ST227: Applications of R in Life Insurance

Functions in R

Dr. Viet Dang

27/02/2021

Overview

- These ST227 workshops explore R's application in the context of Life Insurance. The students are assumed to be familiar with the following topics:
  - Data Types (Numeric, Character and Logical)
  - Data Structures (Vector, List and Data Frames)
  - Basic Iterations (`for` loops and `apply` family of functions).

- If you are not familiar with the above listed topics, you are recommended to consult either:
  - ST226 course material,
  - LSE's pre-sessional R course,
  - Office hours!

Motivation

- Why programming?
  - Data revolution: modern needs to tackle big, diverse and complex data sets.
  - Reusable codes and miminising human errors.
  - **Very** in demand and required in the job market.
- Why R?
  - High-level language and simple syntax. Minimal communication with the underlying machine.
  - *Functional Programming* style focusing on complex operations and simple data structures.
  - The de-facto official language of statistical analysis.
  - Interface with Python, C,C++ and more.
  - The actual official language of the IFoA.

How to be a successful R programmer?

- Most R programmers (including myself) are not very good at programming.
- Reason: R "programmers" are mostly trained statisticians and amateur programmers.
- If that sounds like you, it's important to expect the challenges:
  - It is application-driven.
  - Difficult to maintain readability of codes.
  - The perfect codes can eliminate human errors. However, writing code can be *very* error-prone.
  - You spend more time debugging than writing codes.
- How to overcome these challenges?
  - Study R codes in contexts.
  - Modularising codes into functions (more later).
  - Frequent testing.
  - Again, frequent testing.
- Ultimately, programming is mostly mental muscle memory. Practice often.
- **All** LSE academics in the department of statistics are experienced R users. Chase after your instructors. Utilise your resources!

Functions

- R is a functional programming language:
  - We focus on the many *operations* performed on data. (as opposed to *object-oriented* languages).
  - Almost all tasks in R are achieved by defining, composing and reusing functions.
- What exactly are **functions**?
- **Mathematics** definition: a rule that maps each input value in the domain to a corresponding output in the co-domain. For example:

$$f : \mathbb{R} \to \mathbb{R}, \quad x \to x^2.$$

- **Programming** definition: a set of instructions that can be executed whenever called for. Let's see an example:

```
f <- function(x){
  x^2
}
```

Functions

- Anatomy of a function:
  - A name so we can refer to it. (There are anonymous functions, but let's not worry about it for now.)
  - An input *argument* or several arguments.
  - The *body* of the function which contains instructions.
- When we *call* for this function, its body is executed:

```
f(x = 3)
f(2)
```

- An example of a function with multiple input arguments:

```
g <- function(x,y){
  x^2 + y^2
}
g(x=1,y=2)
```

```
## [1] 5
```

Functions

-Major differences to **mathematical** functions: 1. A programming function can have no input at all. 2. It might modify the context around it. 3. The input might not uniquely determine the output.

- Example

```
g <- function(){
  print("Hello World!")
  print("Are you enjoying ST227?")
}
g()
```


- An example of non-unique output:

```
h <- function(){
  rnorm(1,mean=0,sd=1)
}
```


- Technically - if you know the *seed* and the random number algorithm - then the output is unique. But conceptually we can think about this as a random-output function.
- Modifying external context and non-unique outputs can be undesirable behaviours. Document instances of them and proceed with cautions.

Modelling Lifetime

- Let us fix a few notations:
  - T_x : remaining lifetime of an individual aged x (an $\mathbb{R}^+$ valued random variable).
    - ${}_tp_x = \mathbb{P}(T_x \geq t)$, i.e. the t-year survival probability of the life aged x.
    - ${}_tq_x = \mathbb{P}(T_x < t) = 1 - {}_tp_x$, i.e. the death within t years probability of a life aged x.
    - $f_x(t) = -\frac{d}{dt}\,{}_tp_x$, i.e. the density of T_x.
- Assume the distribution of T_0 is known, the force of mortality is defined by:

$$\mu_x = \lim_{\epsilon \to 0^+} \frac{\mathbb{P}(T_x \leq \epsilon)}{\epsilon}$$

- The reverse is also true. If we are given a function $\mu : \mathbb{R}^+ \to \mathbb{R}^+$, then we can recover ${}_tp_x$ by:

$${}_tp_x = \exp\left(-\int_x^{x+t} \mu_s ds\right) = \exp\left(-\int_0^t \mu_{x+s} ds\right).$$

- The remaining lifetime distribution for all ages are completely determined by the force of mortality.
- From a computational point of view, we have reduced from a two-argument function, i.e. ${}_tp_x : \mathbb{R}^+ \times \mathbb{R}^+ \to [0, 1]$ to a one-argument function, i.e. $\mu : \mathbb{R}^+ \to \mathbb{R}^+$.
- Constructing a mortality function from scratch is beyond the scope of this course. we will examine a few commonly used forms.

- Let us consider the simplest case, constant mortality. Suppose $\mu_t \equiv 0.05$. Calculate the probability that individuals aged $20, 40$ and 80 will survive the next 20 years.
- Direct calculations are possible.
- We will use the "complicated" method to achieve a general template for this type of problems.


```
mu <- function(t){
  rep(0.05,times=length(t))
}
tpx <- function(t,x){
  exp (-integrate(mu,lower =x,upper =t+x)$value)
}
tpx(t=20 ,x =20)
tpx(t=20 ,x =40)
tpx(t=20 ,x =80)
```


- We duplicate the output of μ for vectorisation purpose. We'll revisit this shortly.
- We see that it doesn't matter what the current age is, this person has the same chance of surviving the next 20 years. This is called the memoryless property. Is this a good way to model human lifetime?

- A class of commonly considered mortality is called Gompertz-Makeham's (or simply Makeham's) mortality, which has the form:

$$A + Bc^x$$

- Exercise: using Makeham's mortality with $A = 5 \times 10^{-4}$, $B = 7.5858 \times 10^{-5}$ and $c = 1.09144$, calculate the probability that individuals aged 20,40 and 80 will survive the next 20 years.


```
A = 5e-04; B=7.5858e-05; c = 1.09144
mu <- function(t){
  A+B*c^t
}
tpx <- function(t,x){
  exp(-integrate(mu,lower=x,upper=t+x)$value)
}
tpx(t=20,x=20)
tpx(t=20,x=40)
tpx(t=20,x=80)
```

A quick note on vectorisation

- In programming - one often needs to apply a function $f : \mathbb{R} \to \mathbb{R}$ to a vector $x = (x_1, .., x_n)$ on an element-by-element basis, i.e.

$$f(x) = (f(x_1), f(x_2), .., f(x_n)) .$$

- This is called *vectorisation*. Many base functions in R have been vectorised by design. For instance:

```
(1:6)^2
```


- The integrate routine requires a **vectorised** function (see ?integrate).
- The above defined function tpx has not been vectorised. If you try tpx(t=20:25,x=20), it will throw an error.
- We will need to vectorise it in t for later applications of integrate. Let us redefine it:


```
tpx <- function(t,x){
  sapply(t, function(t){
    exp(
      -integrate(mu,lower=x,upper=t+x)$value
    )
  })
}
```

Curtate Lifetime

- The curtate lifetime of an individual is the integer part of their total lifetime. For instance, if a person dies at age 85 years and 6 months, than the curtate life time is 85.
- Denote by K_x the remaining curtate lifetime of an individual aged x. Then:

$$K_x = \lfloor T_x \rfloor,$$

where the mapping $t \to \lfloor t \rfloor$ is the *floor function*.
- Exercise: calculate the curtate lifetime of a 20-year old individual with Makeham's mortality function.
- The expected curtate lifetime is:

$$\mathbb{E}(K_{20}) = \mathbb{E}(\lfloor T_{20} \rfloor) = \int_0^\infty \lfloor t \rfloor \times \mu_{20+t} \times {}_tp_{20} dt.$$

- Let us define the integrand. The floor function is built into R, so we can use it directly:


```
integrand <- function(t){
    floor(t)*mu(20+t)*tpx(t,20)
}
integrate(integrand,lower=0,upper=100)
```

- Handling an integral over $(0, \infty)$ is a bit tricky. We replace ∞ with 100 as an approximation.

Curtate Lifetime

- It can be shown that:

$$\mathbb{E}(K_x) = \sum_{n=1}^{\infty} {}_n p_x.$$

- Utilise this formula to calculate the expected curtate lifetime.

```
probs <- sapply(1:100,function(n){tpx(n,20)})
sum(probs)
```

- Given an initial age x_0 and maximum age ω, let l_{x_0} be an arbitrary positive number (usually 100,000). For any $0 \le t \le \omega - x_0$, define:

$$l_{x_0+t} = l_{x_0} \times {}_t p_{x_0}.$$

- l_{x+0+t} can be interpreted as the number of survival at time t out of an l_{x_0} number of individuals aged x_0 at time 0.
- Moreover, define:

$$d_x = l_x \times q_x = l_x \times (1 - p_x).$$

- A life table is typically expressed in the following format (for concreteness, assume $x_0 = 10$ and $\omega = 100$):

x	l_x	d_x
10	l_{10}	d_{10}
11	l_{11}	d_{11}
...	...	...
100	l_{100}	d_{100}

Life Tables

- Given a mortality function μ and boudnary ages x_0 and ω, we can construct a life table:

```
x0 <- 10; omega <- 100
x <- x0:omega
lx <- numeric(length(x))
lx[1] <- 1e05 #1e05 = 10^5
for(i in 2: length(lx)){
  print(x[i-1])
  lx[i] = lx[i-1]* tpx(1,x[i-1])
}
```

- We will also need the sequence of probabilities p_x and q_x for $x = 10, ..., 100$.

```
px <- sapply(x,function(n){tpx(t=1,x=n)})
qx <- 1 - px
dx = lx*qx
```

- The last step is to assumble the data frame:

```
lifeTable <- data.frame(x,lx,dx)
```