. Line 16 stores the benefits paid in the current year in *payment*. Lines 17-29 checks if the length of *payment* is equal to zero or not. If it is , let *paymentRec* be zore. Else, copy the data of *payment* to *paymentRec*. Lines 32-39 calculates the benefit payment of next year. Line 33 gets the benefit that should be paid for i survived years. Lines 34-37 checks if the payment is equal to zero or not. If it is, the benefit payment for the next year will be zero and that meas nobody died this year. Otherwise, the benefit payment for the next year will be equal to *payment* that contains the benefit. Finally, Line 41 creates a data frame called *profitTable* that has four columns SurviveYear, money, earnInterest and benefitPayment.

As mentioned earlier, x11() function opens a new graph window before creating a new graph to avoid overwriting of previous plotted graphs. The following Listings plots the Age, Benefit, NetSinglePremium, Die and SurviveYears columns from *bDataFrame*. Each of these column is plotted on its own window. All these plots are plotted as histograms. All the images will be saved in images directory which should be created on the same working directory. Each listing will have its output graph under it.

```
1 x11()
2 myGraph <- ggplot(bDataFrame, aes(Age))
3 myGraph <- myGraph + geom_histogram(binwidth=1, colour="black",
      fill="white") + labs(title = "Random Ages Histogram")
4 print(myGraph)
5 ggsave(filename = "images/Random Ages Histogram.png", plot = myGraph)
```

Listing 10: Random Ages Histogram

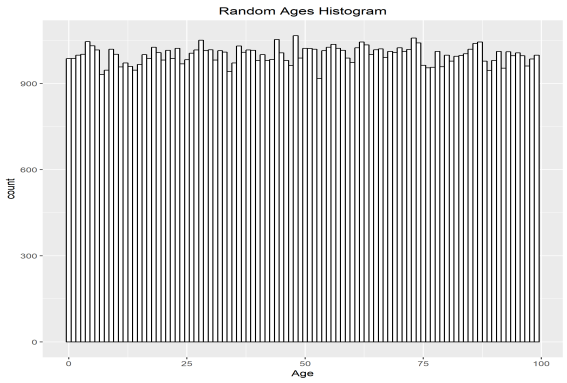


Figure 2: Random Ages Histogram

Listing 11: Random Benefit Histogram

```
1 x11()
2 myGraph <- ggplot(bDataFrame, aes(Benefit))
3 myGraph <- myGraph + geom_histogram(colour="black", fill="white") +
      labs(title = "Random Benefit Histogram")
4 print(myGraph)
5 ggsave(filename = "images/Random Benefit Histogram.png", plot = myGraph)
```

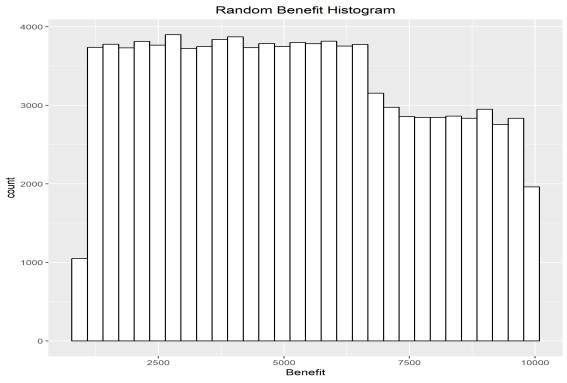


Figure 3: Random Benefit Histogram

```
x11()
myGraph <- ggplot(bDataFrame, aes(NetSinglePremium))
myGraph <- myGraph + geom_histogram(colour="black", fill="white") +
    labs(title = "Random Net Single Premium Histogram")
print(myGraph)
ggsave(filename = "images/Random Net Single Premium Histogram.png", plot
    = myGraph)
```

Listing 12: Random Net Single Premium Histogram

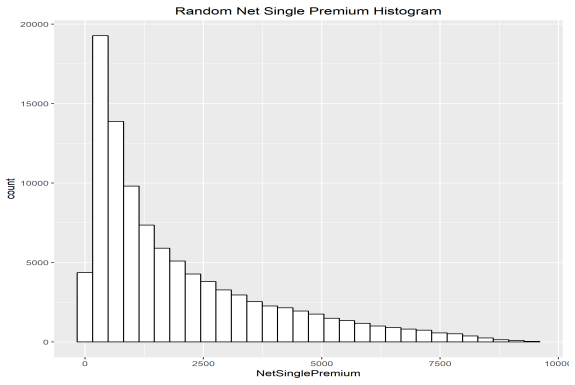


Figure 4: Random Net Single Premium Histogram

```
x11()
myGraph <- ggplot(bDataFrame, aes(Die))
myGraph <- myGraph + geom_histogram(binwidth=1, colour="black",
    fill="white") + labs(title = "Random Dead Ages Histogram")
print(myGraph)
ggsave(filename = "images/Random Dead Ages Histogram.png", plot = myGraph)
```

Listing 13: Random Death Ages Histogram

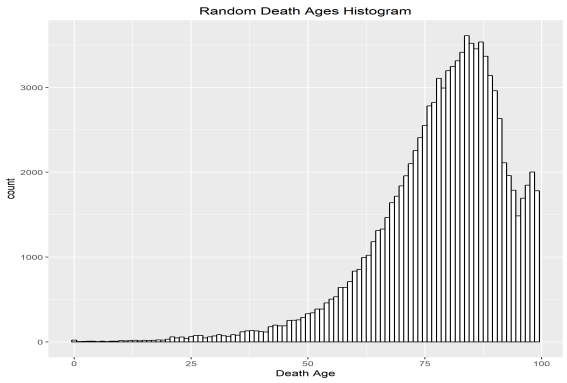


Figure 5: Random Death Ages Histogram

```
x11()
myGraph <- ggplot(bDataFrame, aes(SurviveYears))
myGraph <- myGraph + geom_histogram(binwidth=1, colour="black",
    fill="white") + labs(title = "Random Survive Years Histogram")
print(myGraph)
ggsave(filename = "images/Random Survive Years Histogram.png", plot =
    myGraph)
```

Listing 14: Random Survived Years Histogram

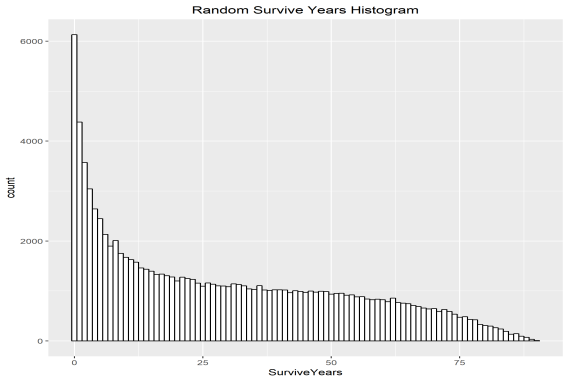


Figure 6: Random Survived Years Histogram

```
1  #------------------- print profit graph ------------------
2  meltProfitTable <- melt(profitTable, id="surviveYears")
3  x11()
4  myGraph <- ggplot(meltProfitTable, aes(x = surviveYears, y = value,
       colour = variable))
5  myGraph <- myGraph + geom_point() + labs(title="Company Profit Graph", y
       = "Money [$US]") + geom_line(linetype = "dashed") +
6  scale_color_manual(labels = c("Profit", "Interest", "Payment"), values =
       c("Green","blue", "red"))
7  print(myGraph)
8
9  ggsave(filename = "images/Company Profit Graph.png", plot = myGraph)
10
11 endTime <- Sys.time() # calulating the ending time
12 totalTime <- endTime - startTime #calulating the total time
13 print(totalTime) # printing the total time
```

Listing 15: Company Profit Graph

In Listing 15, Line 2 uses one of the reshapes's library functions. Melt function which allows you to plot more that one thing on the same window. As mentioned earlier x11() function opens a new graph window before creating a new graph to avoid overwriting of previous plotted graphs. Line 4 sets the axis of the plot. Line 5 sets the title and the plot will be plotted as lines and sets the legend of the plot. Line 7 displays the graph on the screen. Line 9 saves the graph on images directory which should be created on the same working directory. Line 11 stores the time when the code is ended its execution. Line 12 calculates the total time . Line 13 prints the total time. Figure 7 shows the output of line 7.

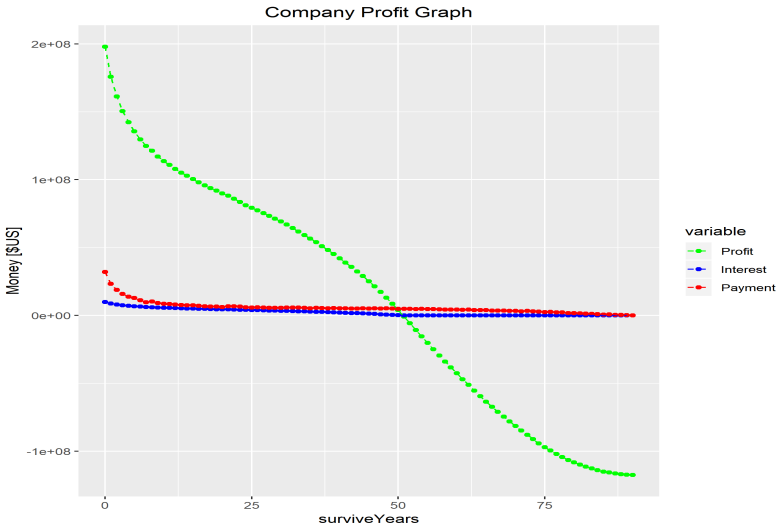


Figure 7: Company Profit Graph

```
# Now, claulating the Fund value:
# F(t)= Premium (t)+ accumlatedMonthlyinterest(t)- benefit(t)
for (i in 1:(tTimes-1)) {
fundValues[i+1] <- fundValues[i] + interestMonthBased[i] - benefitPay[i]
if(fundValues[i+1] > 0){ #only calculate Interest when fund > 0
interestMonthBased[i+1] <- fundValues[i+1] * (1+investmentInterest/12)^12
    - fundValues[i+1]
}
else {
interestMonthBased[i+1] <- 0
}
accInterMonthBased[i+1] <- accInterMonthBased[i] + interestMonthBased[i+1]
accBenPay[i+1] <- accBenPay[i] + benefitPay[i+1]
}
fundValueTable<-data.frame(year,fundValues, interestMonthBased,
    benefitPay, accInterMonthBased, accBenPay)
```

Listing 16: Calculating The Fund Value

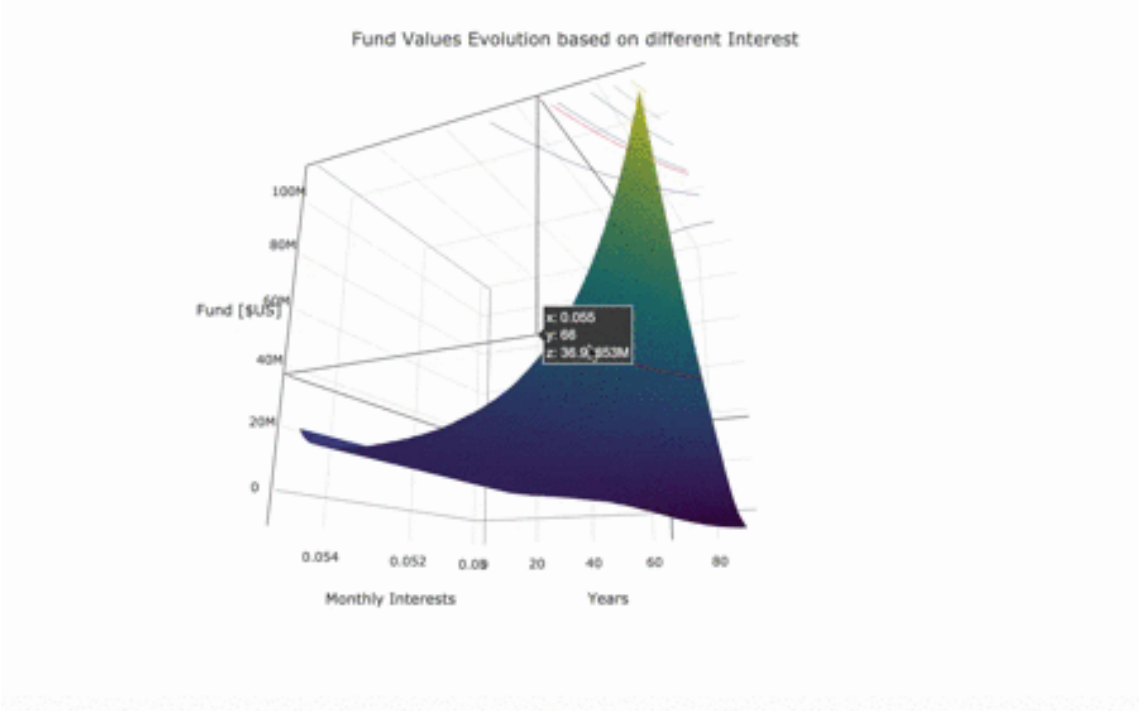
The code in listing 16 calculates the fund value introduced in equation 3. Lines 3-12 loop through the number of survived years and calculate the fund value according to equation 3. The monthly interest is calculated as well. The code stores the fund values and monthly interests in *fundValues* and *interestMonthBased* respectively. Line 14 creates a data frame called *fundValuesTable* that contains the year, fund values, monthly interest, benefit payment, accumulated monthly interest and accumulated benefit payment.

$$F(t) = Premium(t) + accumlatedMonthlyinterest(t) - benefit(t) \tag{3}$$

```
1  interestSeq <- seq(0.05, 0.055, 0.0005)
2  yearSeq <- fundValueTable$year
3  matrixFund <- matrix(, nrow = length(yearSeq), ncol =
       length(interestSeq)) # empty matrix
4  column <- 1
5
6  for (i in interestSeq){
7  matrixFund[,column] <- CalculateFundValue(i)$fundValues
8  column <- column + 1
9  }
```

Listing 17: Calculating The Fund Value Based on Different Interests

The code in listing 17 is similar to the code in listing 16 except that listing 17 calculates the fund value based on a variety of interest. The first loop calculates the fund value based in 0.05% interest. The second loop calculates the fund value based in 0.055% interest. The third loop calculates the fund value based in 0.0005% interest. The code stores the results in a matrix for a 3D surface plotting. Below is a gif message that visualizes the outcome.




```r
nrowbDataFrame <- nrow(bDataFrame)
reserveTable <- data.frame(matrix(ncol = 3, nrow = nYears))
colnames(reserveTable) <- c("SurviveYears", "Reserve", "NumberPolicies")
for (reserveYear in 1:nYears){
counter<-0
reserveAmount <- 0
for (i in 1:nrowbDataFrame){
if (bDataFrame$SurviveYears[i] >= reserveYear)
{counter <- counter + 1
#t_V = 1 - a_(x+t)/a_x
x <- bDataFrame$Age[i]
A_x_t <- lifeTable$A_x[(x+1+reserveYear)] # faster
t_V <- A_x_t
reserveAmount <- reserveAmount + bDataFrame$Benefit[i]*t_V
}}
reserveTable$SurviveYears[reserveYear] <- reserveYear-1
reserveTable$NumberPolicies[reserveYear] <- counter
reserveTable$Reserve[reserveYear] <- reserveAmount
}
```

Listing 18: Calculating The Reserve Value

Listing 18 computes the reserve value based on equation 4.

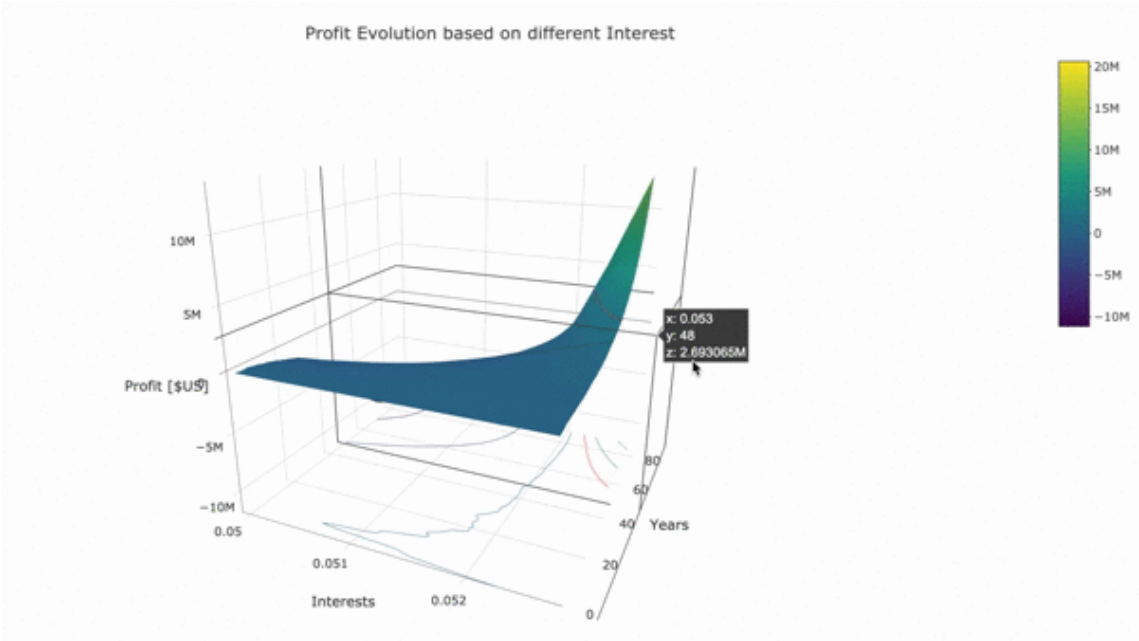
$$ {}_sV = A_{x+s} = EPV[Benefit] = \sum_{k=1}^{\infty} V^{k+1} *_{k1} q_{x+3} \tag{4} $$

$$ Profit = Premium + YearlyInterests - BenefitPayment \tag{5} $$

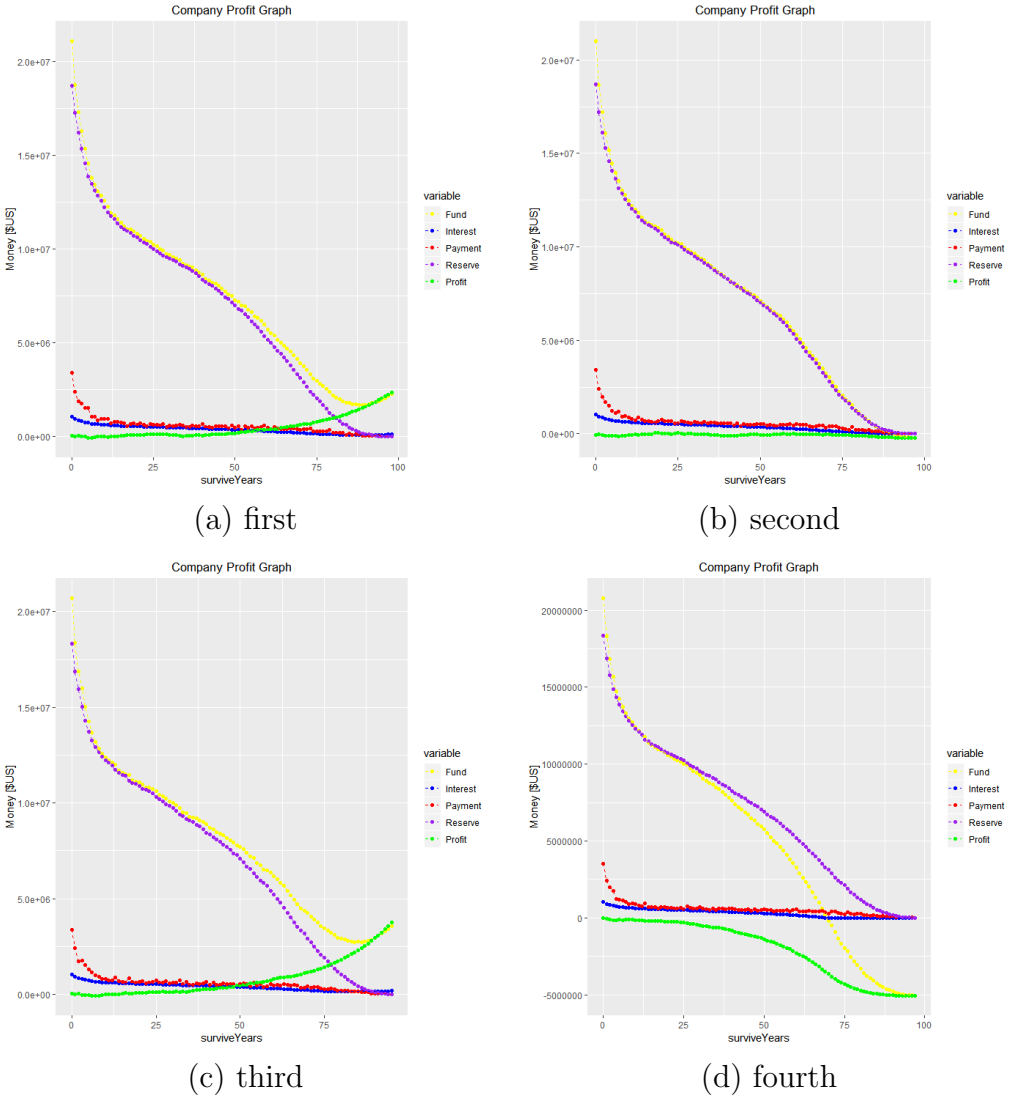
```
#find payment of the year 0
payment <- paymentTable[paymentTable$SurviveYears == 0, c("Benefit")]
if(length(payment) == 0){ #sometime there is no payment for specific
    year, so convert numeric(0) to 0
benefitPayment[1] <- 0
}else{
benefitPayment[1] <- payment
}
#find reserve of the year 0
reserve <- reserveTable[reserveTable$SurviveYears ==0, c("Reserve")]
if(length(reserve) == 0){ #sometime there is no reserve for specific
    year, so convert numeric(0) to 0
reserveHold[1] <- 0
}else{
reserveHold[1] <- reserve
}
surviveYears <- vector(mode="integer", length=(nYears+1))
surviveYears[1] <- 0
for (i in 1:nYears) {
surviveYears[i+1] <- i
money[i+1] <- money[i] + earnInterest[i] - benefitPayment[i]
if(money[i+1] > 0){ #only calculate earnInterest when money > 0
earnInterest[i+1] <- money[i+1]*investmentInterest
}
else {
earnInterest[i+1] <- 0
}
#find payment of next year
payment <- paymentTable[paymentTable$SurviveYears == i, c("Benefit")]
if(length(payment) == 0){ #sometime there is no payment for specific
    year, so convert numeric(0) to 0
benefitPayment[i+1] <- 0
}
else{
benefitPayment[i+1] <- payment
}
```

```
#find reserve of next year
reserve <- reserveTable[reserveTable$SurviveYears == i, c("Reserve")]
if(length(reserve) == 0){ #sometime there is no reserve for specific
    year, so convert numeric(0) to 0
reserveHold[i+1] <- 0
}else{
reserveHold[i+1] <- reserve
}}
profitTable <- data.frame(surviveYears, money, earnInterest,
    benefitPayment, reserveHold)
#calculate profit
profitTable$profit <- profitTable$money + profitTable$earnInterest -
    profitTable$benefitPayment - profitTable$reserveHold
```

Listing 19: Calculating The Annual Profit and The Reserve Value



Listing 19 calculates the yearly profit along with the annual reserve based on equation 4 and equation5. The gif image above visualizes the outcome. Figure 8 shows profit graphs of the project. Since we generate the benefit randomly, the profit graph will have different shape each time we run the code.



(a) first (b) second

(c) third (d) fourth

Figure 8: Profit Graphs

4 Experimenting

We increased the policies to 100000 and ran the R code (without using C) on a Windows 10 x64-bit operating system with 3.20 GHz processor. It took 4.4 minutes to simulate 100000 policies. Moreover, we simulated the process for 1000000 policies and it took 44 minutes. The reason why we did not use C code for this simulation is that the C randomizer is not working well and the R randomizer is more efficient than the C randomizer.