ANNUITY CALCULATOR TECHNICAL DOCUMENTATION

February 19, 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

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 annuity expected present value for given ages. The program will accept user input of a starting age range, maturity age, monthly annuity benefit, and interest rate. Then using the life table data, 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 see the insurance company's annual profit. These projected years will take user input of number of years, ROI interest, investment percentage, and a sales goal for number of policies to sell per year. 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. Two user defined files, input.csv and ROI_input.csv, are edited by the user and read into several other variables. Each of these .csv files can handle multiple rows of data if the user would like to test several scenarios. The rows of these files correspond with each other, but the program will also handle if the user would like to test several scenario rows in the input.csv file with only one row in the ROI_input.csv file.

```
# Read and assign input parameters
for (input_index in 1:length(user_input$input_age_start)) {
input_age_start = user_input[,1]
input_age_end = user_input[,2]
maturity_age = user_input[,3]
monthly_annuity = user_input[,4]
interest_rate = user_input[,5]
```

```
8 iterations = user_input[,7]
```

Listing 1: This for loop starts the main body of the program and will loop for every line in the 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 calculated 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.

```
1 # Calculated initial variables
2 d = interest_rate / (1 + interest_rate)
3 im = 12 * (((1 + interest_rate) ** (1 / 12)) - 1)
4 dm = 12 * (1 - (1 - d) ** (1 / 12))
5 a12 = (interest_rate * d) / (im * dm)
6 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.

```
1 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.

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

Equation 1. The Whole Life Insurance Expected Present Value Equation

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

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 (Listing 4).

```
# Function for determining Whole Life Net Single Premium Profit for company
#
# @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_profit <- function(in_age, mat_age){
  xEy = (lx[mat_age + 1] / 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 Whole Life Net Single Premium loss for company
# Occurs only when death_age > maturity_age
#
# @param mat_age The age in which the policy matures
# @param death_age The age in which the policy holder dies
# @return A double representing the total paid out to the client for the
    policy
WNS_loss <- function(mat_age, death_age)
  return ((death_age - mat_age) * monthly_annuity * 12)

# Calculate net profit or loss for Whole Life Net Single Premium
#
# @param in_age The input age for beginning the insurance policy
```

```r
# @param mat_age The age in which the policy matures
# @param death_age The age in which the policy holder dies
# @return A double representing the net profit or loss for a single policy
    holder
WNS_gross_profit <- function(in_age, mat_age, death_age){
  profit <- WNS_profit(in_age, mat_age)
  if (death_age <= maturity_age){
    return (profit)
  }
  else
    loss <- WNS_loss(mat_age, death_age)
    return (profit - loss)
}
```

Listing 4: Three functions to calculate the insurance company's gross profit for whole single net premiums. One calculates the profit (or the premium price), one the loss as annuity payments are made to the policy holder, and one to return the gross profit.

The first function, WNS_profit(), 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_loss(), calculates and returns any loss from the insurance company if the age of maturity is reached before death of the policy holder. The third function, WNS_net_profit(), calculates and returns the combined gross profit and loss for the company for a single policy holder.

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 gross profit on that individual's life policy. This simulation loop is set up to test if every client chose the net single premium option. All data generated by this loop is stored in a data frame called policy_table. This table stores the starting age, maturity age, death age, policy premium price, a logical displaying if the individual died before the maturity date, and the gross profit at the end of the individual's lifetime.

```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
  policy_table[nrow(policy_table)+1,] <- c(input_age,
                    maturity_age,
                    death_age,
                    WNS_profit(input_age, maturity_age),
                    (death_age <= maturity_age),
                    WNS_gross_profit(input_age, maturity_age, death_age))
}
```

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 gross profit can be determined.

The second simulation in this program (Listing 6) determines the projected yearly annual gross income for a user-defined set of years. A .csv file takes user input of number of years, ROI interest, investment percentage, and a yearly sales goal. The yearly sales goal determines how many policies are sold per year of the simulation. Before the loop initiates, the yearly sales goal number of these policies is randomly selected from the pool of individuals that were generated in the lifetime simulation. This sets the year zero profit from "selling" these policies. Then the investment percentage of this profit is "invested" for year zero.

Once the loop begins, it checks to see if the individual policy holder is still alive and if the age of the policy has reached the maturity date yet. If the maturity date is met and the client is still alive, then this will contribute to a loss for the insurance company, as the annuity payments are paid out.

After the loss is determined, the number of new policies sold is again randomly selected from the lifetime simulation pool. The new policies are added to the end of the previous policies and the polices age for all older policies is incremented by 1 (year). These new policies will add to yearly profit and a portion equal to the investment percent is added to the amount invested.

Finally, the data generated by this loop is added to a table called ROI_tracker. This table will show year, total loss (payouts), sold_policies, ROI (calculated by ROI interest), invested amount, and the final ROI_adjusted_profit. The ROI adjusted profite represents the gross annual profit for one year.

```
for (i in 2:company_years){
  # checking the current age of the policy holder (if mature, and the
    policy holder is not dead yet)
  one_year_loss = 0
  for (j in 1:nrow(policies)){
    if (isTRUE(policies$StartAge[j] + policies$policyAge[j] > policies$
   MatAge[j]) && isTRUE(policies$StartAge[j] + policies$policyAge[j] <
    policies$DeathAge[j])) {
      loss <- WNS_loss(policies$MatAge[j], policies$DeathAge[j])
      one_year_loss = one_year_loss + loss
    }
  }

  # add new policies sold
  new_policies <- policy_table[sample(nrow(policy_table),policy_sales_goal)
    ,]
  new_policies$policyAge <- c(0)
  policies <- rbind(policies,new_policies)
  policies$policyAge <- policies$policyAge + 1 # increment policy ages

  # concatenate data from loop to ROI variables
  year <- c(year, i-1)
  total_loss <- c(total_loss, one_year_loss)
  sold_policies <- c(sold_policies, sum(new_policies$PolicyCost))
  ROI <- c(ROI, invested[i-1] * ROI_interest)
  # This reinvests the ROI for the year and the investment_percent value of
     the policies_sale_goal sold for the year
  invested <- c(invested, ROI[i] + invested[i-1] + (sold_policies[i] *
    investment_percent))
  ROI_adjusted_profit <- c(ROI_adjusted_profit, (invested[i] + sold_
    policies[i] - (sold_policies[i] * investment_percent) - one_year_loss))
```

```
25 } # End yearly profit projections
```

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, age effect on the annuity expected present value (äx), how increasing age with the user-defined maturity age and monthly benefit of net single premium prices, and a projected gross income for a user-defined set of years with sales, investments, ROIs, and paying client's their annuities. Sample output graphs can be seen in the Figures 1, 2, 3, and 4 below.

7 Future Considerations

There are a few future considerations for expanding this program. One obvious one would be adding yearly, monthly, and all n-term policy premiums. As it is, this program calculates only whole net single premiums prices. This would probably take a major overhaul of 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 is entirely dependant on the user's needs however, but may help if the user wants to give a single client an estimate. A third consideration may be adjusting the loop that forecasts yearly gross profits. As it stands, the profit projection is calculated with an initial set of policy sales but there are no policy holders with a policy age over zero years when the loop starts. This creates data that looks very profitable in the first few years because there are no annuity checks being paid out. However, the data generated can still be useful, and again depends on the user's needs.

References

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

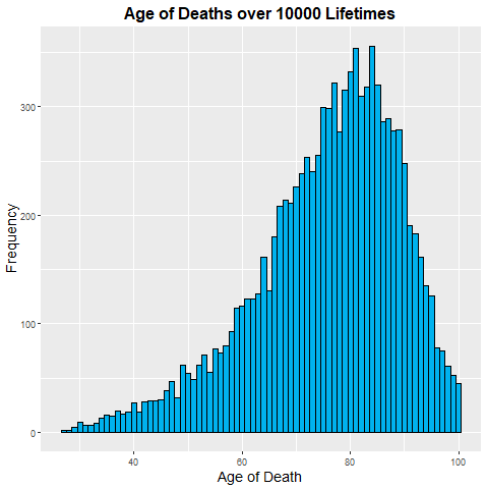

Figure 1: Histogram of deaths at a given age.

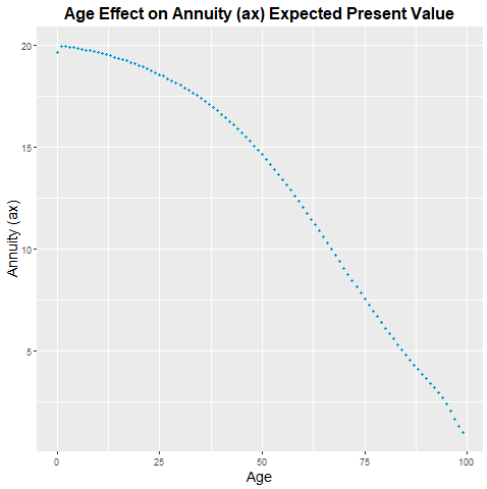

Figure 2: Annuity expected present value at a given age.

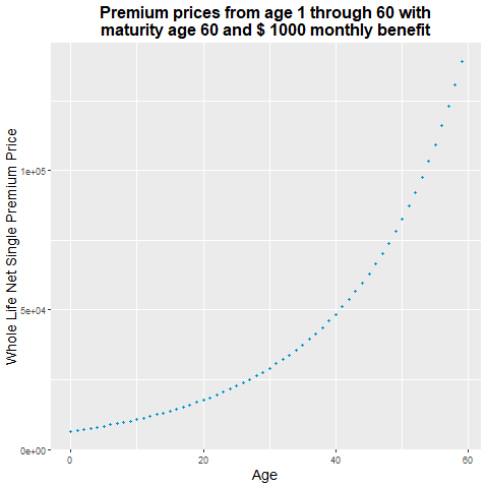

Figure 3: Prices of whole life single net premium at given age range, maturity age, and desired annuity benefit.

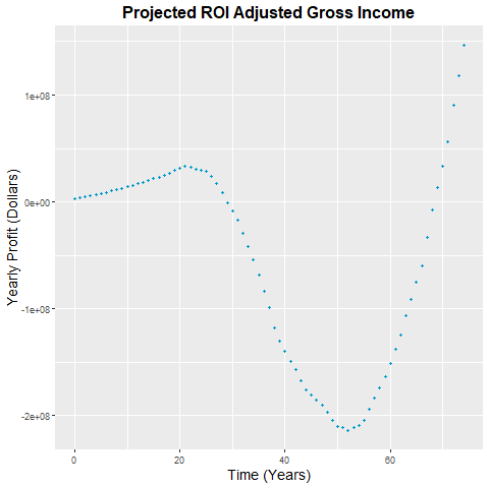

Figure 4: Company's projected gross profit per year adjusted with ROI from investments and selling a user-defined number of whole life single net premium policies per year.