The lifecontingencies Package: Performing Financial and Actuarial Mathematics Calculations in R

Giorgio Alfredo Spedicato
StatisticalAdvisor

Abstract

It is possible to model life contingency insurances with the **lifecontingencies** R package, which is capable of performing financial and actuarial mathematics calculations. Its functions permit one to determine both the expected value and the stochastic distribution of insured benefits. Therefore, life insurance coverage can be priced and portfolios risk-based capital requirements can be assessed. This paper briefly summarizes the theory regarding life contingencies that is based on financial mathematics and demographic concepts. Then, with the aid of applied examples, it shows how the **lifecontingencies** package can be a useful tool for executing routine, deterministic, or stochastic calculations for life-contingencies actuarial mathematics.

Keywords: life tables, financial mathematics, actuarial mathematics, life insurance.

1. Introduction

This vignette is based on the package's paper published in JSS, Spedicato (2013). Apart of keeping track of package's updates, a major difference with respect to JSS publication is that parallel features are turned off to cope with CRAN packages' submission policies. As of March 2014, the **lifecontingencies** package (Spedicato 2013) appears as the first R package that deals with life contingent actuarial mathematics. The R statistical programming environment (Team 2012) has become the primary reference software for academics. Even in a business context, R is now considered a valid alternative to affirmed proprietary packages for statistics and data analysis, namely SAS (SAS Institute Inc. 2011), MATLAB (The MathWorks, Inc. 2011) and SPSS (IBM Corp 2012). Some packages for actuarial applications have already been developed within R. However, most of them mainly focus on non-life insurance. In fact, non-life insurance modeling involves more data analysis and applied statistical modeling than that of life insurance. Functions allowing one to fit loss distributions and perform credibility analysis are provided within the package **actuar** (Christophe Dutang, Vincent Goulet, and Mathieu Pigeon 2008). This package represents the computational side of the classical actuarial textbook on loss distribution (Klugman, Panjer, Willmot, and Venter 2009). The package **ChainLadder** (Gesmann and Zhang 2011) provides functions that are capable of estimating loss reserves for non-life insurance. Generalized linear models (GLMs), widely used in non-life insurance rate-making, can be fit by functions bundled within base R distributions. The generalized additive models for location, shape and scale (GAMLSS) and tweedie regression, which are both more advanced predictive models used by actuaries, are handled by specifically developed packages such as **gamlss** (Rigby and Stasinopoulos 2005;

Stasinopoulos and Rigby 2007) and **cplm** (Zhang 2011).

Life insurance actuarial work deals mainly with demographic and financial data. The CRAN task view "Empirical Finance" (Eddelbuettel 2013a) lists several packages tailored specifically for financial analysis. Packages **YieldCurve** (Guirreri 2010) and **termstrc** (Ferstl and Hayden 2010) are capable of financial modeling for interest rates. Among the few packages that handle demographic data, **demography** (Rob J Hyndman, Heather Booth, Leonie Tickle, and John Maindonald 2011) and **LifeTables** (Riffe 2011) can be used to manage demographic projections and life tables.

On the other hand, many commercial software packages tailored specifically for the actuarial analysis of life insurance are already available. **MoSes** (Towers Watson 2011) and **Prophet** (SunGard 2012) are currently the leading actuarial software packages for life insurance modeling. The **lifecontingencies** package aims to represent the computational R companion of the theoretical concepts exposed in textbooks like the classical Bowers, Jones, Gerber, Nesbitt, and Hickman (1997) and Dickson, Hardy, and Waters (2009) for actuarial mathematics and Broverman (2008) for financial mathematics. Package **lifecontingencies** is available from the Comprehensive R Archive Network at `https://cran.r-project.org/package=lifecontingencies`. The examples used throughout this paper have been taken from Chris Ruckman and Joe Francis (2006) and Finan (2014), both freely available financial and actuarial mathematics textbooks. The paper is structured as follows: Section 2 outlines the statistical and financial mathematical theory regarding life contingencies, Section 3 overviews the structure of the **lifecontingencies** package, Section 4 gives a wide choice of applied **lifecontingencies** examples, and finally, Section 6 discusses the packages current and future development as well as its known limitations.

2. Statistical and financial foundations of life contingencies

The actuarial pricing and reserving of life contingent insurances involves the calculation of statistics regarding occurrences and amounts of future cash flows. For example, the insurance pure premium (also known as benefit premium) can be regarded as the expected value of the prospective benefits cash flow distribution, valued at time zero for a given interest rate structure. The probabilities of the prospective benefits cash flow are based on the occurrence of the policyholder's life events (life contingencies). In addition, the theory of interest is used to find the present value of these amounts that will occur in the future. Therefore, life insurance actuarial mathematics is based on concepts derived from demography and theory of interest.

A life table (also called a mortality table or actuarial table) is a table that shows how mortality affects subjects of a cohort across different ages. For each age x, it reports the number of l_x individuals living at the beginning of age x. It is a sequence of $l_0, l_1, \ldots, l_\omega$, where l_ω, the terminal age, represents the latest age that a subject of the cohort can survive until. Life tables are typically distinguished according to gender, year of birth and nationality, with different categories being used depending on the type of life contingency (i.e., assurance or annuity). As another example, businesses may have different customers with different underlying mortalities, so they will be in need of different life tables.

From a statistical point of view, a life table allows one to deduce the probability distribution of the future lifetime for a policyholder aged x. In particular, a life table also permits one to

derive two key probability distributions: $\tilde{T}_x$, the complete future lifetime for a policyholder aged x, and its curtate form, $\tilde{K}_x$, the number of future years completed before death. Therefore, many demographic statistics can be derived from a life table, of which a non-exhaustive list follows:

- ${}_tp_x = \frac{l_{x+t}}{l_x}$, the probability that a policyholder alive at age x will reach age $x+t$.
- ${}_tq_x$, the complementary probability of ${}_tp_x$.
- ${}_td_x$, the number of deaths between age x and $x+t$.
- ${}_tL_x = \int_0^t l_{x+y}dy$, the expected number of years lived by the cohort between ages x and $x+t$.
- ${}_tm_x = \frac{{}_td_x}{{}_tL_x}$, the central mortality rate between ages x and $x+t$.
- e_x, the curtate expectation of life for a subject aged x, $e_x = \mathsf{E}[\tilde{K}_x] = \sum_{k=1}^{\infty} {}_kp_x$.
- $\mathring{e}_x$, the complete expectation of life for a subject aged x, $\mathring{e}_x = \mathsf{E}[\tilde{T}_x] = \int_0^{\infty} {}_tp_x\, dt = T_x/l_x$, with $T_x = \int_0^{\infty} l_{x+y}\, dy$. Under the uniform distribution of deaths assumption (used by the package) $\mathring{e}_x \approx e_x + 0.5$. Both quantities can also be restricted to an n-year period, $e_{x:\overline{n}|} = \sum_{k=1}^{n} {}_kp_x$ and $\mathring{e}_{x:\overline{n}|} = \int_0^n {}_tp_x\, dt = {}_nL_x/l_x$; in that case $\mathring{e}_{x:\overline{n}|} = e_{x:\overline{n}|} + 0.5\,(1 - {}_np_x)$ under UDD.

The Keyfitz textbook (Keyfitz and Caswell 2005) provides an exhaustive coverage of life table theory and practice. Life tables are usually published by institutions that have access to large amounts of reliable historical data, like social security bureaus. It is common practice for actuaries to start from these life tables and adapt them to the insurer's portfolio actual experience.

Classical financial mathematics deals with monetary amounts that can be available at different times. The present value of a series of cash flows, expressed by Equation 1, is probably the most important concept. The present value (PV) can be considered as the value – in current money – of a series of financial cash flows, CF_t, that are available at different periods of time.

$$\mathrm{PV} = \sum_{t \in T} \mathrm{CF}_t (1 + i_t)^{-t} \tag{1}$$

The interest rate, i, represents the measure of the price of money available in future times. Like the interest rate, the time value of money can be expressed by the discount rate $d = \frac{i}{1+i}$. This paper will use the i symbol to express effective compound interest, with money invested once per period. In the case where money is invested more frequently, say m times per period, each fractional period is named the interest conversion period. During each interest conversion period, the real interest rate $\frac{i^{(m)}}{m}$ is earned, where the $i^{(m)}$ expression defines the convertible (also known as "nominal") rate of interest payable m times per period.

Equation 2 combines the various notations for interest and discount rates, both on an effective and convertible basis, to express what an amount of \$1 invested now will grow to at time t.

$$A(t) = (1+i)^t = (1-d)^{-t} = v^{-t} = \left(1 + \frac{i^m}{m}\right)^{tm} = \left(1 - \frac{d^m}{m}\right)^{-tm} \tag{2}$$

All financial mathematics functions (like annuities, $a_{\overline{n}|}$, or accumulated values, $s_{\overline{n}|}$) can be rewritten as particular expressions of Equation 1, as shown in classical actuarial mathematics textbooks.

Actuaries use the probabilities inherent in life tables to price life contingent insurances. In fact, life contingencies are stochastic variables themselves. A life contingent insurance can be represented by a sequence of one or more payments whose occurrence, timing, and consequent present value are not certain. In fact, their eventual occurrence and timing depend on events in the life of the policyholder; for this reason, they are named "life contingencies". Since the actuary focuses on the present value of such uncertain payments, future payments of life contingent insurances need to be discounted using interest rates that may also be considered stochastic. The **lifecontingencies** package provides functions that model most of the standard life contingent random variables, $\tilde{Z}$, and in particular their expected value, the actuarial present value (APV). Of all the statistics used by actuaries, the APV is certainly the most important. In fact, it represents the average cost of the benefits guaranteed to the policyholder by the insurer. In a non-life insurance context, it would be named pure premium. The policyholder pays out the gross premium, G, which is a sum of benefit premiums, loading for expense, profits and taxes. Life contingencies can be either continuous or discrete, as the cited actuarial mathematics textbooks explain. Directly, the **lifecontingencies** package can only model discrete life contingencies with a non-stochastic interest rate. Nevertheless, most continuous time life contingent insurances can be easily derived from their discrete form under broad assumptions that can be found in the cited textbooks.

A few examples of life contingent insurances follow:

1. An n-year term life insurance provides a payment, if the insured dies within n years from issue. If the payment is performed at the end of the year of death, $\tilde{Z}$ can be written as
$$\tilde{Z} = \begin{cases} v^{K+1}, & \tilde{K}_x = 0, 1, \ldots, n-1, \\ 0, & \tilde{K}_x \geq n. \end{cases}$$
Its APV expression is $A^1_{x:\overline{n}|}$.

2. A life annuity consists of a sequence of benefits paid out as long as the insured life survives. In particular, a temporary life annuity due pays a benefit at the beginning of each period as long as the annuitant aged x survives, for up to a total of n years, or n payments. $\tilde{Z}$ can be written as
$$\tilde{Z} = \begin{cases} \ddot{a}_{\overline{K+1}|}, & \tilde{K}_x < n, \\ \ddot{a}_{\overline{n}|}, & \tilde{K}_x \geq n. \end{cases}$$
Its APV expression is $\ddot{a}_{x:\overline{n}|}$.

3. An n-year pure endowment insurance grants a benefit payable at the end of n years if the insured survives at least n years from issue. $\tilde{Z}$ can be written as
$$\tilde{Z} = \begin{cases} 0, & \tilde{K}_x < n, \\ v^n, & \tilde{K}_x \geq n. \end{cases}$$
Its APV expression is $A_{x:\overline{n}|^1}$ (or ${}_nE_x$).

4. An n-year endowment insurance will pay a benefit at the year of death or at the end of the n-th year, whichever occurs earlier. $\tilde{Z}$ can be written as

$$\tilde{Z} = \begin{cases} v^{K+1}, & \tilde{K}_x = 0, 1, \ldots, n-1, \\ v^n, & \tilde{K}_x \geq n. \end{cases}$$

   Its APV expression is $A_{x:\overline{n}|}$.

Interested readers could see the cited references for formulas regarding other life contingent insurances like $(DA)^1_{x:\overline{n}|}$, the decreasing term life insurance, or $(IA)^1_{x:\overline{n}|}$, the increasing term life insurance. There are also common variations of payments that form arrangements like deferment or fractional payments. Similarly, it is possible to define insurances and annuities depending on the survival status of two or more lives. For example, A_{xy} and $a_{\overline{xy}}$ represent, respectively, the APV symbols for the two lives joint-live insurance and the two lives last-survivor annuity immediate.

The **lifecontingencies** package provides functions that allow an actuary to perform classical financial and actuarial mathematics calculations. In addition to standard deterministic modeling, a peculiar feature of **lifecontingencies** is that it allows one to generate variates from the stochastic distribution of the present value of future benefits, $\tilde{Z}$, for most life contingent insurances. This feature allows for a deeper assessment of the insurance liabilities variability.

3. The structure of the package

The package **lifecontingencies** contains classes and methods that handle life tables and actuarial tables in a convenient manner.

The package is loaded within the R command line interace as follows:

```
R> library("lifecontingencies")
```

Two main S4 classes have been defined within the **lifecontingencies** package: the '`lifetable`' class and the '`actuarialtable`' class. The '`lifetable`' class is defined as follows:

```
R> showClass("lifetable")

Class "lifetable" [package "lifecontingencies"]

Slots:

Name:          x        lx      name
Class:   numeric   numeric character

Known Subclasses: "actuarialtable"
```

Class '`actuarialtable`' inherits from the '`lifetable`' class; it is different from the '`lifetable`' class because it has one more slot accounting for the interest rate.

```
R> showClass("actuarialtable")

Class "actuarialtable" [package "lifecontingencies"]

Slots:

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

Extends: "lifetable"
```

The following methods have been defined for the '`lifetable`' and '`actuarialtable`' classes.

```
R> showMethods(classes=c("actuarialtable","lifetable"))

Function "%&%":
 <not an S4 generic function>

Function ".DollarNames":
 <not an S4 generic function>

Function "AIC":
 <not an S4 generic function>

Function "BIC":
 <not an S4 generic function>

Function "BunchKaufman":
 <not an S4 generic function>

Function "Cholesky":
 <not an S4 generic function>

Function "Schur":
 <not an S4 generic function>

Function "absorbingStates":
 <not an S4 generic function>

Function "absorptionProbabilities":
 <not an S4 generic function>

Function "aggregateStates":
 <not an S4 generic function>

Function "all.equal":
 <not an S4 generic function>
```

```
Function "as.array":
 <not an S4 generic function>

Function "as.matrix":
 <not an S4 generic function>

Function "asJSON":
 <not an S4 generic function>

Function "autoLump":
 <not an S4 generic function>

Function "band":
 <not an S4 generic function>

Function "canonicForm":
 <not an S4 generic function>

Function "chol":
 <not an S4 generic function>

Function "chol2inv":
 <not an S4 generic function>

Function "closestReversible":
 <not an S4 generic function>

Function "coef":
 <not an S4 generic function>
Function: coerce (package methods)
from="actuarialtable", to="data.frame"
from="actuarialtable", to="numeric"
from="data.frame", to="lifetable"
from="lifetable", to="data.frame"
from="lifetable", to="markovchainList"
from="lifetable", to="numeric"


Function "colMeans":
 <not an S4 generic function>

Function "colSums":
 <not an S4 generic function>

Function "communicatingClasses":
 <not an S4 generic function>
```

```
Function "complete":
 <not an S4 generic function>

Function "conditionalDistribution":
 <not an S4 generic function>

Function "confint":
 <not an S4 generic function>

Function "cov2cor":
 <not an S4 generic function>

Function "determinant":
 <not an S4 generic function>

Function "diag":
 <not an S4 generic function>

Function "diag<-":
 <not an S4 generic function>

Function "diff":
 <not an S4 generic function>

Function "drop":
 <not an S4 generic function>
Function: dxt (package lifecontingencies)
object="lifetable"


Function "entropyRate":
 <not an S4 generic function>

Function "expand":
 <not an S4 generic function>

Function "expand1":
 <not an S4 generic function>

Function "expand2":
 <not an S4 generic function>

Function "expm":
 <not an S4 generic function>

Function "facmul":
```

```
 <not an S4 generic function>

Function "forceSymmetric":
 <not an S4 generic function>

Function "formals<-":
 <not an S4 generic function>

Function "format":
 <not an S4 generic function>

Function "functions":
 <not an S4 generic function>
Function: getOmega (package lifecontingencies)
object="lifetable"

Function: head (package utils)
x="lifetable"


Function "hittingProbabilities":
 <not an S4 generic function>

Function "image":
 <not an S4 generic function>

Function "impliedTimescales":
 <not an S4 generic function>
Function: initialize (package methods)
.Object="actuarialtable"
.Object="lifetable"


Function "is.accessible":
 <not an S4 generic function>

Function "is.irreducible":
 <not an S4 generic function>

Function "is.lumpable":
 <not an S4 generic function>

Function "is.regular":
 <not an S4 generic function>

Function "is.reversible":
 <not an S4 generic function>
```

```
Function "is.stochasticallyMonotone":
 <not an S4 generic function>

Function "isDiagonal":
 <not an S4 generic function>

Function "isSymmetric":
 <not an S4 generic function>

Function "isTriangular":
 <not an S4 generic function>

Function "kemenyConstant":
 <not an S4 generic function>

Function "lazyChain":
 <not an S4 generic function>

Function "logLik":
 <not an S4 generic function>

Function "lu":
 <not an S4 generic function>

Function "lump":
 <not an S4 generic function>

Function "mean":
 <not an S4 generic function>

Function "meanAbsorptionTime":
 <not an S4 generic function>

Function "meanFirstPassageTime":
 <not an S4 generic function>

Function "meanNumVisits":
 <not an S4 generic function>

Function "meanRecurrenceTime":
 <not an S4 generic function>
Function: median (package stats)
x="lifetable"


Function "mergeWith":
```

```
 <not an S4 generic function>

Function "mixingTime":
 <not an S4 generic function>

Function "name":
 <not an S4 generic function>

Function "name<-":
 <not an S4 generic function>

Function "nnzero":
 <not an S4 generic function>

Function "nobs":
 <not an S4 generic function>

Function "norm":
 <not an S4 generic function>

Function "normalizedEntropyRate":
 <not an S4 generic function>

Function "pack":
 <not an S4 generic function>
Function: plot (package base)
x="lifetable", y="ANY"


Function "predict":
 <not an S4 generic function>
Function: print (package base)
x="actuarialtable"
x="lifetable"


Function "profile":
 <not an S4 generic function>

Function "prompt":
 <not an S4 generic function>
Function: pxt (package lifecontingencies)
object="lifetable"


Function "qr":
 <not an S4 generic function>
```

```
Function "qr.Q":
 <not an S4 generic function>

Function "qr.R":
 <not an S4 generic function>

Function "qr.X":
 <not an S4 generic function>

Function "qr.coef":
 <not an S4 generic function>

Function "qr.fitted":
 <not an S4 generic function>

Function "qr.qty":
 <not an S4 generic function>

Function "qr.qy":
 <not an S4 generic function>

Function "qr.resid":
 <not an S4 generic function>
Function: quantile (package stats)
x="lifetable"

Function: qxt (package lifecontingencies)
object="lifetable"


Function "rcond":
 <not an S4 generic function>

Function "recurrentClasses":
 <not an S4 generic function>

Function "recurrentStates":
 <not an S4 generic function>

Function "redistribute":
 <not an S4 generic function>

Function "relaxationTime":
 <not an S4 generic function>

Function "rowMeans":
```

```
 <not an S4 generic function>

Function "rowSums":
 <not an S4 generic function>

Function "sensitivity":
 <not an S4 generic function>
Function: show (package methods)
object="actuarialtable"
object="lifetable"


Function "skewpart":
 <not an S4 generic function>

Function "slem":
 <not an S4 generic function>

Function "solve":
 <not an S4 generic function>

Function "sort":
 <not an S4 generic function>

Function "spectralGap":
 <not an S4 generic function>

Function "states":
 <not an S4 generic function>

Function "steadyStates":
 <not an S4 generic function>

Function "subchain":
 <not an S4 generic function>
Function: summary (package base)
object="actuarialtable"
object="lifetable"


Function "symmpart":
 <not an S4 generic function>

Function "t":
 <not an S4 generic function>
Function: tail (package utils)
x="lifetable"
```

```
Function "timeCorrelations":
 <not an S4 generic function>

Function "timeRelaxations":
 <not an S4 generic function>

Function "toBoundedChain":
 <not an S4 generic function>

Function "toDictionary":
 <not an S4 generic function>

Function "toDot":
 <not an S4 generic function>

Function "toFile":
 <not an S4 generic function>

Function "toMermaid":
 <not an S4 generic function>

Function "toeplitz":
 <not an S4 generic function>

Function "topologicalEntropy":
 <not an S4 generic function>

Function "transientClasses":
 <not an S4 generic function>

Function "transientStates":
 <not an S4 generic function>

Function "transitionProbability":
 <not an S4 generic function>

Function "tril":
 <not an S4 generic function>

Function "triu":
 <not an S4 generic function>

Function "unname":
 <not an S4 generic function>
```

Parameter	Significance
`x`	the policyholder's age
`n`	the coverage duration or payment duration
`i`	interest rate (could be varying)
`k`	the frequency of payments

Table 1: **lifecontingencies** functions parameters naming conventions.

```
Function "unpack":
 <not an S4 generic function>

Function "update":
 <not an S4 generic function>

Function "updown":
 <not an S4 generic function>

Function "vcov":
 <not an S4 generic function>

Function "which":
 <not an S4 generic function>

Function "writeMM":
 <not an S4 generic function>

Function "zapsmall":
 <not an S4 generic function>
```

The computation of financial, demographic and actuarial quantities is based on dedicated functions that use objects of classes '`lifetable`' and '`actuarialtable`' when required. Table 1 shows the naming convention for common input parameters used within the package. The sections that follow briefly present such functions with the aid of examples.

Finally, the **lifecontingencies** package depends on the **methods** package, which defines its classes, and the **parallel** package, which is used to speed up computations. As detailed in Section 6, implementation of C or C++ code snippets is expected to shorten computational times in the future versions of the package.

4. Code and examples

This section is structured as follows: Section 4.1 shows classical financial mathematics examples, Section 4.2 deals with life tables and actuarial table management, Section 4.4 shows classical actuarial mathematics examples, and Section 4.5 presents functions in the **lifecontingencies** package that perform simulation analysis.

4.1. Classical financial mathematics example

The **lifecontingencies** package provides functions that perform classical financial mathematics calculations listed in Table 2.

Some of these implement closed form formulas and their inverses; this is also shown in financial mathematics textbooks. A broader discussion, however, shall be dedicated to the `presentValue` function. In fact, the `presentValue` function is internally called by most other financial and actuarial functions within the **lifecontingencies** package. This function calculates present values or APVs by computing Equation 3.

$$\text{PV} = \sum_{i=1}^{n} c_i \cdot v^{t_i} \cdot p_i \tag{3}$$

The terms in Equation 3 are the cash flows vector, c_i, the corresponding discount factors vector, v^{t_i}, and the occurrence probabilities vector, p_i, respectively. Many **lifecontingencies** package functions, like `axn` or `annuity`, work by first defining the pattern vectors of cash flows, interest rate and probabilities (in case of actuarial functions), which are then passed as arguments to the `presentValue` function.

Examples that follow show how to handle interest and discount rates with different compounding frequencies, how to perform present value, annuity, and future value analysis, and how to deal with loan amortization and bond pricing.

Interest rate functions

Interest rates represent the time-value of money. Different types of rates can be found in literature. As a remark, Equation 4 displays the relationship between the effective interest rate, the convertible interest rate, the discount factor, the force of interest, the effective discount rate and the convertible discount rate.

$$(1+i)^t = \left(1 + \frac{i^{(m)}}{m}\right)^t = v^{-t} = \exp(\delta t) = (1-d)^{-t} = \left(1 - \frac{d^{(m)}}{m}\right)^{-t} \tag{4}$$

The functions `interest2Discount`, `discount2Interest`, `convertible2Effective`, `effective2Convertible`, `interest2Intensity`, `intensity2Interest` have all been based

Function	Purpose
`presentValue`	present value of a series of cash flows
`annuity`	present value of an annuity-certain, $a_{\overline{n}\rceil}$
`accumulatedValue`	future value of a series of cash flows, $s_{\overline{n}\rceil}$
`increasingAnnuity`	present value of an increasing annuity – certain, IA_n
`decreasingAnnuity`	present value of a decreasing annuity – certain, $DA_{\overline{n}\rceil}$
`convertible2Effective`	conversion from convertible to effective interest (discount) rates
`effective2Convertible`	`convertible2Effective` inverse
`intensity2Interest`	conversion from force of interest to the interest rate
`interest2Intensity`	`intensity2Interest` inverse
`duration`	dollar / Macaulay duration of a series of cash flows
`convexity`	convexity of a series of cash flows

Table 2: **lifecontingencies** functions for financial mathematics.

on Equation 4; their inverse formulas are implied therein. Throughout the paper, an effective interest rate is used unless otherwise stated.

As examples, functions `interest2Discount` and `discount2Interest` represent a convenient way to switch from interest to discount rates and vice versa.

```
R> interest2Discount(0.03)

[1] 0.02912621

R> discount2Interest(interest2Discount(0.03))

[1] 0.03
```

The function `convertible2Effective` allows one to find the effective interest rate implied in a consumer - credit loan that offers a 10% convertible (nominal) interest rate with quarterly compounding.

```
R> convertible2Effective(i=0.10,k=4)

[1] 0.1038129
```

Analysis of present value and internal rate of return

Performing a project appraisal means evaluating the net present value (NPV) of all projected cash flows. The code below shows an example of NPV analysis.

```
R> capitals <- c(-1000,200,500,700)
R> times <- c(0,1,2,5)
R> presentValue(cashFlows=capitals, timeIds=times, interestRates=0.03)

[1] 269.2989
```

When interest rates vary and cash flows are uncertain, the `probabilities` parameter can be properly set as the following code shows:

```
R> presentValue(cashFlows=capitals, timeIds=times,
+  interestRates=c( 0.04, 0.02, 0.03, 0.05),
+  probabilities=c(1,1,1,0.5))

[1] -58.38946
```

The internal rate of return (IRR) is defined as the interest rate that makes the NPV zero. It is an alternative NPV that allows financial investment projects to be ranked by the timing and amount of their cash flows. The following example displays how the **lifecontingencies** package can use base R functions to calculate the IRR.

```
R> getIrr <- function(p) (presentValue(cashFlows=capitals, timeIds=times,
+  interestRates=p) - 0)^2
R> nlm(f=getIrr, p=0.1)$estimate

[1] 0.1105091
```

Annuities and future values

An annuity (certain) is a sequence of payments with a specified amount that is present valued. If it is valued at the end of the term of payment is is called future value (or accumulated value). The code below shows examples of annuity, $a_{\overline{n}|}$, and accumulated value, $s_{\overline{n}|}$, evaluations. The PV of an annuity immediate \$100 payable at the end of each year for the next 5 years at an interest rate of 3% is:

```
R> 100 * annuity(i=0.03, n=5)

[1] 457.9707
```

while the corresponding future value is:

```
R> 100 * accumulatedValue(i=0.03, n=5)

[1] 530.9136
```

Annuities and future values payable k-thly (where fractional payments of $\frac{1}{k}$ are received for each k-th of a period) can be evaluated properly by setting the functions' parameters as follows:

```
R> ann1 <- annuity(i=0.03, n=5, k=1, type="immediate")
R> ann2 <- annuity(i=0.03, n=5, k=12, type="immediate")
R> c(ann1,ann2)

[1] 4.579707 4.642342
```

`increasingAnnuity` and `decreasingAnnuity` functions handle increasing and decreasing annuities, whose symbols are IA_x, DA_x respectively. Assuming a ten-year term and a 3% interest rate, examples of increasing and decreasing annuities follow.

```
R> incrAnn <- increasingAnnuity(i=0.03, n=10, type="due")
R> decrAnn <- decreasingAnnuity(i=0.03, n=10, type="immediate")
R> c(incrAnn, decrAnn)

[1] 46.18416 48.99324
```

The last example within this section displays the calculation of the present value of a geometrically increasing annuity. As known by classical financial and actuarial mathematic, geometric annuities are priced using a synthetic discount rate $j = i - g$ being g the geometric growth rate and i the interest rate used to discount future payments. So, if payment amounts increase by 3%, the interest rate is 4%, and its term is 10 years, the implied present value is:

```
R> annuity(i=((1+0.04)/(1+0.03)-1), n=10)

[1] 9.48612
```

Loan amortization

As this section exemplifies, **lifecontingencies** financial mathematics functions allow one to define the repayment schedule of any loan arrangement. Let C denote the loaned capital (principal). Assuming an interest rate i, the amount due to the lender at each installment is $R = \frac{C}{a_{\overline{n}|}}$. Therefore, the R amount repays $I_t = C_{t-1} * i$ as interest, while $C_t = R - I_t$ is the amount of the loan still outstanding after installment t has been paid. The periodic installment of loan repayment , R, is calculated as follows:

```
R> capital <- 100000
R> interest <- 0.05
R> payments_per_year <- 2
R> rate_per_period <- (1+interest)^(1/payments_per_year)-1
R> years <- 30
R> R <- 1/payments_per_year *
+  capital/annuity(i=interest, n=years,
+                  k=payments_per_year)
R> R

[1] 3212.9
```

Then the balance due at end of the period (EoP) is calculated as follows:

```
R> balanceDue <- numeric(years * payments_per_year)
R> balanceDue[1] <- capital * (1+rate_per_period) - R
R> for(i in 2:length(balanceDue)) balanceDue[i]<-
+    balanceDue[i-1] * (1+rate_per_period) - R
```

Figure 1 shows the EoP balance due for a 30 year loan, assuming a 5% interest rate on a principal of $ 100,000.

Bond pricing

Bond pricing represents another application of present value. A standard bond with face value C and term length T consists of equal coupons c paid at regular intervals. The final payment at time T is $C_T + c$. Equation 5 expresses the present value of a bond with n remaining coupons.

$$B = c * a_{\overline{n}|} + C * v^T \tag{5}$$

Perpetuities are financial contracts that offer an indefinite sequence of payments either at the end (perpetuity-immediate) or at the beginning of each period (perpetuity-due).

The following examples show how elementary functions in the **lifecontingencies** package can be combined to price bonds and perpetuities.

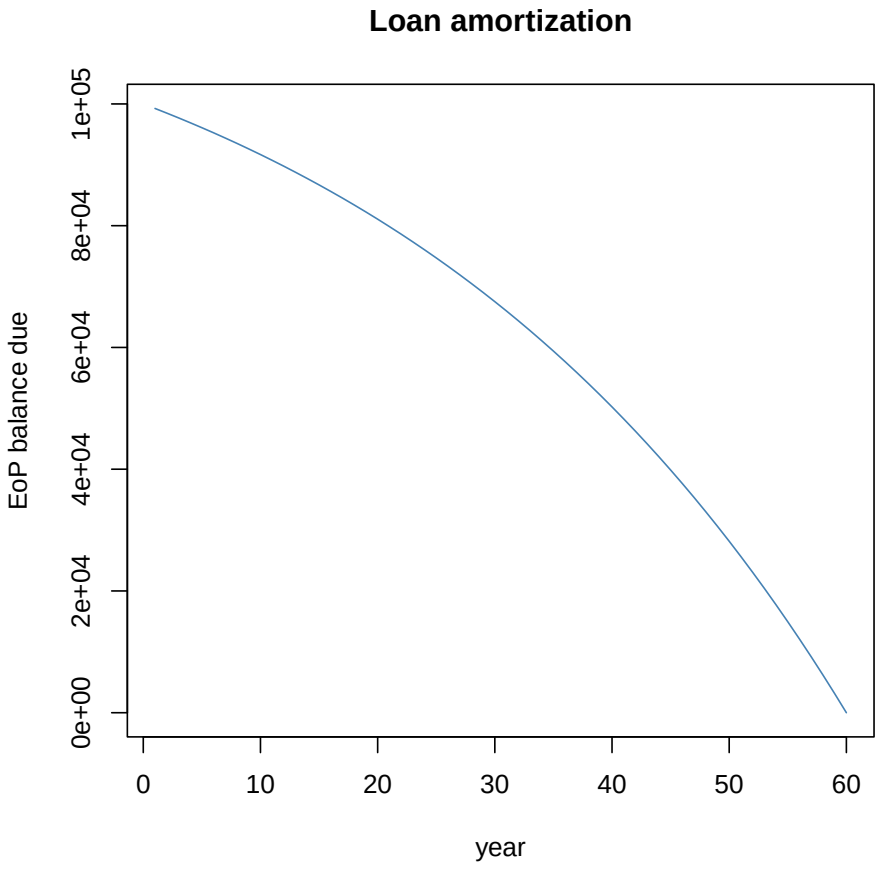


Figure 1: Loan amortization: EoP balance due.

```
R> bond<-function(faceValue, couponRate, couponsPerYear, yield,maturity)
+  {
+          out <- numeric(1)
+          numberOfCF <- maturity * couponsPerYear
+          CFs <- numeric(numberOfCF)
+          payments <- couponRate * faceValue / couponsPerYear
+          cf <- payments * rep(1,numberOfCF)
+          cf[numberOfCF] <- faceValue + payments
+          times <- seq.int(from=1/couponsPerYear, to=maturity,
+                  by=maturity/numberOfCF)
+          out <- presentValue(cashFlows=cf, interestRates=yield,
+                     timeIds=times)
+          return(out)
+  }
R> perpetuity<-function(yield, immediate=TRUE)
+  {
+          out <- numeric(1)
+          out <- 1 / yield
+          out <- ifelse(immediate==TRUE, out, out*(1+yield))
+          return(out)
+  }
R>
```

As displayed below, the `bond` and `perpetuity` functions defined above can be used to price any bond, given face value, coupon rate, and term.

```
R> bndEx1 <-bond(1000, 0.06, 2, 0.05, 3)
R> bndEx2 <-bond(1000, 0.06, 2, 0.06, 3)
R> ppTy1 <-perpetuity(0.1)
R> c(bndEx1, bndEx2, ppTy1)

[1] 1029.250 1002.371   10.000
```

Duration and ALM

As defined within the package, duration and convexity formulas are reported in Equation 6 and Equation 7 respectively. Their typical application lies within porfolios' asset - liability management (ALM). The interested reader can find details in Chris Ruckman and Joe Francis (2006) and Broverman (2008) textbooks. However, the following example shows how the Macaulay duration (`ex1`), modified duration (`ex2`), and convexity (`ex3`) of any series of cash flows can be calculated by **lifecontingencies** package functions.

$$D = \sum_{t}^{T} \frac{t * \mathrm{CF}_t \left(1 + \frac{i}{m}\right)^{-t*m}}{P} \tag{6}$$

$$C = \sum_{t}^{T} t * \left(t + \frac{1}{m}\right) * \mathrm{CF}_t \left(1 + \frac{y}{m}\right)^{-m*t-2} \tag{7}$$

```
R> cashFlows <- c(100,100,100,600,500,700)
R> timeVector <- seq(1:6)
R> interestRate <- 0.03
R> dur1 <-duration(cashFlows = cashFlows, timeIds = timeVector,
+                  i = interestRate, k = 1, macaulay = TRUE)
R> dur2 <-duration(cashFlows = cashFlows, timeIds = timeVector,
+                  i = interestRate, k = 1, macaulay = FALSE)
R> cvx1 <-convexity(cashFlows = cashFlows, timeIds = timeVector,
+                  i = interestRate, k = 1)
R> c(dur1, dur2, cvx1)

[1]  4.563124  4.430218 25.746469
```

This example works out a small ALM problem. Suppose an insurance company has sold a guaranteed term certificate (GTC) with face value $ 10,000 that will mature in 7 years at an interest rate of 5% . Its final value would be:

```
R> GTCFin<- 10000 * (1 + 0.05)^7
R> GTCFin

[1] 14071
```

Imagine the company can hedge its liability with two available investment instruments:

1. A five year bond with face value of 100 and 3% coupons paid annually.
2. A perpetuity-immediate. As a remark, the formulas for the PV, duration and convexity of a perpetuity immediate are $PV_{\mathrm{pp}} = \frac{1}{y}$, $D_{\mathrm{pp}} = \frac{1+y}{y}$, $C_{\mathrm{pp}} = \frac{2}{y^2}$ respectively, if the yield rate is y.

Assume the issuing company wants to hedge its liability with an investment portfolio that will not be impacted by changes in the investment yield. In order to solve the ALM problem, the composition of assets within the portfolio shall be chosen accordingly. Moreover, assume that the current market yield rate is 4%. The following lines of code figure out some parameters that are used within the example.

```
R> yieldT0 <- 0.04
R> durLiab <- 7
R> pvLiab <- presentValue(cashFlows = GTCFin,timeIds = 7,
+                  interestRates = yieldT0)
R> convLiab <- convexity(cashFlows=GTCFin, timeIds = 7,
+                  i=yieldT0)
R> pvBond <- bond(100,0.03,1,yieldT0,5)
```

```
R> durBond <- duration(cashFlows=c(3,3,3,3,103),
+                  timeIds=seq(1,5), i = yieldT0)
R> convBond <- convexity(cashFlows=c(3,3,3,3,103),
+                  timeIds=seq(1,5), i = yieldT0)
R> pvPpty <- perpetuity(yieldT0)
R> durPpty <- (1+yieldT0)/yieldT0
R> covnPpty <- 2/(yieldT0^2)
```

Then the ALM problem can be set up as a three step problem, as the Chris Ruckman and Joe Francis (2006) textbook remarks:

1. Setting the initial present value of cash inflows (assets) to be equal to the present value of cash outflows (liabilities).

2. Setting the interest rate sensitivity (i.e., the duration) of assets to be equal to the interest rate sensitivity of liabilities. This is done by solving the system of equations shown in Equation 8. The parameters w_i and D_i stand for asset hedging weights and duration values respectively.

$$\begin{cases} w_{\text{bnd}} D_{\text{bnd}} + w_{\text{ppt}} D_{\text{ppt}} = D_{\text{GTC}} \\ w_{\text{bnd}} + w_{\text{ppt}} = 1 \end{cases} \tag{8}$$

3. Setting the convexity of assets to be greater than the convexity of liabilities. In other words, this means verifying that asset decline (growth) will be slower (faster) than liability decline in case of a change in the interest rate.

The following lines of code calculate the asset weights vector by linear algebra functions bundled in R base.

```
R> a <- matrix(c(durBond, durPpty,1,1), nrow=2,
+                  byrow=TRUE)
R> b <- as.vector(c(7,1))
R> weights <-solve(a,b)
R> weights

[1] 0.8924163 0.1075837
```

Vector `weights` displays the portfolio composition in terms of bonds and perpetuities, respectively. Therefore, the number of bonds and perpetuities that can be purchased is determined by:

```
R> bondNum <- weights[1] * pvLiab / pvBond
R> pptyNum <- weights[2] * pvLiab / pvPpty
R> bondNum

[1] 99.87041
```

```
R> pptyNum

[1] 46.01486
```

It can be verified that the convexity of assets is greater than the convexity of liabilities.

```
R> convAsset <- weights[1] * convBond + weights[2] * covnPpty
R> convAsset>convLiab

[1] TRUE
```

The portfolio is immunized from yield rate variations because the present value of assets will be greater than the present value of the liabilities if the interest rate suddently drops to 3% just after hedging the asset purchase. The same occurs in case of an upward shift in the interest rate toward 5%.

```
R> yieldT1low <- 0.03
R> immunizationTestLow <- (bondNum * bond(100,0.03,1,yieldT1low,5) +
+                          pptyNum * perpetuity(yieldT1low)>
+                          GTCFin / (1+yieldT1low)^7)
R> yieldT1high <- 0.05
R> immunizationTestHigh <- (bondNum * bond(100,0.03,1,yieldT1high,5) +
+                          pptyNum * perpetuity(yieldT1high)>
+                          GTCFin/(1+yieldT1high)^7)
R> immunizationTestLow

[1] TRUE

R> immunizationTestHigh

[1] TRUE
```

It is worthwhile to remember that asset allocation within the portfolio should be rebalanced with some frequency, since the portfolio's duration and convexity will change as time goes on.

4.2. Analysis of life tables and actuarial tables

`lifetable` and `actuarialtable` classes are designed to handle demographic and actuarial mathematics calculations. An `actuarialtable` class inherits from `lifetable` class; it adds one more slot for the rate of interest. Both classes have been designed using the `S4` R classes framework.
Table 3 lists the functions that have been developed for performing demographic analysis within **lifecontingencies** package. This section briefly exemplifies these functions.

Creating lifetable and actuarialtable objects

Life table objects can be created by using either raw R commands or existing `data.frame` objects. However, three components are needed to build a `lifetable` class object:

Function	Purpose
`dxt`	deaths between age x and $x+t$, ${}_td_x$.
`pxt`	survival probability between age x and $x+t$, ${}_tp_x$.
`pxyzt`	survival probability for two (or more) lives, ${}_tp_{xy}$.
`qxt`	death probability between age x and $x+t$, ${}_tq_x$.
`qxyzt`	death probability for two (or more) lives, ${}_tq_{xy}$.
`Txt`	number of person-years lived after exact age x, ${}_tT_x$.
`mxt`	central death rate, ${}_tm_x$.
`qx2mx`	convert death probabilities into mortality rate.
`mx2qx`	convert mortality rate into death probabilities.
`exn`	expected lifetime between age x and age $x+n$, ${}_ne_x$.
`rLife`	sample from the time until death distribution underlying a life table.
`rLifexyz`	sample from the time until death distribution underlying two or more lives.
`exyz`	n-year curtate lifetime of the joint-life status.
`probs2lifetable`	life table l_x from raw one - year survival / death probabilities.

Table 3: **lifecontingencies** functions for demographic analysis.

1. The years sequence, which is an integer sequence $0, 1, \ldots, \omega$. It shall start from zero and end at ω, the terminal age (the age x for which $p_x = 0$).
2. The l_x vector, which is the number of subjects living at the beginning of age x; in other words, the number of subjects at risk of dying between year x and $x+1$.
3. The name of the life table.

There are three main approaches for creating a `lifetable` object:

1. Directly from the x and l_x vector.
2. By importing x and l_x from an existing `data.frame` object.
3. From using raw survival probabilities.

Creating a `lifetable` object directly can be done as shown by the code below:

```
R> x_example <- seq(from=0,to=9, by=1)
R> lx_example <- c(1000,950,850,700,680,600,550,400,200,50)
R> exampleLt <- new("lifetable", x=x_example, lx=lx_example,
+                   name="example lifetable")
```

`print` and `show` methods tabulate the x, l_x, ${}_tp_x$ and e_x values for a given life table.

```
R> print(exampleLt)

Life table example lifetable

  x   lx        px  Lx   Tx       ex
```

```
1 0 1000 0.9500000 975 5480 4.980000
2 1  950 0.8947368 900 4505 4.242105
3 2  850 0.8235294 775 3605 3.741176
4 3  700 0.9714286 690 2830 3.542857
5 4  680 0.8823529 640 2140 2.647059
6 5  600 0.9166667 575 1500 2.000000
7 6  550 0.7272727 475  925 1.181818
8 7  400 0.5000000 300  450 0.625000
9 8  200 0.2500000 125  150 0.250000
```

`head` and `tail` methods for `data.frame` S3 classes have also been implemented on `lifetable` classes.

```
R> head(exampleLt)
```

```
  x   lx
1 0 1000
2 1  950
3 2  850
4 3  700
5 4  680
6 5  600
```

Still, the easiest way to create a `lifetable` object is to start from a suitable existing `data.frame`. This will probably be the most practical approach for practicing actuaries. Some life or mortality rate tables have been bundled within the **lifecontingencies** package, as Table 4 displays.

Data set	Description
`AF92Lt`	UK AF92 life table.
`AM92Lt`	UK AF92 life table.
`de_angelis_di_falco`	List containing ten data frames showing projected mortality rates for healty and
`demoChina`	China mortality rates from SOA website.
`demoIta`	Various Italian life tables including RG48 and IPS55 projected tables.
`demoJapan`	Japan mortality rates from SOA website.
`demoUsa`	US Social Security life tables.
`demoFrance`	1990 and 2002 French life tables.
`demoCanada`	UP94 (standard, 2015, 2020) mortality rates for males and females.
`soa08`	SOA illustrative life table.
`soa08Act`	SOA illustrative actuarial table at 6%.

Table 4: Life tables and other data objects bundled within **lifecontingencies**.

The following example shows how US Social Security life tables are loaded from the existing `demoUsa` data set bundled in the **lifecontingencies** package.

```
R> data("demoUsa")
R> data("demoIta")
```

```
R> usaMale07 <- demoUsa[,c("age", "USSS2007M")]
R> usaMale00 <- demoUsa[,c("age", "USSS2000M")]
R> names(usaMale07) <- c("x","lx")
R> names(usaMale00) <- c("x","lx")
R> usaMale07Lt <-as(usaMale07,"lifetable")
R> usaMale07Lt@name <- "USA MALES 2007"
R> usaMale00Lt <-as(usaMale00,"lifetable")
R> usaMale00Lt@name <- "USA MALES 2000"
```

The same operation can be performed on IPS55 tables bundled in the `demoIta` data set. The purpose of the following example is to stress the importance of using a clean l_x series as an input for the coerce method. A "clean" l_x series is a decreasing series without zeroes or missing values.

```
R> lxIPS55M <- with(demoIta, IPS55M)
R> pos2Remove <- which(lxIPS55M %in% c(0,NA))
R> lxIPS55M <-lxIPS55M[-pos2Remove]
R> xIPS55M <-seq(0,length(lxIPS55M)-1,1)
R> ips55M <- new("lifetable",x=xIPS55M, lx=lxIPS55M,
+                  name="IPS 55 Males")
R> lxIPS55F <- with(demoIta, IPS55F)
R> pos2Remove <- which(lxIPS55F %in% c(0,NA))
R> lxIPS55F <- lxIPS55F[-pos2Remove]
R> xIPS55F <- seq(0,length(lxIPS55F)-1,1)
R> ips55F <- new("lifetable",x=xIPS55F, lx=lxIPS55F,
+                  name="IPS 55 Females")
```

The final method of creating a `lifetable` object uses one year survival or death probabilities, combining the `probs2lifetable` function with `as.data.frame` coerce methods. Two potential benefits arise from using this function. The first benefit lies in the use of mortality projection method results. The Lee - Carter method (Lee and Carter 1992) allows one to vary mortality table by cohort of birth. Thus, demographic quantities, like the expected lifetime, e_0,can be projected as functions of the year of birth.
A second advantage lies in the creation of "cut-down" mortality tables. This latter application is exemplified in the code that follows, where a `itaM2002reduced` life table is obtained; the one - year mortality rates of Italian males aged between 20 and 60 are cut down to 20% of their original value.

```
R> data("demoIta")
R> itaM2002 <- demoIta[,c("X","SIM92")]
R> names(itaM2002) <- c("x","lx")
R> itaM2002Lt <- as(itaM2002,"lifetable")
R> itaM2002Lt@name <- "IT 2002 Males"
R> itaM2002 <- as(itaM2002Lt,"data.frame")
R> itaM2002$qx <- 1-itaM2002$px
R> for(i in 20:60) itaM2002$qx[itaM2002$x==i] = 0.2 * itaM2002$qx[itaM2002$x==i]
R> itaM2002reduced <- probs2lifetable(probs=itaM2002[,"qx"], radix=100000,
+                  type="qx",name="IT 2002 Males reduced")
```

An `actuarialtable` can be easily created from an existing `lifetable` object.

```
R> exampleAct <- new("actuarialtable",x=exampleLt@x, lx=exampleLt@lx,
+  interest=0.03, name="example actuarialtable")
```

When applied to either `actuarialtable` or `lifetable` classes, Method `getOmega` returns the terminal age, ω.

```
R> getOmega(exampleAct)
```

```
[1] 9
```

Method `print` behaves differently for `lifetable` and `actuarialtable` objects. In fact, when the `print` method is applied on a `lifetable` object, it tabulates both the one year survival probability and the curtate expected remaining life until death (the `ex` column, as returned by `exn(..., type = "curtate")`; the complete expectation is `exn(..., type = "complete")`, see below). Conversely, when the `print` method is applied on an `actuarialtable` object, classical commutation functions (D_x, N_x, C_x, M_x, R_x), discussed later, are printed out.

```
R> print(exampleLt)
```

```
Life table example lifetable

  x   lx        px  Lx   Tx       ex
1 0 1000 0.9500000 975 5480 4.980000
2 1  950 0.8947368 900 4505 4.242105
3 2  850 0.8235294 775 3605 3.741176
4 3  700 0.9714286 690 2830 3.542857
5 4  680 0.8823529 640 2140 2.647059
6 5  600 0.9166667 575 1500 2.000000
7 6  550 0.7272727 475  925 1.181818
8 7  400 0.5000000 300  450 0.625000
9 8  200 0.2500000 125  150 0.250000
```

```
R> print(exampleAct)
```

```
Actuarial table  example actuarialtable interest rate  3 %

   x   lx         Dx         Nx        Cx       Mx        Rx
1  0 1000 1000.00000 5467.92787  48.54369 840.7400 4839.7548
2  1  950  922.33010 4467.92787  94.25959 792.1963 3999.0148
3  2  850  801.20652 3545.59778 137.27125 697.9367 3206.8185
4  3  700  640.59916 2744.39125  17.76974 560.6654 2508.8819
5  4  680  604.17119 2103.79209  69.00870 542.8957 1948.2164
6  5  600  517.56527 1499.62090  41.87421 473.8870 1405.3207
```

```
7  6  550  460.61634  982.05563 121.96373 432.0128  931.4337
8  7  400  325.23660  521.43929 157.88185 310.0491  499.4210
9  8  200  157.88185  196.20268 114.96251 152.1672  189.3719
10 9   50   38.32084   38.32084  37.20470  37.2047   37.2047
```

It is possible to convert the `actuarialtable` object into a `data.frame` object, as shown below.

```
R> exampleActDf <- as(exampleAct, "data.frame")
```

A recent **lifecontingencies** package enhancement allows to export a life table as non - homogeneous discrete Markov chain by means of `markovchainList` S4 class object as defined in **markovchain** (Spedicato, Giorgio Alfredo 2015) R package.

```
R> data(soa08)
R> if (requireNamespace("markovchain", quietly = TRUE) && methods::canCoerce(soa08, "marko
+    soa08Mc<-as(soa08,"markovchainList")
+  } else {
+    message("'markovchain' package or S4 coercion method unavailable: skipping lifetable
+  }
```

Finally, a `plot` method can be applied to either`lifetable` or `actuarialtable` objects. The underlying survival function (which is the plot of x vs l_x) is displayed in both cases. Figure 2 shows the `plot` methods applied on a Society of Actuaries (SOA) actuarial table at 6% interest, which comes bundled within the **lifecontingencies** package as `soa08Act` object.

4.3. Visualising life tables

Life tables can be visualized directly using the `autoplot()` method provided by the package. The method follows the `autoplot()` interface used by **ggplot2** (Wickham 2016), and returns a `ggplot` object. This makes it possible to obtain a graphical representation of the number of survivors at each age while retaining the usual **ggplot2** facilities for further customization.

For example, the following code creates a simple life table and plots the number of survivors as a function of age:

```
R> library(lifecontingencies)
R> lt <- new("lifetable",
+            x = 0:3,
+            lx = c(100, 90, 50, 10))
R> ggplot2::autoplot(lt)
```

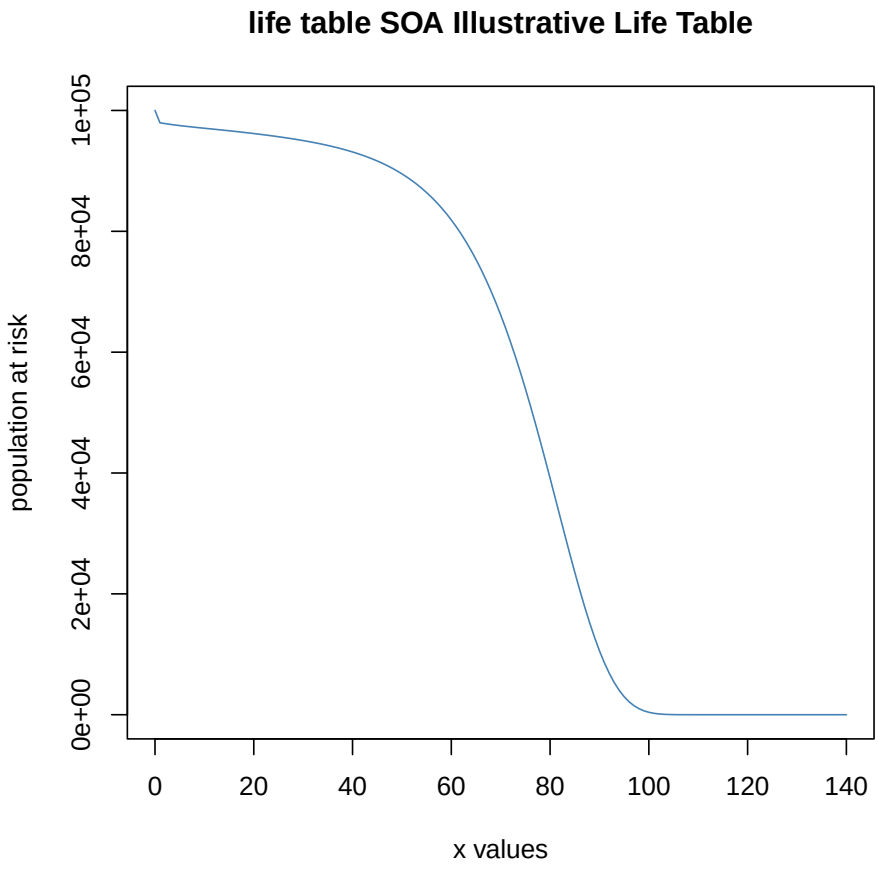
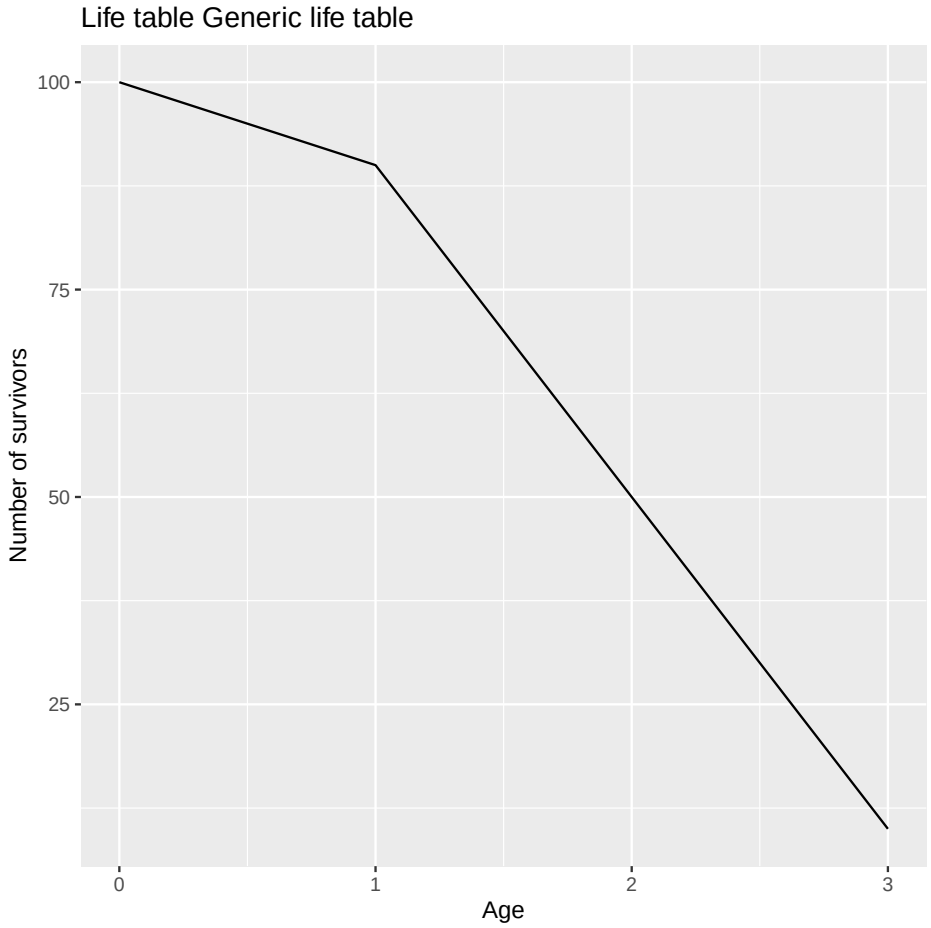
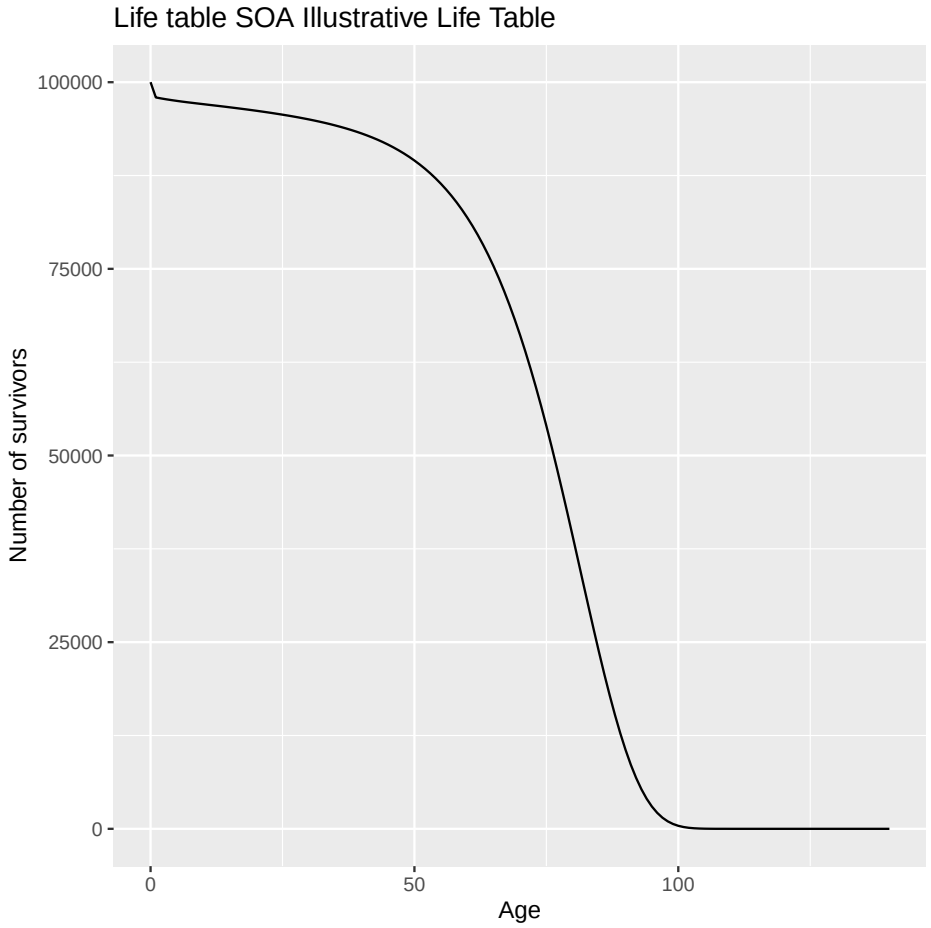


Figure 2: Underlying survival function of SOA illustrative life table.



The same method is available for `actuarialtable` objects because `actuarialtable` is defined as a subclass of `lifetable`. Consequently, no separate plotting method is required for actuarial tables:

```
R> ggplot2::autoplot(soa08Act)
```



The plot focuses on the demographic component of the table, namely the survivor function represented by l_x. The interest rate stored in an `actuarialtable` affects actuarial calculations but does not alter this basic demographic visualisation. Additional arguments supplied through `...` are passed to `geom_line()`, allowing the graphical appearance of the curve to be customised using standard **ggplot2** syntax.

The method therefore provides a convenient bridge between the life-contingencies classes of the package and the broader **ggplot2** visualisation ecosystem.

Basic demographic analysis

Basic demography calculations can be performed on valid `lifetable` or `actuariatable` objects. The functions discussed in this section calculate proper ratios or sums on l_x or d_x values using functions that access to the lifetable object slots. This is all done in accordance with the demographic formula definitions.

The code below shows how ${}_1p_{20}$, ${}_2q_{30}$ and $\mathring{e}_{50:\overline{20}|}$ (the 20-year temporary complete expectation of life at age 50) are calculated respectively on the IPS55 male population table

```
R> demoEx1<-pxt(ips55M,20,1)
R> demoEx2<-qxt(ips55M,30,2)
R> demoEx3<-exn(ips55M, 50,20,"complete")
R> c(demoEx1,demoEx2,demoEx3)

[1]  0.999595096  0.001332031 19.472765230
```

Getting mortality rates and moving to death probabilities is also possible

```
R> mx20t1 <- mxt(ips55M,20,1)
R> qx20t1 <- mx2qx(mx20t1)
R> c(mx20t1,qx20t1)

[1] 0.0004049862 0.0004049042
```

Beyond these core quantities the package also provides the dispersion of the future lifetime, `varxn` and `sdxn` (the variance and standard deviation of $\tilde{K}_x$ or $\tilde{T}_x$, the second-moment companions of `exn`), and distributional summaries of the age at death exposed through the usual R generics `median` and `quantile`, together with the Lexis modal age at death `modalAge`. The `Tx` and `exn` functions additionally accept an `fxt` argument and an `axOmega` argument that control, respectively, the within-year distribution of deaths and the closure of the last (open) age interval, so that published life tables using the open-interval convention $L_\omega = l_\omega / m_\omega$ can be reproduced exactly. A dedicated *Demographic analysis with the lifecontingencies package* vignette works through all of these on real life tables.

The package allows one to calculate fractional survival probabilities through the use of linear interpolation, constant force of mortality and hyperbolic Balducci's assumptions as shown by the code below.

```
R> data("soa08Act")
R> pxtLin <- pxt(soa08Act,80,0.5,"linear")
R> pxtCnst <- pxt(soa08Act,80,0.5,"constant force")
R> pxtHyph <- pxt(soa08Act,80,0.5,"hyperbolic")
R> c(pxtLin,pxtCnst,pxtHyph)

[1] 0.9598496 0.9590095 0.9581701
```

Calculations for survival probabilities on two (or more) lives can also be performed. As a remark, two different life statuses are defined within the analysis of multiple lives survival: "joint" survival status and "last" survival status. The "joint" survival status exists while all the members of the pool are alive, while the "last" survival status exists until the last member of the pool dies. All calculations assume that the multiple lives are independent. Equation 9 expresses the remaining future lifetime on a couple xy under the joint and last survival status respectively.

$$\begin{aligned} \tilde{T}_{xy} &= \min\left(T_x, T_y\right) \\ \tilde{T}_{\bar{xy}} &= \max\left(T_x, T_y\right) \end{aligned} \tag{9}$$

The following code shows how the joint survival probability, last survival probability, and expected joint lifetime can be evaluated using functions in **lifecontingencies**.

```
R> tablesList <- list(ips55M, ips55F)
R> jsp <- pxyzt(tablesList, x=c(65,63), t=2)
R> lsp <- pxyzt(tablesList, x=c(65,63), t=2, status="last")
R> jelt <- exyzt(tablesList, x=c(65,63), status="joint")
R> c(jsp,lsp,jelt)

[1]  0.9813187  0.9999275 19.1982972
```

4.4. Classical actuarial mathematics examples

Function	Purpose	APV symbol
`Axn`	one life insurance	$A^1_{x:\overline{n}\rceil}$.
`AExn`	the n-year endowment	$A_{x:\overline{n}\rceil}^{\ 1}$.
`Axyzn`	two lives life insurances	$\bar{A}^1_{\overline{xy}:\overline{n}\rceil}$.
`axn`	one life annuity	$\ddot{a}_x$.
`axyzn`	two lives annuities	$\ddot{a}_{xy}$.
`Exn`	pure endowment	${}_nE_x$.
`Iaxn`	increasing annuity	Ia_x.
`IAxn`	increasing life insurance	$(IA)^1_{x:\overline{n}\rceil}$.
`DAxn`	decreasing life insurance	$(DA)^1_{x:\overline{n}\rceil}$.
`rLifeContingencies`	generates variates from the $\tilde{Z}$ distribution.	
`rLifeContingenciesXyz`	multiple lives version of `rLifeContingencies`.	

Table 5: **lifecontingencies** functions for actuarial mathematics.

Table 5 lists examples of functions contained in **lifecontingencies** that allow the user to perform classical actuarial mathematics calculations. In the selection of examples that follow, the SOA illustrative life table with an interest rate of 6% will be used unless otherwise stated

Life insurance examples

The evaluation of the APV has traditionally followed one of three approaches: the use of commutation tables, the current payment technique, or the expected value method.

Commutation tables extend the life table by tabulating special functions of age and rate of interest, as Anderson (1999) further considers. Ratios of commutation table functions allow an actuary to evaluate APV for standard insurances. However, commutation table usage has become less prominent in the computer era. In fact, these tables are not flexible enough and their usage is computationally inefficient. Therefore, the **lifecontingencies** package does not use the commutation table approach to evaluate APVs.
The current payment technique calculates the APV of a life contingency insurance, $\bar{Z}$, as the scalar product of three vectors: $\bar{Z} = \langle \langle \bar{c} \bullet \bar{v} \rangle \bullet \bar{p} \rangle$; this uses the vector of all possible uncertain cash flows, $\bar{c}$, the vector of discount factors, $\bar{v}$, and the vector of cash flow probabilities, $\bar{p}$. The **lifecontingencies** package implements the current payment technique by using actuarial functions listed in Table 5 to evaluate APVs. Finally, the expected value approach models $\bar{Z}$ as the scalar product of two vectors: $\bar{Z} = \left\langle \bar{pk} \bullet \bar{x} \right\rangle$. $\bar{pk}$ is $Pr\left[\tilde{K} = k\right]$, the probability that the future curtate lifetime will be exactly k years, where $\bar{x}$ is the present value of benefits due under the policy term if $\tilde{K} = k$. `rLifeContingencies` and `rLifeContingenciesXyz` implement the expected value approach to generate $\tilde{Z}$ variates.

Consider an n year annuity due. Its APV, $\ddot{a}_{x:\overline{n}|}$, using the commutation table approach is reported in Equation 10, while Equation 11 reports the same APV using the current payment technique. Finally, Equation 12 calculates the APV using the expected value approach.

$$\text{APV} = \frac{N_x - N_{x+n}}{D_x} \tag{10}$$

$$\text{APV} = \sum_{k=0}^{\min(\omega - x, n)} {}_kp_x * v^k \tag{11}$$

$$\text{APV} = \sum_{k=0}^{\omega - x} \Pr\left[\tilde{K}_x = k\right] * \ddot{a}_{\overline{\min(k,n)}|} \tag{12}$$

In order to understand how **lifepackage** implements the current payment technique in its actuarial function, it is worthwhile to look closer at the core of the `axn` function. This function takes the following parameters as inputs: `n`, the term of the annuity; `k` the fractional payment frequency; `x` the annuitant age; `m`, the deferring period. Then, it defines:

1. The vector of possible payments, $\bar{c}$, by

```
payments = rep(1/k, n * k)
```

2. The vector timing of payments, by

```
times=m + seq(from=0, to=(n-1/k), by=1/k)
```

3. The vector of payment probability, $\bar{p}$, by

```
for(i in 1:length(times)) probs[i] = pxt(
actuarialtable, x,times[i])
```

4. Finally, the three vectors are passed as input parameters to the `presentValue` function as the following code shows:

```
presentValue(cashFlows=payments, timeIds=times,
interestRates = interest,
probabilities=probs)
```

In the examples that follow, the SOA illustrative actuarial table is used in the calculation of premiums and reserves of life contingencies.

The first example values a 40-year insurance on a policyholder aged 25, with benefits payable at the end of the month of death. Equation 13 would determine the benefit premium using the commutation table approach.

$$U = \frac{M_{25} - M_{65}}{D_{65}} \frac{i}{i^{(12)}} \tag{13}$$

The following lines of code compute the benefit premium using `UComm`, the commutation technique, and `UCpt`, the current payment technique.

```
R> data(soa08Act)
R> UComm <- Axn(actuarialtable=soa08Act, x=25, n=65-25, k=12)
R> UCpt <- ((soa08ActDf$Mx[26]-soa08ActDf$Mx[66])/soa08ActDf$Dx[26]) *
+                   0.06/real2Nominal(i=0.06,k=12)
R> c(UComm, UCpt)

[1] 0.0492762 0.0492762
```

If, while the policyholder is alive, the premium is paid in ten equal installments at the beginning of each year instead of a lump sum, then the yearly premium, P , would be determined as follows:

```
R> P <- UCpt/axn(actuarialtable=soa08Act,x=25,n=10)
R> P

[1] 0.006351046
```

The **lifecontingencies** package allows one to evaluate APVs of endowment insurances; this can be calculated for increasing and decreasing life insurances as well. The lines of code that follow will prove the actuarial equivalence expressed by Equation 14 in a computational context.

$$(n+1) * A^1_{x:\overline{n}|} = (DA)^1_{x:\overline{n}|} + (IA)^1_{x:\overline{n}|} \tag{14}$$

```
R> (10 + 1 ) * Axn(actuarialtable=soa08Act, x=25, n=10)

[1] 0.1194392

R> DAxn(actuarialtable = soa08Act, x=25, n=10) +
+  IAxn(actuarialtable = soa08Act, x=25, n=10)
```

```
[1] 0.1194392
```

Life annuity examples

Life contingent annuities form sequences of payments whose occurrence and duration depend the policyholder's future lifetime. The few examples that follow demonstrate how the **lifecontingencies** package can directly compute the APV for typical life contingencies using either bundled functions or classical commutation tables.

Equation 15 expresses the full premium of a ten-year deferred annuity-due for a policyholder aged 75 by means of commutation functions.

$$U = {}_{10|}\ddot{a}_{75} = \frac{N_{85}}{D_{75}} \tag{15}$$

```
R> UCpt <- axn(actuarialtable=soa08Act, x=75, m=10)
R> UComm <- with(soa08ActDf,Nx[86]/Dx[76])
R> c(UCpt,UComm)
```

```
[1] 1.146484 1.146484
```

If the premium were paid by means of five annual payments as long as the insured were alive, Equation 15 would be rewritten as Equation 16.

$${}_5P({}_{10|}\ddot{a}_{75}) = \frac{{}_{10|}\ddot{a}_{75}}{\ddot{a}_{75:\overline{5}|}} = \frac{\frac{N_{85}}{D_{75}}}{\frac{N_{75}-N_{80}}{D_{75}}} \tag{16}$$

```
R> P=axn(actuarialtable=soa08Act, x=75, m=10) /
+                   axn(actuarialtable=soa08Act, x=75, n=5)
R> P
```

```
[1] 0.2854727
```

```
R> PComm <- with(soa08ActDf,(Nx[86]/Dx[76]) /
+                                   ((Nx[76]-Nx[81])/Dx[76]))
R> PComm
```

```
[1] 0.2854727
```

If amounts of $\frac{1}{m}$ were paid at the beginning of each month, the APV of the annuty would be $U = {}_{10|}\ddot{a}^{(12)}_{75}$.

```
R> U <- axn(actuarialtable=soa08Act, x=75, m=10, k=12)
R> P <- axn(actuarialtable=soa08Act, x=75, m=10, k=12) /
+                   axn(actuarialtable=soa08Act, x=75, n=5)
R> c(U,P)
```

```
[1] 1.0325687 0.2571079
```

Benefit reserves examples

The (prospective) benefit reserve consists in the difference between the APV of obligated future benefit payments due by the insurer and the APV of the projected inflows due by the policyholder. It represents the outstanding net insurer's obligation arising from the underwritten insurance policy. An example will clarify this concept.
The code below evaluates the benefit reserve for a 25 year old 40 - year life insurance of $ 100,000, with benefits payable at the end of the year of death, assuming a level benefit premium payable at the beginning of each year. The benefit premium and reserve equations for this life contingent insurance are displayed by Equation 17.

$$\begin{aligned} P\ddot{a}_{25:\overline{40|}} &= 100000 A^{1}_{25:\overline{40|}} \\ {}_{t}V^{1}_{25+t:\overline{n-t|}} &= 100000 A^{1}_{25+t:\overline{40-t|}} - P\ddot{a}_{25+t:\overline{40-t|}} \end{aligned} \tag{17}$$

```
R> P=100000 * Axn(soa08Act,x=25,n=40)/axn(soa08Act,x=25,n=40)
R> reserveFun = function(t) return(100000*Axn(soa08Act,x=25+t,n=40-t)-P*
+                                          axn(soa08Act,x=25+t,n=40-t))
R> for(t in 0:40) {if(t%%5==0) cat("At time ",t,
+                                  " benefit reserve is ",
+                                  reserveFun(t),"\n")}

At time  0  benefit reserve is  0
At time  5  benefit reserve is  1109.887
At time  10  benefit reserve is  2401.368
At time  15  benefit reserve is  3825.877
At time  20  benefit reserve is  5256.254
At time  25  benefit reserve is  6421.799
At time  30  benefit reserve is  6789.186
At time  35  benefit reserve is  5328.03
At time  40  benefit reserve is  0
```

Another reserve calculation example shows the benefit reserve for a deferred annuity-due on a policyholder aged 25 when the annuity is deferred until age 65. The code below shows the reserve calculation while Figure 3 plots the outstanding reserve at the end of each contract year.

```
R> yearlyRate <- 12000
R> irate <- 0.02
R> APV <- yearlyRate*axn(soa08Act, x=25, i=irate,m=65-25,k=12)
R> levelPremium <- APV/axn(soa08Act, x=25,n=65-25,k=12)
R> annuityReserve<-function(t) {
+          out<-NULL
+          if(t<65-25) out <- yearlyRate*axn(soa08Act, x=25+t,
+      i=irate, m=65-(25+t),k=12)-levelPremium*axn(soa08Act,
```

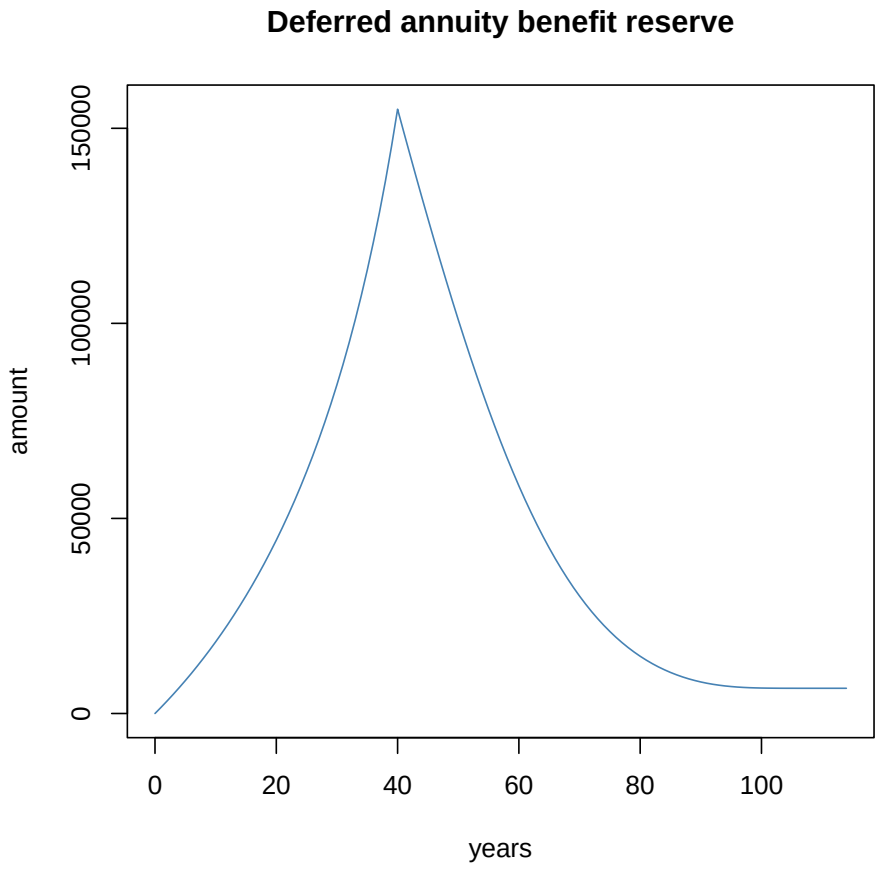


Figure 3: Benefit reserve profile for the exemplified annuity contract

```
+                 x=25+t, n=65-(25+t),k=12) else {
+                   out <- yearlyRate*axn(soa08Act, x=25+t, i=irate,k=12)
+           }
+           return(out)
+  }
R> years <- seq(from=0, to=getOmega(soa08Act)-25-1,by=1)
R> annuityRes <- numeric(length(years))
R> for(i in years) annuityRes[i+1] <- annuityReserve(i)
R> dataAnnuityRes <- data.frame(years=years, reserve=annuityRes)
```

Expense considerations

The premium paid by the policyholder usually contains an allowance for expenses and profit loading. Expenses cover policy servicing and the producers' commissions. The insurers' profit load is explicitly taken into account in the benefit premium as a flat amount, or, sometimes, as a percentage of the final premium. In other cases implicit profit loading is generated by using demographic and financial assumptions more prudentially than would be necessary. The equivalence principle can be extended to both the gross premium, G, and the expense augmented reserve, ${}_tV^E$, when expense allowances are taken into account by using Equation 18.

$$\begin{aligned} G &= \text{APV (Benefits)} + \text{APV (Expenses)} \\ {}_tV^E &= \text{APV (Benefits)} + \text{APV (Expenses)} - \text{APV (Gross Premium)} \end{aligned} \tag{18}$$

The following example shows how an expense loaded premium G is calculated for a \$ 100,000 whole life insurance on a 35 year old policyholder. 10% of premium expense per year, 25 policy expenses per year, and annual maintenance expense of 2.5 per 1,000 units of capital are assumed.

The equation to be solved is $G * \ddot{a}_{35} = 100000 * A_{35} + (2.5 * 100000/1000 + 25 + 0.1G) * \ddot{a}_{35}$.

```
R> G <- (100000*Axn(soa08Act, x=35) + (2.5*100000/1000 + 25)*
+                            axn(soa08Act,x=35))/((1-.1)*axn(soa08Act,x=35))
R> G

[1] 1234.712
```

Insurances and annuities on two lives

The package provides functions designed to evaluate life insurances and annuities on two lives. The following example checks the actuarial mathematics identity on joint and last survival status annuities expressed by Equation 19.

$$a_{\overline{xy}} = a_x + a_y - a_{xy} \tag{19}$$

```
R> twoLifeTables <- list(maleTable=soa08Act, femaleTable=soa08Act)
R> axn(soa08Act, x=65,m=1)+axn(soa08Act, x=70,m=1)-
+  axyzn(tablesList=twoLifeTables,        x=c(65,y=70),status="joint",m=1)

[1] 10.35704

R> axyzn(tablesList=twoLifeTables, x=c(65,y=70), status="last",m=1)

[1] 10.35704
```

Finally, reversionary annuities (annuities payable to life y upon death of x) APVs, $a_{x|y} = a_y - a_{xy}$, can also be computed by combining **lifecontingencies** functions as the code below shows.

```
R> axn(actuarialtable = soa08Act, x=60,m=1)-
+                   axyzn(tablesList = twoLifeTables,
+                                      x=c(65,60),status="joint",m=1)

[1] 2.695232
```

4.5. Stochastic analysis

This last section illustrates some stochastic analysis that can be performed by the **lifecontingencies** package, in both demographic (Section 4.5.1) and life insurance contexts (Section 4.5.2).

Demographic examples

The age-until-death, both in the continuous, $\tilde{T}_x$, or curtate form, $\tilde{K}_x$, is a stochastic variable whose distribution is intrinsic in the deaths within a life table. Therefore, a dedicated function, `rLife`, has been designed within the **lifecontingencies** package to draw a sample from either $\tilde{K}_x$ or $\tilde{T}_x$. Drawing from K_x is quite simple: the distribution of the curtate future lifetime is defined as $\Pr\left[\tilde{K}_x = t\right] = \frac{d_{x+t}}{l_x} = \frac{d_{x+t}}{\sum_{j=0}^{\omega-x} d_{x+j}}$, and it is passed as a `prob` parameter to base R `sample` function. For example, the code below shows how the `rLife` function can be used to draw a sample of size five from the curtate future lifetime of a policyholder aged 45 as implied by the SOA life table.

```
R> rLife(n = 5, object = soa08Act, x = 45, type = "Kx")

[1] 40 18 29 12 51
```

`rLifexyz` represents the multiple lives extension of the `rLife` function. It returns a matrix of sampled expected future lifetimes of J policyholders given a list of J lifetables. The simulation approach is useful in evaluating demographic quantities when the analytical approach is not feasible. One example could be the expected years of widowhood, which Equation 20 defines. $\tilde{T}_x$ and $\tilde{T}_y$ in Equation 20 stand for complete future lifetimes for the husband and the wife respectively.

$$E\left[\tilde{W}_y\right] = \max\left(0, \tilde{T}_y - \tilde{T}_x\right) \tag{20}$$

The following code shows how this function could be used to evaluate the expected years of widowhood for the wife of a couple. The example makes use of the Italian projected life tables ips55M and ips55F, whose derivation was shown in Section 4.2.

```
R> futureLifetimes <- as.data.frame(rLifexyz(n=numSim,
+                                    tablesList=list(husband=ips55M,wife=ips55F),
+                                    x=c(68,65), type="Tx"))
R> names(futureLifetimes) <- c("husband","wife")
```

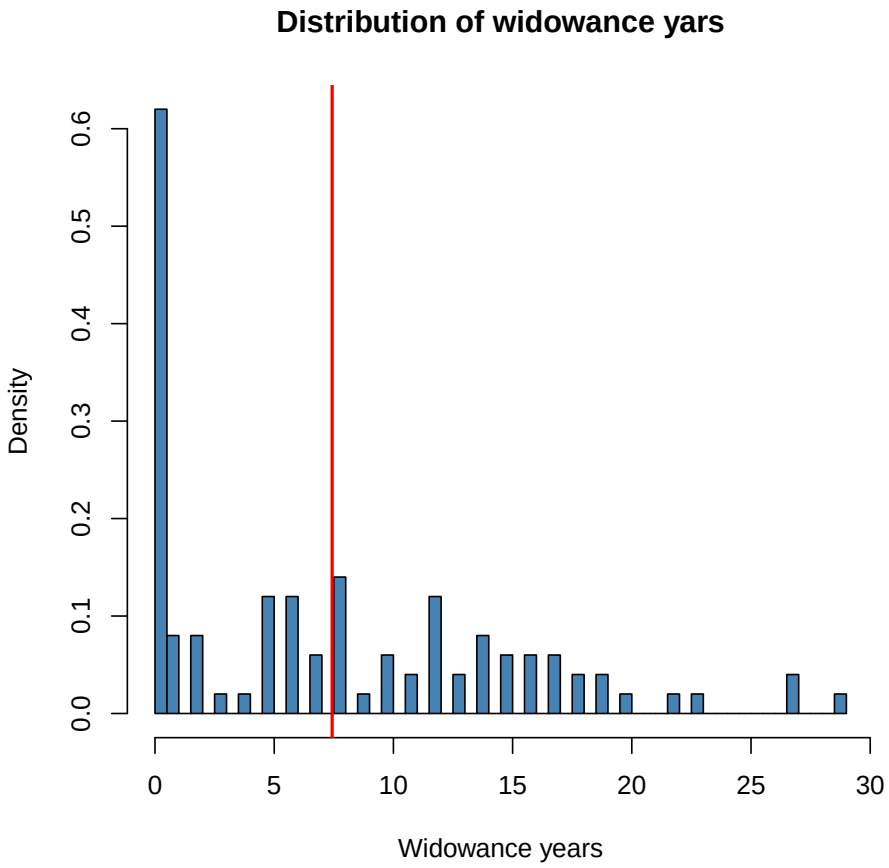


Figure 4: Years of widowance distribution, where the red line represents the expected value.

```
R> temp <- futureLifetimes$wife - futureLifetimes$husband
R> futureLifetimes$widowance  <-  sapply(temp, max,0)
R> mean(futureLifetimes$widowance)

[1] 7.43
```

Finally, Figure 4 shows the distribution of widowance years as determined in the previous example.

Actuarial mathematics examples

The present value of the future benefits cash flows distribution, $\tilde{Z}$, is a random variable. It is a function of both the interest rate and the indicator variables which are determined by the life status of the insured. Both of these quantities can be deemed stochastic. However, interest rates are considered deterministic within the framework of the current version of **lifecontingencies** package.
The generation of n-size variates from $\tilde{Z}$ is performed by the following algorithm:

1. Define a function, PV that returns the present value of the life contingent insurance benefits, given the age at death of the policyholder, as T_0, $PV(T_0)$. Within the **lifecontingencies** package, present value functions have been defined for the most important life contingencies. Such functions are not visibly exported in the package namespace.
2. Sample n variates from T_0.
3. Use T_0 variates as inputs for $PV(T_0)$ to get variates from $\tilde{Z}$.

The code below shows the internal function `.faxn`, which returns the present value of a life contingent insurance. `.faxn` is internally called by the `rLifeContingencies` function, as discussed below. `T`, `y`, `n`, `i`, `m`, `k` represent the age at death, the attained age, the term of the annuity, the interest rate, the deferring period, and the fractional payment frequency respectively.

```
.faxn<-function(T,y,n, i, m, k=1)
{
        out <- numeric(1)
        K <- T-y
                if(K<m) {
                        out <- 0
                } else {
                  times <- seq(from=m, to=min(m+n-1/k,K),by=1/k)
                   out <- presentValue(cashFlows=rep(1/k, length(times)),
          timeIds=times, interestRates=i)
                }
        return(out)
}
```

Life contingency insurance functions return the APV, $E\left[\tilde{Z}\right]$, as a default value. The functions in Table 5 compute APVs by using the current payment technique. Another possible approach for evalutating APVs, even if computationally inefficient, could be to draw a sample from the underlying $\tilde{Z}$ distribution and compute its sample mean.

Every function in Table 5 returns a sample of size one if the `type` parameter default value, "EV" (expected value), is overridden by the string "ST" (stochastic).

However, when samples of greater size are required, the most straightforward approach is the `rLifeContingencies` function. The code below shows how to generate $\tilde{Z}$ variates from either term life insurances, increasing term insurances, temporary annuities, or endowment insurances respectively. For each example, the lack of bias is verified by comparing the mean of the sample with the theoretical APV using a classical t - test. All examples refer to an individual aged 20 with a 40 year insurance. Figure 5 shows the resulting $\tilde{Z}$ distributions.

Note on performance. The present values of the simulated lives are computed by vectorized C++ kernels, so that large samples are generated much faster than with an element-wise loop. Multi-threaded computation through OpenMP is available but strictly opt-in: it requires both `parallel = TRUE` and `options(lifecontingencies.openmp = TRUE)`; the number of threads is set by the `nthreads` argument or by `options(lifecontingencies.nthreads)` (default 2). When the option is not set, or the package was built without OpenMP support, the sequential kernel is used. Results do not depend on the number of threads, as lifetimes are drawn in R before the parallel step. All examples below use the default, sequential, setting.

```
R> APVAxn <- Axn(soa08Act,x=25,n=40,type="EV")
R> APVAxn

[1] 0.04797088

R> sampleAxn <- rLifeContingencies(n=numSim, lifecontingency="Axn",
+                  object=soa08Act,x=25,t=40,parallel=FALSE)
R> tt1 <-t.test(x=sampleAxn,mu=APVAxn)$p.value
R> APVIAxn <- IAxn(soa08Act,x=25,n=40,type="EV")
R> APVIAxn

[1] 1.045507

R> sampleIAxn <- rLifeContingencies(n=numSim, lifecontingency="IAxn",
+                  object=soa08Act,x=25,t=40,parallel=FALSE)
R> tt2 <-t.test(x=sampleIAxn,mu=APVIAxn)$p.value
R> APVaxn <- axn(soa08Act,x=25,n=40,type="EV")
R> APVaxn

[1] 15.46631

R> sampleaxn <- rLifeContingencies(n=numSim, lifecontingency="axn",
+                  object=soa08Act,x=25,t=40,parallel=FALSE)
R> tt3 <- t.test(x=sampleaxn,mu=APVaxn)$p.value
R> APVAExn <- AExn(soa08Act,x=25,n=40,type="EV")
R> APVAExn

[1] 0.1245487
```

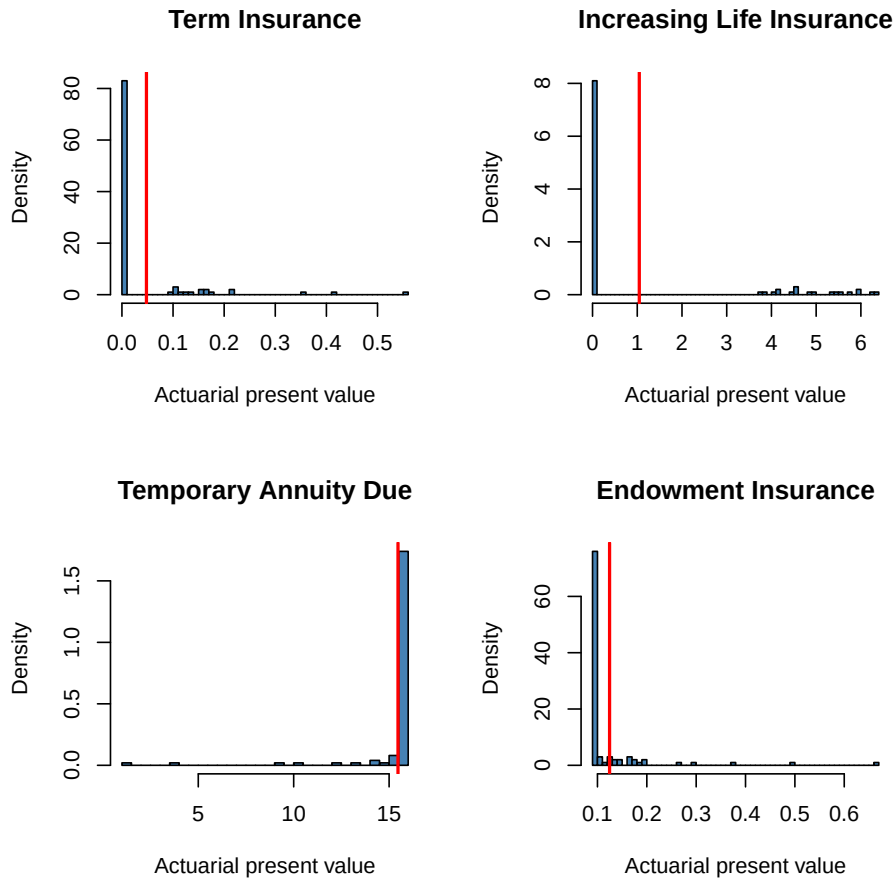


Figure 5: Life insurance stochastic variables distributions. Red vertical line represents APV.

```
R> sampleAExn <- rLifeContingencies(n=numSim, lifecontingency="AExn",
+                  object=soa08Act,x=25,t=40,parallel=FALSE)
R> tt4 <-t.test(x=sampleAExn,mu=APVAExn)$p.value
R> c(tt1, tt2,tt3, tt4)

[1] 0.1157566 0.6123668 0.8523421 0.7995087
```

The same analysis can be performed on life contingencies insurance on two (or more lives) as listings exemplify below.

```
R> tablesList=list(soa08Act,soa08Act);x=c(60,60);m=0;status="last";t=30;k=1
R> APVAxyz<-Axyzn(tablesList=tablesList,x=x,n=t,status=status,type="EV")
R> samplesAxyz<-rLifeContingenciesXyz(n=numSim,lifecontingency = "Axyz",
+                  tablesList = tablesList,x=x,t=t,m=m,k=k,status=status,
+                  parallel=FALSE)
R> tt5<-t.test(x=samplesAxyz, mu=APVAxyz)$p.value
```

```
R> APVaxyz<-axyzn(tablesList=tablesList,x=x,n=t,m=m,k=k,status=status,type="EV")
R> samplesaxyz<-rLifeContingenciesXyz(n=numSim,lifecontingency = "axyz",
+                   tablesList = tablesList,x=x,t=t,m=m,k=k,status=status,
+                   parallel=FALSE)
R> tt6<-t.test(x=samplesaxyz, mu=APVaxyz)$p.value
R> c(tt5,tt6)

[1] 0.3111299 0.4808113
```

All contingency functions have been provided an argument `power`, whose default value is set to one, that becomes useful when moments higher than one (the expected) of the life contingency random variable are needed. For example a direct computation of term insurance variance can be performed as follows.

```
R> var(sampleAxn)

[1] 0.008209687

R> Axn(soa08Act, x=25,n=40, power=2)-Axn(soa08Act, x=25,n=40, power=1)^2

[1] 0.01468876
```

The example that follows verifies that the variance an endowment insurance, calculated by the rule of moments, matches with the variance of the simulated distribution.

The full distribution of a life contingent insurance variable $\tilde{Z}$, can be used to compute premiums using the percentile premium principle. Under this approach, the premium is set to ensure that the insurer will suffer financial loss with a sufficiently low probability (made explicit by the percentile).

An example will clarify the concept. For a 40 - year insurance on a single policyholder aged 25, the actuarial present value of benefits, i.e., the expected value of discounted future benefits, would be

```
R> APV <- Axn(actuarialtable = soa08Act, x=25, n=40)
R> APV

[1] 0.04797088
```

while the benefit premium at the 90th percentile, that is, the premium that would make the insurer to suffer an underwriting loss with 10% probability, would be

```
R> samples <- rLifeContingencies(n=numSim, lifecontingency = "Axn",
+                   object= soa08Act, x=25,t=40,parallel=FALSE)
R> pct90Pr <- as.numeric(quantile(samples,.90))
R> pct90Pr

[1] 0.1234772
```

Finally, if $N = 1000$ similar policyholders were insured, the Law of Large Numbers would lead to a strong reduction in the premium charged on each policyholder, as computed below.

```
R> pct90Pr2 <- qnorm(p=0.90,mean=APV, sd=sd(samples)/sqrt(1000))
R> pct90Pr2

[1] 0.05307155
```

The final example in this paper shows how the stochastic functions bundled in the **lifecontingencies** package can be used to make an actuarial appraisal of embedded benefits.
Suppose a corporation grants its 100 employees life insurance benefits equal to their annual salary and payable at the month of death. Moreover, suppose that:

1. The expected value and the standard deviation of the salary are $ 50,000 and $ 15,000 respectively and the salaries follow a log-normal distribution.
2. The distribution of the employees' age is uniform on [25,65]. Assume 65 is the retirement age.
3. The SOA illustrative table represents an unbiased description of the population mortality.
4. Assume no lapse to hold.
5. The policy length is annual.

We evaluated the best estimate, or the fair value of the insured benefits according to both IFRS accounting standards and risk margin measure. In this example, the risk margin measure, which is the difference between the 75th percentile and the best estimate, will be used. IFRS standards (Post, Grandl, Schmidl, and Dorfman 2007) define the fair value of an insurance liability as the sum of its best estimate and its risk margin.

In the initial part of the example, we set up the parameter of the model and configured the parallel computation facility available with the package **parallel**. The code parallelization has been adapted from examples found in the (McCallum and Weston 2011) textbook.

```
R> nsim <- 50
R> employees <- 100
R> salaryDistribution <- rlnorm(n=employees,m=10.77668944,s=0.086177696)
R> ageDistribution <- round(runif(n=employees,min=25, max=65))
R> policyLength <- sapply(65-ageDistribution, min, 1)
R> getEmployeeBenefit<-function(index,type="EV") {
+          out <- numeric(1)
+          out <- salaryDistribution[index]*Axn(actuarialtable=soa08Act,
+                          x=ageDistribution[index],n=policyLength[index],
+                          i=0.02,m=0,k=1, type=type)
+          return(out)
+  }
```

```
R> require(parallel)
R> cl <- makeCluster(1)
R> worker.init <- function(packages) {
+          for (p in packages) {
+                  library(p, character.only=TRUE)
+          }
+          invisible(NULL)
+  }
R> clusterCall(cl,
+                  worker.init, c('lifecontingencies'))

[[1]]
NULL

R> clusterExport(cl, varlist=c("employees","getEmployeeBenefit",
+                                    "salaryDistribution","policyLength",
+                                    "ageDistribution","soa08Act"))
```

Then we perform best estimate and risk margin calculations.

```
R> employeeBenefits <- numeric(employees)
R> employeeBenefits <- parSapply(cl, 1:employees,getEmployeeBenefit, type="EV")
R> employeeBenefit <- sum(employeeBenefits)
R> benefitDistribution<-numeric(nsim)
R> yearlyBenefitSimulate<-function(i)
+  {
+          out <- numeric(1)
+          expenseSimulation <- numeric(employees)
+          expenseSimulation <- sapply(1:employees, getEmployeeBenefit, type="ST")
+          out <- sum(expenseSimulation)
+          return(out)
+  }
R> benefitDistribution <- parSapply(cl, 1:nsim,yearlyBenefitSimulate )
R> stopCluster(cl)
R> riskMargin <- as.numeric(quantile(benefitDistribution,.75)-employeeBenefit)
R> totalBookedCost <- employeeBenefit+riskMargin
R> employeeBenefit

[1] 22754.53

R> riskMargin

[1] 24325.88

R> totalBookedCost

[1] 47080.41
```

4.6. Multiple Decrement Models within lifecontingencies Package

Until now no R package provides a good tool to manage multiple decrement tables, even if Deshmukh (2012) provides an R based focus on multiple decrement tables with applications in R. The topic is deeply related to multistate analysis of life histories on which Willekens (2014) provide a very good introduction. The **lifecontingencies** package's `mdt` class has been specifically engineered to manage multiple decrements models with R.

The mdt class

We create a `mdt` class object using the first example found on (Valdez 2011, p. 4).

```
R> valdezDf<-data.frame(
+                   x=c(50:54),
+                   lx=c(4832555,4821937,4810206,4797185,4782737),
+                   heart=c(5168, 5363, 5618, 5929, 6277),
+                   accidents=c(1157, 1206, 1443, 1679,2152),
+                   other=c(4293,5162,5960,6840,7631)
+  )
R> valdezMdt<-new("mdt",name="ValdezExample",table=valdezDf)

Added fictional decrement below last x and completed x and lx until zero....
Completed the table at top, all decrements on first cause
```

The `mdt` class is an S4 class object (Chambers 2008) comprised by a character slot `name` and a `data.frame` slot `table`, made of an `x` (age) column, an `lx` (subjects at risk) column, and one or more columns for different causes of decrement, holding the absolute number of subjects dying for each cause between age x and $x + 1$. When the supplied table only spans part of the age range, `setValidity` and the internal `.tableSanitizer` function complete it below and/or above (reporting what was done on the console); the survival ratio used to synthetically reconstruct ages below the lowest one supplied can be set via the `bottomCompletionSurvival` argument of `new("mdt", ...)` (default `0.99`).

Decrement counts and probabilities can then be computed with `dxt` and `qxt`:

```
R> dxt(valdezMdt,x=51,decrement="other")
R> qxt(valdezMdt,x=53,t=2,decrement=1)

[1] 0.002544409
```

The dedicated vignette (`vignette("multiple_decrement_tables", package = "lifecontingencies")`) develops this topic in full: decrement probability calculations, print/show/summary methods and coercion to `data.frame`/`markovchainList`, Associated Single Decrement Tables (ASDT) under the uniform distribution of decrements (UDD) assumption, actuarial applications (`Axn.mdt` and multiple-decrement annuities), and a complete worked example that builds a real, cause-specific `mdt` object from an official U.S. mortality report.

5. Software engineering and package architecture

Although **lifecontingencies** is primarily presented as a collection of actuarial and demographic functions, its implementation follows the standard architecture of an R package and combines interpreted R code with compiled code where computational efficiency is important. This separation makes it possible to maintain a high-level, user-facing R interface while implementing selected numerical operations in compiled C++ code.

The main components of the source tree can be summarized as follows:

```
lifecontingencies/
  R/                    R functions and classes
  src/                  C++ implementations
  tests/
    testthat.R          entry point for package tests
    testthat/           automated test suite
  inst/
    benchmarks/         performance-oriented development benchmarks
  vignettes/             package documentation and examples
```

The R directory contains the public-facing functions and the implementation of the package's actuarial and demographic classes. The C++ code in `src/` is used for selected computationally intensive operations. The interface between R and C++ is provided by the **Rcpp** package (Eddelbuettel and Francois 2011). This approach allows performance-critical numerical computations to be compiled while retaining the R API used by package users.

The compiled backend is treated as an implementation detail rather than as a separate user-facing API. Consequently, changes to the computational backend can be validated against the established R implementations and against the package's actuarial identities and numerical properties. When an independent historical R implementation is available, regression tests can use it as a reference implementation. This is particularly useful when replacing a frequently called R computation with a compiled implementation: the objective is to improve execution time without changing the mathematical result.

Automated testing is organized using the **testthat** framework (Wickham 2011a). The files under `tests/testthat/` contain functional, regression, validation, and defensive-programming tests. The test suite is executed as part of the package checking and continuous integration process. This provides an additional safeguard against unintentional changes in numerical results or in the public behaviour of the package.

Performance measurements are kept separate from the functional test suite. Benchmark scripts are useful during development for comparing alternative implementations, in particular interpreted R and compiled C++ implementations, but execution time is inherently dependent on the hardware and software environment. Such measurements are therefore not used as pass/fail criteria for the standard package tests.

This architecture separates four complementary concerns: the actuarial interface and mathematical implementation in R, computationally intensive numerical kernels in C++, automated correctness checks through **testthat**, and performance evaluation through dedicated benchmarks. Together with continuous integration, this structure supports the long-term maintenance of the package while allowing computational optimizations without changing its user-facing interface.

6. Discussion

6.1. Advantages, limitations, and future perspectives

The **lifecontingencies** package allows actuaries to perform demographic, financial and actuarial mathematics calculations within R; in particular, life contingent insurance contracts can be priced and reserved. In addition, a peculiar feature of **lifecontingencies** is its ability to generate variates from the future life time and the underlying stochastic distributions of life contingent insurances.

One of the most significant limitations of the most recent package release is that few functions are available to model multiple decrements tables. In addition, continuous-time life contingent models are currently not handled explicitly.

We expect to remove such limitations in the future. In addition, we expect to provide coerce methods for packages that specialize in demographic analysis, like **demography** and **LifeTables**. Furthermore, we wish to allow easier sharing of analyses with interest rate modeling packages like **termstrc**.

Finally, code optimization and improvement is carried out continuously. Key parts of core functions use C++ code snippets via the **Rcpp** package Eddelbuettel (2013b).

6.2. Accuracy

The accuracy of the calculations has been verified by checking numerical examples reported in Bowers *et al.* (1997) and in the lecture notes of "Actuarial Mathematics" taken by the author years ago at The Catholic University of Milan (Mazzoleni 2000). Such test have been implemented with unit root testing in the package **testthat**, Wickham (2011b). The numerical results are identical to those reported in the cited references for most functions, with the exception of fractional payment annuities; for these, the answer is only reported up to the fifth decimal. The reason for such inaccuracy is that the package calculates the APV using the direct sum of fractional survival probabilities, while the results reported in the cited references are obtained by closed formulas.

Finally, it is worth noting that the package and functions herein are provided as is without any guarantee regarding the accuracy of calculations. The author disclaims any liability arising from potential losses due to direct or indirect use of this package. Users are invited to notice the author any potential bug from the github package site `https://github.com/spedygiorgio/lifecontingencies`.

Acknowledgments

The author wishes to thank all those whose suggestions contributed to the package's enhancements. In particular, he would like to thank Christophe Dutang, Tim Riffle, Reinhold Kainhofer and Kevin J. Owens for their suggestions on code and vignettes. A special thank also to Anton Sylchenko for the copy-editing. Also, many thanks go out to the anonymous Journal of Statistical Software referees that helped improve the quality of the package and underlying vignettes.

References

Anderson J (1999). *Commutation Functions*. Education and Examination Committee of the Society of Actuaries.

Bowers NL, Jones DA, Gerber HU, Nesbitt CJ, Hickman JC (1997). *Actuarial Mathematics, 2nd Edition*. Society of Actuaries. ISBN 9780938959106.

Broverman S (2008). *Mathematics of Investment and Credit*. ACTEX Academic Series. ACTEX Publications. ISBN 9781566986571.

Chambers J (2008). *Software for Data Analysis: Programming with R*. Statistics and computing. Springer-Verlag. ISBN 9780387759357.

Chris Ruckman, Joe Francis (2006). *Financial Mathematics: A Practical Guide for Actuaries and Other Business Professionals*. Warren Centre for Actuarial Studies and Research, 2nd edition edition.

Christophe Dutang, Vincent Goulet, Mathieu Pigeon (2008). "**actuar**: An R Package for Actuarial Science." *Journal of Statistical Software*, **25**(7), 38. URL https://www.jstatsoft.org/v25/i07/.

De Angelis, Paolo, Di Falco, L (2016). *Assicurazioni sulla salute: caratteristiche, modelli attuariali e basi tecniche*. Il Mulino. Il Mulino. ISBN 9788815260840.

Deshmukh S (2012). *Multiple Decrement Models in Insurance: An Introduction Using R*. SpringerLink : Bücher. Springer. ISBN 9788132206590.

Dickson D, Hardy M, Waters H (2009). *Actuarial Mathematics for Life Contingent Risks*. International Series on Actuarial Science. Cambridge University Press. ISBN 9780521118255.

Eddelbuettel D (2013a). "CRAN Task View: Empirical Finance." Version 2013-09-27, URL https://cran.r-project.org/view=Finance.

Eddelbuettel D (2013b). *Seamless R and C++ Integration with Rcpp*. Springer, New York. ISBN 978-1-4614-6867-7.

Eddelbuettel D, Francois R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Ferstl R, Hayden J (2010). "Zero-Coupon Yield Curve Estimation with the Package **termstrc**." *Journal of Statistical Software*, **36**(1), 1–34. URL https://www.jstatsoft.org/v36/i01/.

Finan MA (2014). "A Reading of the Theory of Life Contingency Models: A Preparation for Exam MLC." Lecture notes. Accessed: 01/01/2015.

Gesmann M, Zhang Y (2011). ***ChainLadder**: Mack, Bootstrap, Munich and Multivariate-chain-ladder Methods*. R package version 0.1.4-3.4.

Guirreri S (2010). *Simulating the Term Structure of Interest Rates with Arbitrary Marginals*. Ph.D. thesis, University of Palermo - Department of Statistics and Mathematics "S. Vianelli", Palermo. URL no longer available.

IBM Corp (2012). *SPSS, Release 21.0 Advanced Statistical Procedures Companion*. IBM Corp., Armonk, NY.

Keyfitz N, Caswell H (2005). *Applied Mathematical Demography*. Statistics for biology and health. Springer-Verlag. ISBN 9780387225371.

Klugman S, Panjer H, Willmot G, Venter G (2009). *Loss Models: From Data to Decisions*. 3rd edition. Wiley New York.

Lee R, Carter L (1992). "Modeling and Forecasting U.S. Mortality." *Journal of the American Statistical Association*, **87**(419), 659–675. doi:10.2307/2290201.

Mazzoleni P (2000). *Appunti di Matematica Attuariale*. EDUCatt Università Cattolica. ISBN 9788883110825.

McCallum Q, Weston S (2011). *Parallel R*. O'Reilly Media. ISBN 9781449309923.

Post T, Grandl H, Schmidl L, Dorfman MS (2007). "Implications of IFRS for the European Insurance Industry Insights from Capital Market Theory." *Risk Management and Insurance Review*, **10**(2), 247–265. ISSN 1540-6296. doi:10.1111/j.1540-6296.2007.00117.x. URL https://doi.org/10.1111/j.1540-6296.2007.00117.x.

Riffe T (2011). *LifeTable: **LifeTable**, a Package with a Small Set of Useful Lifetable Functions*. R package version 1.0.1.

Rigby RA, Stasinopoulos DM (2005). "Generalized Additive Models for Location, Scale and Shape." *Applied Statistics*, **54**, 507–554.

Rob J Hyndman, Heather Booth, Leonie Tickle, John Maindonald (2011). *demography: Forecasting Mortality, Fertility, Migration and Population Data*. R package version 1.09-1, URL https://cran.r-project.org/package=demography.

SAS Institute Inc (2011). *The SAS System, Version 9.3*. SAS Institute Inc., Cary, NC. URL https://www.sas.com/.

Spedicato GA (2013). "The lifecontingencies Package: Performing Financial and Actuarial Mathematics Calculations in R." *Journal of Statistical Software*, **55**(10), 1–36. URL https://www.jstatsoft.org/v55/i10/.

Spedicato, Giorgio Alfredo (2015). *markovchain: discrete time Markov chains made easy*. R package version 0.3.

Stasinopoulos DM, Rigby RA (2007). "Generalized Additive Models for Location Scale and Shape (GAMLSS) in R." *Journal of Statistical Software*, **23**(7), 1–46. URL https://www.jstatsoft.org/v23/i07/.

SunGard (2012). *iWorks **Prophet***. SunGard. URL https://www.prophet-web.com/.

Team RDC (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL https://www.r-project.org/.

The MathWorks, Inc (2011). *MATLAB – The Language of Technical Computing, Version R2011b*. The MathWorks, Inc., Natick, Massachusetts. URL https://www.mathworks.com/products/matlab/.

Towers Watson (2011). ***MoSes**: Risk and Financial Modelling Software for Life Insurers*. Towers Watson.

Valdez E (2011). "Multiple Decrement Models: Math 3631 Actuarial Mathematics II." Lecture notes. Original URL no longer available; accessed: 25/04/2014.

Wickham H (2011a). "testthat: Get Started with Testing." *The R Journal*, **3**(1), 5–10. doi:10.32614/RJ-2011-002.

Wickham H (2011b). "testthat: Get Started with Testing." *The R Journal*, **3**, 5–10. doi:10.32614/RJ-2011-002.

Wickham H (2016). *ggplot2: Elegant Graphics for Data Analysis*. 2 edition. Springer-Verlag New York. ISBN 978-3-319-24277-4. doi:10.1007/978-3-319-24277-4. URL https://ggplot2.tidyverse.org/.

Willekens F (2014). *Multistate Analysis of Life Histories with R*. Use R! Springer International Publishing. ISBN 9783319083834.

Zhang W (2011). ***cplm**: Monte Carlo EM Algorithms and Bayesian Methods for Fitting Tweedie Compound Poisson Linear Models*. R package version 0.2-1, URL https://cran.r-project.org/package=cplm.

Affiliation:

Giorgio Alfredo Spedicato
Ph.D FCAS FSA CSPA C.Stat
URL: https://github.com/spedygiorgio/lifecontingencies