ANNUITY CALCULATOR TECHNICAL DOCUMENTATION

March 5, 2019

Brian Hooper
brian.hooper@cwu.edu

Divya Chandrika Kalla
divyachandrika.kalla@cwu.edu

Heather McKinnon
heather.gammon@cwu.edu

Dr. Donald Davendra
Department of Computer Science
Central Washington University

Contents

1 Overview 3

2 Inputs 3

3 Creating a Life Table 4

4 Profit Functions 5

5 Simulations 6

6 Graphing 9

7 Future Considerations 9

1 Overview

This program was coded as an R script (version 3.5.1) written in RStudio. It uses mortality data from the Society of Actuaries Mortality Tables to simulate a number of whole life single annuity premiums prices for an insurance company and produce projected annual profits. The mortality data is used to create a life table which will calculate the annuity expected present value for given ages. The program will accept user input of a starting age range, maturity age, monthly annuity benefit, interest rate, number of policy holder lifetimes to generate, number of company years to simulate, and an ROI interest rate. Using the life table data and user input, the program will create a simulated number of policy holders' lifetimes with random starting and death ages and their calculated premiums. This program will also simulate a profit for a projected number of years to measure the insurance company's annual profit. Several graphs are produced to help illustrate these calculations and together with the tables are saved in an output folder in the same directory as the R script. The calculations and design of this program were advised by Dr. Chin-Mei Chueh, actuarial science professor and Society of Actuaries council member, and Dr. Donald Davendra, computer science professor.

2 Inputs

The mortality data sets from the Society of Actuaries website contain a column for age and a column for the associated probability of death. It is downloaded as a .csv file and read into a variable, mortality_data, in this program. A user defined file, simulation_input.csv, is edited by the user and read into several other variables. This user input file can handle multiple rows of data if the user would like to test several scenarios.

```r
# Read and assign input parameters
for (input_index in 1:length(user_input$age_range_start)) {
    input_age_start = user_input[input_index,1]
    input_age_end = user_input[input_index,2]
    maturity_age = user_input[input_index,3]
    monthly_annuity = user_input[input_index,4]
    interest_rate = user_input[input_index,5]
    iterations = user_input[input_index,6]
    company_years = user_input[input_index,7]
    ROI_interest = user_input[input_index,8]
}
```

Listing 1: This for loop starts the main body of the program and will loop for every line in the simulation_input.csv file. This shows the first few lines where each value is extracted from input.csv file into individual variables.

After the user input values are stored (Listing 1), the user-defined interest rate is used to calculate some initial variables (Listing 2). These variables will be used in creating the life table which will ultimately calculate the annuity expected present values that will be necessary for calculating annuity premiums and running the scenarios.

```
# Calculated initial variables
d = interest_rate / (1 + interest_rate)
im = 12 * (((1 + interest_rate) ** (1 / 12)) - 1)
dm = 12 * (1 - (1 - d) ** (1 / 12))
a12 = (interest_rate * d) / (im * dm)
b12 = (interest_rate - im) / (im * dm)
```

Listing 2: Calculating constant variables that will be used in calculations to create the life table.

3 Creating a Life Table

After input is read in, a life table is initialized with the the age and mortality data (qx) that was read in from the .csv from the Society of Actuaries website (Listing 3). Having the initial age and mortality values is necessary to populate the rest of the table columns.

```
life_table <- data.frame(age, qx)
```

Listing 3: The line to create the life table data frame. The values read in from the mortality data table, age and associated mortality, are used to create the rest of the table.

After the data frame is set a series of calculations is run to set the values for the rest of the table. This life table is created ultimately to calculate values from the insurance (Eq. 1) and annuity (Eq. 2) equations. These values calculate the expected present value of the insurance premium from the starting age.

$$A_x = \sum_{k=0}^{\infty} v^{k+1} \cdot {}_{k|}q_x \tag{1}$$

Equation 1. The Whole Life Insurance Expected Present Value Equation

$$\ddot{a}_x = \sum_{k=0}^{\infty} v^{k} \cdot {}_{k}p_x \tag{2}$$

Equation 2. The Whole Life Annuity Expected Present Value Equation

4 Profit Functions

Functions were created to reuse as the program runs through the simulations and for plotting (Listing 4).

```
# Function for determining Whole Life Net Single Premium Policy Price
#
# @param in_age The input age for beginning the insurance policy
# @param mat_age The age in which the policy matures
# @return A double representing the Net Single Premium that was paid for
    the policy
WNS_premium <- function(in_age, mat_age){
    xEy = (life_table$lx[mat_age + 1] / life_table$lx[in_age + 1]) * (1 /
    (1 + interest_rate)) ** (mat_age - in_age)
    return(monthly_annuity * 12 * (a12 * ax[mat_age + 1] - b12) * xEy)
}

# Function for determining the reserve at time_unit (year)
#
# @param in_age The input age for beginning the insurance policy
# @param time_unit The unit of time (year) that describes the age of the
    policy
# @return A double representing the reserve value for the policy at that
    unit of time
WNS_reserve <- function(in_age, time_unit){
    x = in_age
    Axt = life_table$ax[x + time_unit]
    return(monthly_annuity * 12 * Axt)
}

# Function for determining the profit over time for given interests
#
# @param interest The input interest rate
# @return A list of calculated profits
interests_profit <- function(interest){
    yearly_interest <- c((1 + (interest/12))**12)
    f_value <- c(fund_table$fund_value[1])
    p <- c(fund_table$profit[1])

    for (year in 2:nrow(fund_table)){
      yearly_interest <- c(yearly_interest, (1 + (interest/12))**12)
      f_value <- c(f_value, (f_value[year - 1] * yearly_interest[year]) -
    fund_table$total_benefit_payout[year])
      p <- c(p, (f_value[year] - fund_table$total_reserve[year]))
    }
    return(p)
}
```

Listing 4: Functions to calculate each policy holder's premium price and reserve value and a function to find profit values for a given interest rate.

The first function, WNS_premium(), will calculate a whole life net single premium for a single policy holder with a unique starting age and maturity age. This value can be used to check the validity of the calculations for the whole program with known data. The second function, WNS_reserve(), calculates the reserve value that the company must consider and subtract from their fund value to find their actual profit value at a given time. The third function, interests_profit(), calculates a profit value at a given interest rate. This function is used later to plot profit over time with different interest rates.

5 Simulations

With the user-defined input of iterations, this program will generate lifetimes for individual policy holders in a loop (Listing 5). These individuals will have a randomly generated starting age (age they bought the insurance) and death age with a given maturity age to determine the price of the premium and the insurance company's reserve on that individual's life policy. This simulation loop is set up as if every client chose the net single premium option. All data generated by this loop is stored in a data frame called lifetimes. This table stores the starting age, maturity age, death age, policy premium price, and a reserve price for each lifetime. Placeholder columns are set for an isDead column and a benefits paid out column to be used in the profit simulation loop later.

```r
for(i in 1:iterations) {
    # Generate random integer starting age
    if(input_age_start >= input_age_end) {
        input_age = input_age_start
    } else {
        input_age = sample(input_age_start:input_age_end, 1)
    }

    # Pick a random death date based on mortality table
    death_age = input_age
    while(death_age < length(mortality_data$mortality) &&
           runif(1, 0.0, 1.0) > mortality_data$mortality[death_age]) {
        death_age = death_age + 1
    }

    # Add the simulated lifetime to policy table
    lifetimes[nrow(lifetimes) + 1,] <- c(input_age,
                                         maturity_age,
                                         death_age,
                                         WNS_profit(input_age,
                                                    maturity_age),
                                         WNS_reserve(input_age, 1),
                                         0.0,   # benefits paid out
                                         FALSE) # isDead?
}
```

Listing 5: The loop to run simulations for user-defined number of iterations. Policy starting age is randomly chosen in the age range between age start and age end and the death age is determined by using the mortality percentage form the life table for each year the policy holder lives. Once the death age and start age is determined, the policy premium price and reserve price can be determined.

The second simulation in this program (Listing 6) determines the annual profits for a user-defined set of years. Number of company years to simulate and the ROI interest rate from the user input file are used. The number of lifetimes generated from the previous loop are used to begin the simulation. This sets the year zero reserve value, fund value, and profit from "selling" these policies.

Once the loop begins, it checks to see each individual policy holder is still alive and if the age of the policy has reached the maturity date yet. If the maturity date is not met and the policy holder is alive, the reserve value is updated. If the maturity date is met and the policy holder is alive, then the reserve value is updated and a benefit payout is calculated for that year. If the policy holder is dead at the given year, then the benefit payout and reserve value is set to zero.

The data generated by this loop is added to a table called fund_table. This table will show year, total reserve, total number of payouts, total benefit payout value, yearly interest rate, fund value, profit value, number of accumulated deaths, and number of unmatured policies.

```
# Begin profit simulation loop
for (year in 2:(company_years + 1)){
    # For each year, calculate the benefit payout value and reserve
    # Also keeps track of number of payouts, total deaths, and unmatured
    policies
    one_year_payout = 0
    payouts = 0
    deaths = 0
    unmatured = 0
    for (j in 1:nrow(fund_policies)){
        matured = (fund_policies$StartAge[j] + year - 1 >
                   fund_policies$MatAge[j]) # has policy matured?
        dead = (fund_policies$StartAge[j] + year - 1 >
                fund_policies$DeathAge[j])  # is policy holder dead?

        # if not matured and not dead
        if(isFALSE(matured) && isFALSE(dead)) {
            fund_policies$Reserve[j] = WNS_reserve(fund_policies$StartAge[j
    ], year - 1)
            unmatured = unmatured + 1
            fund_policies$benefitPayout[j] = 0.0
         }

        # if matured and not dead
        else if(isTRUE(matured) && isFALSE(dead)) {
            one_year_payout = 12 * monthly_annuity
            fund_policies$benefitPayout[j] = one_year_payout
            fund_policies$Reserve[j] = WNS_reserve(fund_policies$StartAge[j
    ], year - 1)
            payouts = payouts + 1
        }

        # if dead
        else if(isTRUE(dead)){
            deaths = deaths + 1
            fund_policies$isDead[j] = TRUE
            fund_policies$Reserve[j] = 0.0
            fund_policies$benefitPayout[j] = 0.0
        }
    }

    year_num            <- c(year_num, year - 1)
    total_reserve       <- c(total_reserve, sum(fund_policies$Reserve))
    num_payouts         <- c(num_payouts, payouts)
    total_benefit_payout<- c(total_benefit_payout, sum(fund_policies$benefitPayout))
    yearly_ROI          <- c(yearly_ROI, (1 + (ROI_interest/12))**12)
    fund_value          <- c(fund_value, (fund_value[year - 1] * yearly_ROI[year]) - total_benefit_payout[year])
    profit              <- c(profit, (fund_value[year] - total_reserve[year]))
    accumulated_deaths  <- c(accumulated_deaths, deaths)
    unmatured_policies  <- c(unmatured_policies, unmatured)

} # End profit simulation loop
```

Listing 6: The for loop for simulating projected profit for a set number of years for the insurance company selling whole single net premium annuities.

6 Graphing

Four graphs are generated to illustrate some of the output data: a histogram of death ages generated by the simulated lifetimes loop, monetary values over time, how increasing age with the user-defined maturity age and monthly benefit affects premium prices, and a projected profit value with varying interest rates over time. Sample output graphs can be seen in the Figures 1, 2, 3, and 4.

7 Future Considerations

There are a few future considerations for expanding this program. One obvious one would be adding yearly, monthly, and all term policy premiums. As it is, this program calculates only whole net single premiums prices. This would probably take a major overhaul of

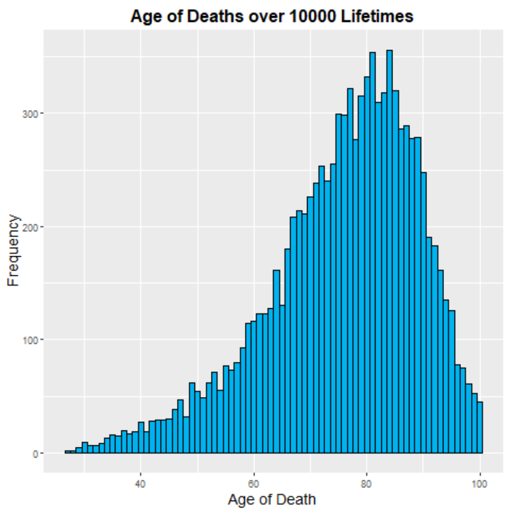

Figure 1: Histogram of deaths at a given age.

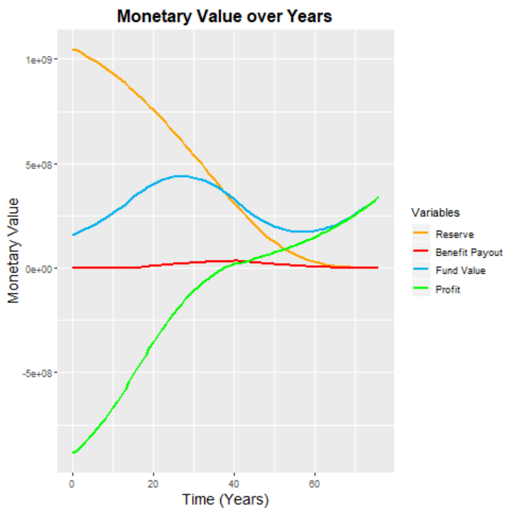

Figure 2: Benefits paid, reserve value, fund value, and profit over a user-defined set of years.

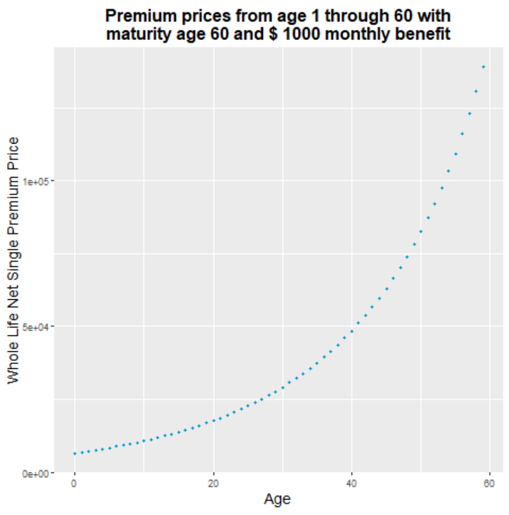

Figure 3: Prices of whole life single net premium at given age range, maturity age, and desired annuity benefit.

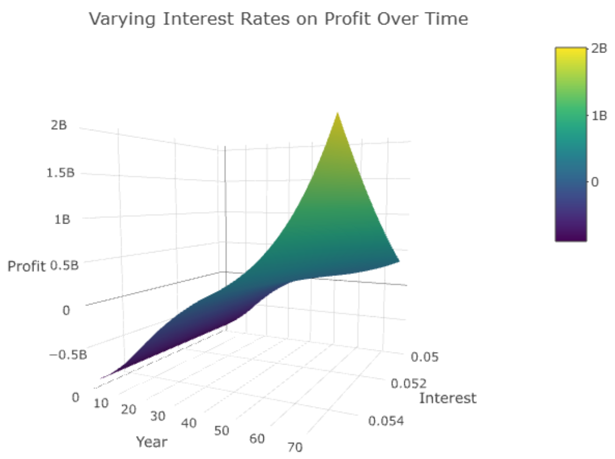

Figure 4: Company's projected profit for varying interest rates over a user-defined set of years.

existing code and make it considerably longer. Another small modification might be to have a single, easy way for a user to test just one policy. This may help if the user wants to give a single client an estimate. A third consideration may be adding another user input file to be able to run several different mortality tables at once to compare to each other. For instance, the client could compare non-smoker mortality data to smoker mortality data. These are just a few ideas however and implementing these considerations or other ideas would be entirely dependant on the client's needs.

References

[1] Society of Actuaries Mortality and Other Rate Tables. Available: https://mort.soa.org/?_ga=2.155909691.1059898131.1547157104-186254515.1517263599.