

# Package **pyliferisk**

September 17, 2023

**Type:** Python Package  
**Version:** 1.12.0  
**Author:** Francisco Gárate Santiago  
**Contact:** fgarate@ucm.es  
**License:** GPLv3  
**Repository:** <https://github.com/franciscogarate/pyliferisk>

**Abstract:** A python library for life actuarial calculations. Simple, powerful and easy-to-use.

This document aims to be the only necessary and authoritative source of information about pyliferisk, usable as a comprehensive reference, an user guide, and tutorial all-in-one. It also includes a sample of illustrative and common examples of actuarial calculations.

Pyliferisk is compatible with both version Python 3.x and 2.7 and has no dependencies other than the Python Standard Library, making it amazingly fast.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. The author does not take any legal responsibility for the accuracy, completeness, or usefulness of the information herein.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Pyliferisk: List of formulas</b>	<b>5</b>
2.1	Biometric functions: <i>Class</i> MortalityTable . . . . .	5
2.1.1	<i>method</i> .qx[x] . . . . .	5
2.1.2	<i>method</i> .lx[x] . . . . .	5
2.1.3	<i>method</i> .w . . . . .	6
2.1.4	<i>method</i> mt.dx[x] . . . . .	6
2.1.5	<i>method</i> .ex[x] . . . . .	6
2.2	Actuarial Present Value: <i>Class</i> Actuarial . . . . .	7
2.2.1	<i>formula</i> Ax() . . . . .	7
2.2.2	<i>formula</i> Axn() . . . . .	7
2.2.3	<i>formula</i> qAx() . . . . .	8
2.2.4	<i>formula</i> qAxn() . . . . .	8
2.2.5	<i>formula</i> AExn() . . . . .	8
2.3	Annuity formula . . . . .	11
2.3.1	<i>function</i> annuity() . . . . .	11
2.4	Pure endowment: Deferred capital . . . . .	14
2.4.1	<i>formula</i> nEx() . . . . .	14
2.5	Commutation factors . . . . .	15
2.6	Other functions . . . . .	15
2.7	Help (Documentation strings) . . . . .	16
<b>3</b>	<b>Mortality tables</b>	<b>17</b>
3.1	Adding tables . . . . .	17
3.1.1	Probability of dying tables (qx) . . . . .	17
3.1.2	Number of surviving tables (lx) . . . . .	17
3.2	<i>Class</i> .view() . . . . .	18
<b>4</b>	<b>Examples</b>	<b>19</b>
4.1	Example 1. Cashflow calculation: Annuity-immediate Geometrically increasing . . . .	19
4.2	Example 2. Drawing graph with matplotlib . . . . .	20
4.3	Example 3. Obtain data from plain text file . . . . .	21
4.4	Example 4. Obtain the Risk-free rate from MS Excel . . . . .	22
4.5	Example 5. Cashflow Calculation Reserving . . . . .	23
4.6	Example 6. Import a mortality table (lx) using pandas . . . . .	25
4.7	Example 7. Cashflows in pandas . . . . .	27
<b>5</b>	<b>Books</b>	<b>29</b>
<b>6</b>	<b>Acknowledgements</b>	<b>29</b>

# 1 Introduction

PyLiferisk is an open library written in Python for life and actuarial calculation contracts, based on commonly used methodologies among actuaries (International Actuarial Notation).

This library is able to cover all life contingencies risks (since the actuarial formulas follow the International Actuarial Notation), as well as to support the main insurance products: Term Life, Whole Life, Annuities and Universal Life. Additionally, the library can be easily tailored to any particular or local specifications, since Python is a very intuitive language.

It is ideal not only for academic purposes but also for professional use by actuaries (implementation of premiums and reserves modules) or by auditors (validation of reserves or capital risk models such as parallel runs).

This library is distributed as a single file module and has no dependencies other than the Python Standard Library, making it amazingly fast. Additionally, the package includes several life mortality tables (`pyliferisk.mortalitytables`), mainly extracted from academic textbooks. Nevertheless, additional libraries as Numpy or Pandas may be required for increasing functionality, such as cash flow operations, random number generation, interpolation, etc.

You can find also examples for different life contracts in the examples section.

## Why Python?

Because computing plays an important role in the actuarial profession, but actuaries are not programmers. Python is friendly and easy to learn.

Nowadays, programming is becoming an indispensable skill for actuaries. Python is a clear, readable syntax, powerful and fast language. Easy to learn, especially when you are not used to coding. This language lets you write quickly the code you need, without cumbersome rules or variable predefined tasks. It is clear, forget ending with commas and using curly brackets in functions.

For European actuaries, Solvency II opens a big opportunity. The new requirements transform into agility, transversality, and auditability. The internal model is not only software, but it should also be an internal process used extensively where all parts must walk hand in hand.

## Python 3 and 2.7

PyLiferisk is compatible with both version: 3.x and 2.7

## Potential uses

This library may be used in tariff processes, in the design phase of new products such as profit testing or estimation of future benefits. Other uses include:

- Auditing purposes tool
- Assumption calibrations, back-testing, etc..
- Replicate the main calculations of the internal model for implementation in pricing, product approval, reserving, etc..
- Perform small reports (output format may be xml, xls, etc...)

If you find something that Python cannot do, or if you need the performance advantage of low-level code, you can write or reuse extension modules in C or C++.

## Installation

Once Python is running, just install this library with `pip install`

```
> pip install pyliferisk
```

Then, to import this library in projects is automatic as usually:

Option 1:

```
1 from pyliferisk import *
2 from pyliferisk.mortalitytables import *
3
4 tariff = MortalityTable(nt=GKM95)
```

Option 2:

```
1 import pyliferisk as life
2 import pyliferisk.mortalitytables as mort
3
4 tariff = life.MortalityTable(nt=mort.GKM95)
```

Option 3, if only like to use specific functions:

```
1 from pyliferisk import MortalityTable
2 from pyliferisk.mortalitytables import GKM95
3
4 tariff = MortalityTable(nt=GKM95)
```

## Update library

Run this command from terminal:

```
> pip install pyliferisk --upgrade
```

## License

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version.

## 2 PyLiferisk: List of formulas

The names of the formulas follow the International Actuarial Notation and are easily guessable ( $q_x$ ,  $l_x$ ...) with a few exceptions regarding special characters.

### 2.1 Biometric functions: Class MortalityTable

The Instance Variables for the `MortalityTable()` class are:

- `nt` = The actuarial table used to perform life contingencies calculations. Syntax: `nt = GKM95` (Note: GKM95 must be created previously or be included in `mortalitytables.py`)

Example:

```
1 tariff = MortalityTable(nt=GKM95)
```

- `perc` = Optional variable to indicate the percentage of mortality to be applied. Syntax: `perc=85`. Variable `perc` can be omitted, in this case it will be 100 by default.

Example:

```
1 experience = MortalityTable(nt=GKM95, perc=85)
```

Once define the variables for your mortality table, all available biometric functions are the following:

#### 2.1.1 method `.qx[x]`

<b>Description</b>	Returns the probability that a life aged $x$ dies before 1 year With the convention: the true probability is $q_x/1000$
<b>Actuarial notation</b>	$q_x$
<b>Usage</b>	<code>mt.qx[x]</code>
<b>Args</b>	<code>x</code> : the age as integer number.
<b>Example</b>	<code>tariff = MortalityTable(nt=SPAININE2004)</code> <code>tariff.qx[50]</code>

#### 2.1.2 method `.lx[x]`

<b>Description</b>	Returns the number of survivors at beginning of age $x$
<b>Actuarial notation</b>	$l_x$
<b>Usage</b>	<code>mt.lx[x]</code>
<b>Args</b>	<code>x</code> : the age as integer number.
<b>Example</b>	<code>tariff.lx[50]</code>

### 2.1.3 method `.w`

<b>Description</b>	ultimate age ( $lw = 0$ )
<b>Actuarial notation</b>	$w$
<b>Usage</b>	<code>mt.w</code>
<b>Example</b>	<code>tariff.w</code>

### 2.1.4 method `mt.dx[x]`

<b>Description</b>	Returns the number of dying at beginning of age $x$
<b>Actuarial notation</b>	$d_x$
<b>Usage</b>	<code>mt.dx[x]</code>
<b>Args</b>	$x$ : the age as integer number.
<b>Example</b>	<code>tariff.dx[50]</code>

### 2.1.5 method `.ex[x]`

<b>Description</b>	Returns the curtate expectation of life. Life expectancy
<b>Actuarial notation</b>	$e_x$
<b>Usage</b>	<code>mt.ex[x]</code>
<b>Args</b>	self: the mortality table $x$ : the age as integer number.
<b>Example</b>	<code>tariff.ex[50]</code>

### Example:

Print the omega (limiting age) of the both mortality tables and the  $qx$  at 50 years-old:

```
1 from pyliferisk import MortalityTable
2 from pyliferisk.mortalitytables import INE2004, GKM95
3
4 tariff = MortalityTable(nt=INE2004)
5 experience = MortalityTable(nt=GKM95, perc=85)
6
7 # Print the omega (limiting age) of the both tables:
8 print(tariff.w)
9 print(experience.w)
10
11 # Print the qx at 50 years old:
12 print(tariff.qx[50] / 1000)
13 print(experience.qx[50] / 1000)
```

Return the following results:

```
101
121
0.003113
0.003662395
```

## 2.2 Actuarial Present Value: Class Actuarial

The Present Value of the benefit payment is a function of time of death given a survival model and an interest rate.

The Instance Variables for the `Actuarial()` class are:

- `nt` = The actuarial table used to perform life contingencies calculations.  
Syntax: `nt=GKM95` (Note: GKM95 must be included in `mortalitytables.py`)
- `i` = interest rate. The effective rate of interest, namely, the total interest earned in a year.  
Syntax: `i=0.02`
- `perc` = Optional variable to indicate the percentage of mortality to be applied.  
Syntax: `perc=85` . Variable `perc` can be omitted, in this case, it will be 100 by default.

```
1 | tariff = Actuarial(nt=GKM95, i=0.05)
```

### 2.2.1 formula $A_x()$

<b>Description</b>	Returns the Expected Present Value (EPV) of a whole life insurance (i.e. net single premium). It is also commonly referred to as the Actuarial Value or Actuarial Present Value.
<b>Actuarial notation</b>	$A_x$
<b>Usage</b>	<code>Ax(mt, x)</code>
<b>Args</b>	mt: Mortality table. x: the age as integer number.
<b>Example</b>	<code>mt = Actuarial(nt=SPAININE2004, i=0.02)</code> <code>Ax(mt, 50)</code>

### 2.2.2 formula $A_{x:n}()$

<b>Description</b>	Returns the EPV (net single premium) of a term insurance.
<b>Actuarial notation</b>	$A_{x:\overline{n} }^1$
<b>Usage</b>	<code>Axn(mt, x, n)</code>
<b>Args</b>	mt: Mortality table. x: the age as integer number. n: period in years.
<b>Example</b>	<code>mt = Actuarial(nt=SPAININE2004, i=0.02)</code> <code>Axn(mt, 50, 10)</code>

### 2.2.3 formula ${}_qA_x()$

<b>Description</b>	This function evaluates the APV of a geometrically increasing annual annuity-due.
<b>Actuarial notation</b>	${}_qA_x$
<b>Usage</b>	<code>Axn(mt, x, q)</code>
<b>Args</b>	mt: Mortality table. x: the age as integer number. q: increase rate.
<b>Example</b>	<code>mt = Actuarial(nt=SPAININE2004, i=0.02)</code> <code>Axn(mt, 50, 0.03)</code>

### 2.2.4 formula ${}_qA_{x:\overline{n}|}()$

<b>Description</b>	This function evaluates the APV of a geometrically increasing Term insurance.
<b>Actuarial notation</b>	${}_qA_{x:\overline{n} }$
<b>Usage</b>	<code>qAxn(nt, x, n, q)</code>
<b>Args</b>	mt: Mortality table. x: the age as integer number. n: period in years. q: increase rate.
<b>Example</b>	<code>mt = Actuarial(nt=SPAININE2004, i=0.02)</code> <code>Axn(mt, 50, 10, 0.03)</code>

### 2.2.5 formula $A_{Exn}()$

<b>Description</b>	Returns the EPV of an endowment insurance. An endowment insurance provides a combination of a term insurance and a pure endowment
<b>Actuarial notation</b>	$A_{x:\overline{n} }$
<b>Usage</b>	<code>AExn(mt, x, n)</code>
<b>Args</b>	mt: Mortality table. x: the age as integer number. n: period in years.
<b>Example</b>	<code>mt = Actuarial(nt=SPAININE2004, i=0.02)</code> <code>AExn(mt, 50, 10)</code>



### Syntax:

Notation	Description	Syntax
$A_x$	whole-life death insurance	$Ax(nt, x)$
$A_{x:\overline{n} }^1$	Term insurance	$Axn(nt, x, n)$
$A_{x:\overline{n} }$	Endowment insurance	$AExn(nt, x, n)$
${}_qA_x$	Increasing whole-life	$qAx(nt, x, n)$
${}_qA_{x:\overline{n} }$	Increasing Term insurance	$qAxn(nt, x, n, q)$

### Examples:

1) A whole-life single premium:

```
1 from pyliferisk import Actuarial, Ax
2 from pyliferisk.mortalitytables import GKM95
3
4 mt = Actuarial(nt=GKM95, i=0.02)
5 x = 50          #age
6 C = 1000        #capital
7
8 print(Ax(mt, x) * C)
```

589.0804423991423

2) A term insurance single premium:

```
1 from pyliferisk import Actuarial, Axn
2 from pyliferisk.mortalitytables import GKM95
3
4 mt = Actuarial(nt=GKM95, i=0.03)
5 x = 40          #age
6 n = 20          #horizon
7 C = 10000       #capital
8
9 print(Axn(mt, x, n) * C)
```

646.1486398262324

3) Example 2 with cashflow approach:

```
1 from pyliferisk import MortalityTable
2 from pyliferisk.mortalitytables import GKM95
3 import numpy as np
4
5 mt = MortalityTable(nt=GKM95)
6 x = 40          #age
7 n = 20          #horizon
8 C = 10000       #capital
9 i = 0.03        #interest rate
10
11 payments = []
12 for t in range(0, n):
13     payments.append((mt.lx[x+t] - mt.lx[x+t+1]) / mt.lx[x] * C)
14
15 discount_factor = []
16 for y in range(0, n):
17     discount_factor.append(1 / (1 + i) ** (y + 0.5))
18
19 print(np.dot(discount_factor, payments).round(2))
```

646.1486398262326

To print the results year by year:

```
print('{0:5} {1:10} {2:10}'.format(' t','factor','payment'))

for t in range(0,n):
    print('{0:2} {1:10} {2:10}'.format(t, np.around(
        discount_factor[t],5), np.around(payments[t],4)))
```

t	factor	payment
0	0.98533	18.694
1	0.95663	19.9456
2	0.92877	21.3621
3	0.90172	22.9574
4	0.87545	24.7629
5	0.84995	26.815
6	0.8252	29.1475
7	0.80116	31.7959
8	0.77783	34.7884
9	0.75517	38.1576
10	0.73318	41.9304
11	0.71182	46.1285
12	0.69109	50.7779
13	0.67096	55.8959
14	0.65142	61.4878
15	0.63245	67.5536
16	0.61402	74.0921
17	0.59614	81.095
18	0.57878	88.5512
19	0.56192	96.4435

## 2.3 Annuity formula

A life annuity refers to a series of payments to an individual as long as the individual is alive on the payment date. It may be temporary or payable for whole-life. The payment intervals may commence immediately or deferred. The payment may be due at the beginnings of the intervals (annuity due) or at the end (annuity immediate).

### 2.3.1 function annuity()

<b>Description</b>	Returns the actuarial present value of annuity payments	
<b>Usage</b>	<code>annuity(mt, x, n, 0/1, m=1, ['a'/'g',q], -d)</code>	
<b>Args</b>		
1.	<code>mt</code>	the mortality
2.	<code>x</code>	The age of the insured
3.	<code>n</code>	The horizon (term of insurance) in years or payment duration or
	or <code>'w'</code>	w as whole-life. Also 99 is defined as whole-life.
4.	<code>0</code>	annuity-immediate
	or <code>1</code>	annuity-due
<i>optional args:</i>		
—	<code>m:</code>	Number of fractional payments per period. If missing, m is set as 1
—	<code>['a', q]</code>	Arithmetically increasing
	or <code>['g', q]</code>	Geometrically increasing
—	<code>-d</code>	The deferring period in years (as negative integer)

### Examples:

1) The present value of a 5-year (financial) annuity with nominal annual interest rate 12% and monthly payments of \$100 is:

```
1 from pyliferisk import *
2 import numpy as np
3
4 mt = Actuarial(nt=FIN, i=0.12/12) #FIN = Financial table
5
6 n = 5 * 12
7 C = 100
8
9 print(annuity(mt, 0, n, 1) * C) #replace age 'x' by 0
```

Returns:

```
4495.503840622397
```

The equivalent formula in Excel is: `PV(12%/12,12*20,500,,0)`

2) Premium calculation: A Life Temporal insurance for a male, 30 years-old and a horizon of 10 years, fixed annual premium (GKM80, interest 6%):

$$\text{Actuarial equivalence: } \pi_{30:\overline{10}|}^1 = 1000 \cdot \frac{A_{30:\overline{10}|}^1}{\ddot{a}_{30:\overline{10}|}}$$

```

1 from pyliferisk import *
2 from pyliferisk.mortalitytables import GKM80
3
4 nt = Actuarial(nt=GKM80, i=0.06)
5 x = 30
6 n = 10
7 C = 1000
8
9 print(C * Axn(nt, x, n) / annuity(nt, x, n, 0))

```

```
1.396420811022648
```

3) Reserving a life risk insurance with regular premium. Applying the equivalence principle, where:

$${}_0V_x = A_x - \pi \cdot \ddot{a}_x = 0 \text{ and: } \pi = \frac{A_{x:\overline{n}|}^1}{\ddot{a}_{x:\overline{n}|}}. \text{ At time } t: {}_tV_x = A_{40+t:\overline{10-t}|}^1 - \pi \cdot \ddot{a}_{40+t:\overline{10-t}|}$$

```

1 from pyliferisk import *
2 from pyliferisk.mortalitytables import GKM80
3
4 nt = Actuarial(nt=GKM80, i=0.06)
5 x = 30
6 n = 10
7 Cm = 1000000
8
9 Premium = Cm * Axn(nt, x, n) / annuity(nt, x, n, 0) # fixed
    premium
10
11 def Reserve(t):
12     return round(Cm * Axn(nt, x+t, n-t)
13                 - Premium * annuity(nt, x+t, n-t, 0), 2)
14
15 for t in range(0, n):
16     print(t, Reserve(t))

```

```

0 0.0
1 218.48
2 436.35
3 644.58
4 824.56
5 957.58
6 1019.75
7 985.7
8 826.45
9 510.18

```

### Actuarial notation vs. Syntax formula

Notation	Description	Syntax
$\ddot{a}_{x:\overline{n} }$	n-year temporary life annuity-due	annuity(nt,x,n,0)
$a_{x:\overline{n} }$	n-year temporary life annuity	annuity(nt,x,n,1)
$\ddot{a}_{x:\overline{n} }^{(m)}$	n-year annuity-due m-monthly payments	annuity(nt,x,n,0,m)
$\ddot{a}_x$	whole life annuity-due	annuity(nt,x,'w',0)
$a_x$	whole life annuity	annuity(nt,x,'w',1)
$\ddot{a}_x^{(m)}$	whole life annuity-due m-monthly	annuity(nt,x,'w',0,m)
$a_x^{(m)}$	whole life annuity m-monthly	annuity(nt,x,'w',1,m)
$n \ddot{a}_x$	d-year deferred whole life annuity-due	annuity(nt,x,n,0,-d)
$n a_x$	d-year deferred whole life annuity	annuity(nt,x,n,1,-d)
$n \ddot{a}_{x:\overline{n} }^{(m)}$	d-year deferred n-year temporal annuity-due m-monthly payments	annuity(nt,x,n,0,m,-d)
$n \ddot{a}_x$	d-year deferred whole life annuity-due	annuity(nt,x,'w',0,-d)
$n a_x$	d-year deferred whole life annuity	annuity(nt,x,'w',1,-d)
<i>Increasing annuities (a sample of them)</i>		
${}^q\ddot{a}_x$	geometrically increasing whole-life annuity-due	annuity(nt,x,'w',0,['g',c])
$n {}^q\ddot{a}_x$	d-year deferred geometrically increasing whole-life annuity-due	annuity(nt,x,'w',0,['g',c],-d)
${}^q n \ddot{a}_{x:\overline{n} }^{(m)}$	d-year deferred geometrically increasing n-year temporal annuity-due m-monthly payments	annuity(nt,x,n,0,m,['g',c],-d)

## 2.4 Pure endowment: Deferred capital

### 2.4.1 formula `nEx()`

<b>Description</b>	Returns the EPV of a pure endowment (deferred capital). Pure endowment benefits are conditional on the survival of the policyholder.
<b>Actuarial notation</b>	${}_nE_x$
<b>Usage</b>	<code>nEx(mt, x, n)</code>
<b>Args</b>	mt: Mortality table. x: the age as integer number. n: period in years.
<b>Example</b>	<code>mt = Actuarial(nt=SPAININE2004, i=0.02)</code> <code>nEx(mt, 50, 10)</code>

#### Syntax:

Notation	Description	Syntax
$A_{x:\overline{n} }$	Pure endowment (Deferred capital)	<code>nEx(nt, x, n)</code>
${}_nE_x$	EPV of a pure endowment (deferred capital)	<code>nEx(nt, x, n)</code>

#### Example:

A deferred capital premium calculation:

```

1  from pyliferisk import *
2  from pyliferisk.mortalitytables import GKM80
3
4  C = 1000
5  x = 60
6  n = 25
7  exp = 0.2 / 100          # expenses over capital
8  com = 0.10              # commision over premium
9
10 mt = Actuarial(nt=GKM80, i=0.025)
11
12 def Premium(mt, x, n):
13     return (nEx(mt, x, n) + Axn(mt, x, n)) / annuity(mt, x, n, 0) *
14         C
15
16 print((Premium(mt, x, n) + C * exp) / (1 - com))

```

Returns:

```
58.8146911381352
```

## 2.5 Commutation factors

Nowadays, the standard techniques for actuarial calculations use cashflow projections. In fact, the use of interest rates curves is required for the calculation of the technical provisions for insurance obligations (such as risk-free interest rate term structures in Solvency II framework) where commutation factors are not adequate.

Despite commutation factors may seem quite prehistoric, it may be useful for academic purposes or replicating older products exactly “to the letter”. For this reason, the `pyliferisk` library includes a list of commutations factors (`Dx`, `Nx`, `Cx`, `Mx`) in order to facilitate the migration from older actuarial software (such as Cobol or Cactus) to Python, or for simple calculations (where your model shouldn't be subject to future changes). If your goal is to reduce the timing, there are a lot of other aspects where you can save time.

```
1 from pyliferisk import *
2 from pyliferisk.mortalitytables import GKM95
3
4 tariff = Actuarial(nt=GKM95, i=0.02)
5
6 print(tariff.Dx[50])
7 print(tariff.Nx[50])
8 print(tariff.Cx[50])
9 print(tariff.Mx[50])
```

Returns:

```
35633.87396668312
844266.356048293
109.8353338458543
19269.483448767027
```

## 2.6 Other functions

Other functions can be derived from the  $lx$  figures. Anyway, the library include the following additional formulas:

- `px(mt, x)`: Returns the probability of surviving within 1 year ( $p_x$ ).
- `tpx(mt, x, t)`: Returns the probability that  $x$  will survive within  $t$  years ( ${}_tp_x$ ).
- `tqx(mt, x, t)`: Returns the probability to die within  $n$  years at age  $x$  ( ${}_tq_x$ ).
- `tqxn(mt, x, n, t)`: Probability to die in  $n$  years being alive at age  $x$  ( ${}_n|q_x$ ).
- `mx(mt, x)`: Returns the central mortality rate ( $m_x$ ).

## 2.7 Help (Documentation strings)

According Python PEP257 convention, formulas includes a documentation string (also known as docstrings):

```
1 from pyliferisk import *
2
3 print(qx.__doc__)
```

Returns:

```
qx: Returns the probability that a life aged x dies before 1 year
    With the convention: the true probability is qx/1000
Args:
    mt: the mortality table
    x: the age as integer number.
```



## 3 Mortality tables

The package includes a sample of life mortality tables (`mortalitytables.py`), mainly extracted from academic textbooks or contributions. It's possible to add tables as a list or to import tables of an external source, i.e. txt or csv files (see Example 6).

Tables must be imported for use, either all (`import *`) or only which you will use, example:

```
1 from pyliferisk.mortalitytables import GKM95, UK43
```

Notes:

- The probability is  $qx * 1000$ .
- The first item indicates the age when the table starts. For example, UK43 table is 0 for the first 30 ages.

There is a financial table (called FIN) for financial annuities which doesn't include mortality.

In the SOA repository (<http://mort.soa.org>) is available a variety (over 2,500) of rate tables of interest to actuaries: SOA experience mortality and lapse tables, regulatory valuation tables, population tables and various international tables).

### 3.1 Adding tables

#### 3.1.1 Probability of dying tables ( $qx$ )

Probability of dying between age  $x$  and  $x+1$  tables ( $qx$  tables of mortality tables) are added in list format where first item indicates the age when table starts. ie:

SCOT\_DLT\_00\_02.M = [0, 0.006205, 0.000328, 0.00026 ....]

#### 3.1.2 Number of surviving tables ( $lx$ )

In case you need to use an surviving table (with numbers of surviving to age  $x$ ) it's possible using the following instruction:

```
1 from pyliferisk import MortalityTable
2
3 lx_sample = [100000,99500,99250,99200,99000,98900]
4 mt = MortalityTable(lx=lx_sample)
5
6 print(mt.qx[0]/1000)
7 print(mt.qx[1]/1000)
8 print(mt.qx[2]/1000)
```

```
0.005
0.002512
0.000503
```

See Example 6 where an external file is read.

## 3.2 Class .view()

Instruction `.view()` provides a view of the main variables (`qx`, `lx`, `dx`, `ex`, `Dx`, `Nx`, `Cx`, `Mx`, `nEx`) of any mortality table. The default view is `lx` for 10-years. Usage example:

```
1 mt = MortalityTable(nt=INM05)
2 mt.view()
```

```
[x=0]  lx=100000.0
[x=1]  lx=99564.1
[x=2]  lx=99530.7460265
[x=3]  lx=99506.659586
[x=4]  lx=99488.4498673
[x=5]  lx=99473.4271113
[x=6]  lx=99460.6945127
[x=7]  lx=99448.1624651
[x=8]  lx=99435.5325485
[x=9]  lx=99421.9098806
[x=10] lx=99409.3827199
Total number of rows for lx = 107
```

Other view (`qx` from 60 to 65 years-old):

```
1 mt = MortalityTable(nt=INM05)
2 mt.view(60, 65, var = 'qx')
```

```
[x=60]  qx=9.958
[x=61]  qx=10.736
[x=62]  qx=11.199
[x=63]  qx=12.455
[x=64]  qx=13.861
[x=65]  qx=15.224
Total number of rows for qx = 106
```

## 4 Examples

This section includes a list of examples using pyliferisk.

### 4.1 Example 1. Cashflow calculation: Annuity-immediate Geometrically increasing

Prospective Reserves look forward, as the expected present value of future outgo less the expected present value of the future income. Using NumPy library for discount them. Life annuity immediate geometrically increasing for a male 67 years-old, as single premium (PASEM2010, interest 5%):

```
1 import pyliferisk as life
2 from pyliferisk.mortalitytables import PASEM2010M
3 import numpy as np
4
5 mt = life.MortalityTable(nt=PASEM2010M)
6
7 age = 67
8 initial_payment = 8000
9 incr = 0.03                                # increment
10 i = 0.05                                    # interest rate
11
12 discount_factor = []
13 for y in range(0, mt.w-age):
14     discount_factor.append(1 / (1 + i) ** (y + 1))
15
16 payments = [initial_payment]
17 for x in range(0, mt.w-age-1):
18     payments.append(payments[x] * (1 + incr) * (1 - mt.qx[age+
19     x] / 1000))
20
21 print('Premium:', np.dot(payments, discount_factor).round(2))
```

```
Premium: 96718.36
```

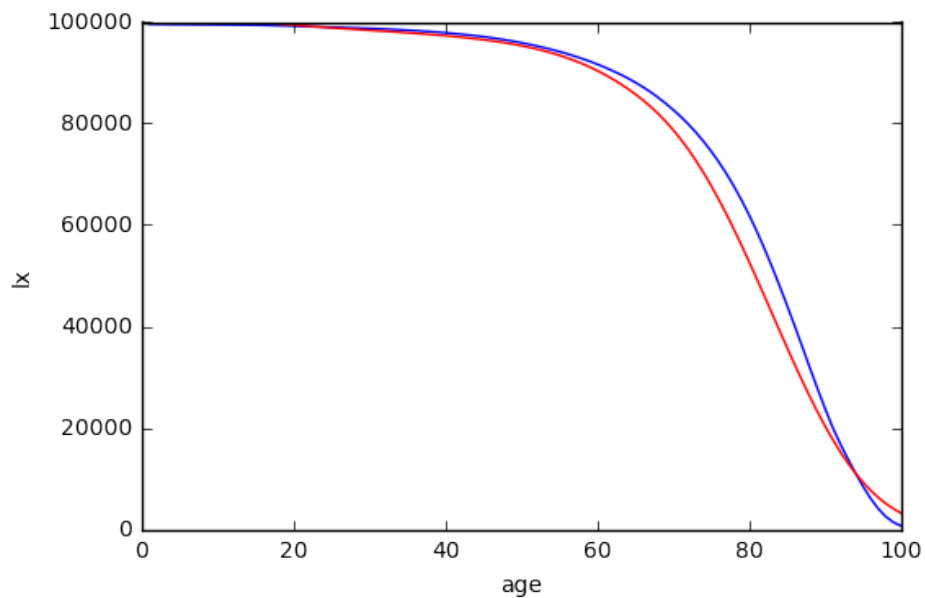
There are different ways of discounting (see previous line 15). In case of:

- not discount first year: the exponent should be only  $y$ .
- discount first year: the exponent should be  $y+1$ .
- discount half year: the exponent should be  $y+0.5$ .

## 4.2 Example 2. Drawing graph with matplotlib

Plotting a surviving graph with pyplot library:

```
1 import matplotlib.pyplot as plt
2 import pyliferisk as life
3 from pyliferisk.mortalitytables import INE2004, GKM95
4
5 tariff = life.MortalityTable(nt=INE2004)
6 experience = life.MortalityTable(nt=GKM95, perc=75)
7 x = range(0, tariff.w)
8 y = tariff.lx[:tariff.w]
9 z = experience.lx[:tariff.w]
10 plt.plot(x,y, color = 'blue')
11 plt.plot(x,z, color = 'red')
12 plt.ylabel('lx')
13 plt.xlabel('age')
```



### 4.3 Example 3. Obtain data from plain text file

Term Life with maturity capital benefit and similar death capital.

```
1 from pyliferisk import *
2 from pyliferisk.mortalitytables import INE2004, GKM95
3 import csv
4
5 mt = Actuarial(nt=GKM80, i=0.04)
6
7 def single_risk_premium(x, n):
8     return nEx(mt, x, n) + Axn(mt, x, n)
9
10 def annual_risk_premium(x, n):
11     return (single_risk_premium(x, n) / annuity(mt, x, n, 0))
12
13 SingleRiskPrem=[]
14 AnnualRiskPrem=[]
15
16 columns = '{0:8} {1:2} {2:9} {3:8} {4:10} {5:10}'
17 print(columns.format('Contract', 'Age', 'Duration', 'Capital', '
18     Single Pr', 'Annual Pr'))
19 print('--' * 26)
20
21 with open('collective.csv', 'r') as file:
22     collective = csv.DictReader(file, delimiter=';')
23     for row in collective:
24         age = int(row['age'])
25         dur = int(row['duration'])
26         capital = int(row['capital'])
27         single_premium = round(capital * single_risk_premium(age,
28             dur), 2)
29         annual_premium = round(capital * annual_risk_premium(age,
30             dur), 2)
31         AnnualRiskPrem.append(annual_premium)
32         print(columns.format(row['N_pol'], age, dur, capital,
33             single_premium, annual_premium))
34
35 print('--'*26)
36 print('Total Annual Premium:', sum(AnnualRiskPrem))
```

Contract	Age	Duration	Capital	Single Pr	Annual Pr
00001	42	10	2000000	1362199.46	163939.94
00002	35	15	1500000	842908.92	73904.7
00003	51	12	1000000	643890.86	69150.37
00004	31	5	3500000	2878725.12	623332.65
00005	37	20	2000000	943160.32	68500.76
-----					
Total Annual Premium: 998828.42					

For example, this run spent 0m0.024s in a MacBook Pro 2,6 GHz Intel Core i5 8 GB 1600 MHz DDR3.

#### 4.4 Example 4. Obtain the Risk-free rate from MS Excel

Obtain the risk-free rate from the EIOPA Excel file at 30-06-2023 with pandas. Pandas is an open source library providing high-performance, easy-to-use data structures, and data analysis tools. <http://pandas.pydata.org>

```
1 import pandas as pd
2
3 df = pd.read_excel('EIOPA_RFR_20230630_Term_Structures.xlsx',
4                   sheet_name='RFR_spot_no_VA', skiprows=9, usecols='C:C', names=[
5                       'Euro'])
6 df['Disc_factor'] = (1 + df['Euro']) ** df.index
7
8 print(df.head(20))
```

	Euro	Disc_factor
0	0.03983	1.000000
1	0.03772	1.037720
2	0.03501	1.071246
3	0.03282	1.101727
4	0.03132	1.131290
5	0.03037	1.161358
6	0.02970	1.191967
7	0.02926	1.223702
8	0.02897	1.256671
9	0.02879	1.291043
10	0.02869	1.326921
11	0.02859	1.363532
12	0.02851	1.401207
13	0.02839	1.438971
14	0.02819	1.475799
15	0.02788	1.510554
16	0.02752	1.543990
17	0.02716	1.577058
18	0.02684	1.610831
19	0.02660	1.646736

## 4.5 Example 5. Cashflow Calculation Reserving

Reserving for a Whole Life contract using a lineal interest rate and the risk-free rate obtained in the previous example.

Once again, the following two examples should be enough clear:

### Using lineal interest rate

```
1  from pyliferisk import *
2  from pyliferisk.mortalitytables import INM05
3  import numpy as np
4
5  tariff = Actuarial(nt=INM05, i=0.05)
6  reserve = MortalityTable(nt=INM05)
7  age = 32                                # age
8  Cd = 3000                              # capital death
9  Premium = Cd * Ax(tariff, 25) / annuity(tariff, 25, 'w', 0) #fixed
    at age 25
10
11  qx_vector = []
12  px_vector = []
13  for i in range(age, reserve.w + 1):
14      qx = ((reserve.lx[i] - reserve.lx[i+1]) / reserve.lx[age])
15      qx_vector.append(qx)
16      qx_sum = sum(qx_vector)
17      px_vector.append(1 - qx_sum)
18
19  def Reserve(i):
20      discount_factor = []
21      for y in range(0, reserve.w-age + 1):
22          discount_factor.append(1 / (1 + i) ** (y + 1))
23
24      APV_Premium = np.dot(Premium, px_vector)
25      APV_Claims = np.dot(Cd, qx_vector)
26      # Reserve = APV(Premium) - APV(Claim)
27      return np.dot(discount_factor, np.subtract(APV_Claims,
28          APV_Premium)).round(2)
29
30  print(Reserve(0.0191))
31  print(Reserve(0.0139))
```

```
820.8
1088.37
```

## Including risk free rate curve

```
1 from pyliferisk import *
2 from pyliferisk.mortalitytables import INM05
3 import numpy as np
4 import pandas as pd
5
6 rfr = pd.read_excel('EIOPA_RFR_20230630_Term_Structures.xlsx',
7                     sheet_name='RFR_spot_no_VA', skiprows=9, usecols='C:C', names=[
8                         'Euro'])
9
10 tariff = Actuarial(nt=INM05, i=0.05)
11 reserve = MortalityTable(nt=INM05)
12 x = 32 # age
13 Cd = 3000 # capital death
14 Premium = Cd * Ax(tariff, 25) / annuity(tariff, 25, 'w', 0)
15
16 qx_vector = []
17 px_vector = []
18 for i in range(x, reserve.w + 1):
19     qx = ((reserve.lx[i] - reserve.lx[i+1]) / reserve.lx[x])
20     qx_vector.append(qx)
21     qx_sum = sum(qx_vector)
22     px_vector.append(1 - qx_sum)
23
24 def Reserve(i):
25     discount_factor = []
26     for y in range(0, reserve.w - x + 1):
27         if isinstance(i, float):
28             discount_factor.append(1 / (1 + i) ** (y + 1))
29         elif i == 'rfr':
30             discount_factor.append(1 / (1 + rfr['Euro']
31                                     ][y]) ** (y + 1))
32
33     APV_Premium = np.dot(Premium, px_vector)
34     APV_Claims = np.dot(Cd, qx_vector)
35     return np.dot(discount_factor, np.subtract(APV_Claims,
36                                                 APV_Premium)).round(2)
37
38 print(Reserve(0.0191))
39 print(Reserve(0.0139))
40 print(Reserve('rfr'))
```

```
820.8
1088.37
474.57
```



## 4.6 Example 6. Import a mortality table (lx) using pandas

Normally, we will have the tables in an external file in xls or csv format. In this case, we can use the instruction `read_csv` or `read_xls` of pandas.

Example 3.3.1 from Actuarial Mathematics, by Newton L. Bowers (et al.), Society of Actuaries, 1997 (pg. 64)

```
1 import pandas as pd
2 from pyliferisk import MortalityTable
3
4 USLife1979 = pd.read_csv('uslife79.csv')
5 USLife79lx = USLife1979['lx'].tolist()
6
7 mt = MortalityTable(lx=USLife79lx)
8
9 print('Live to 100:', mt.lx[100]/mt.lx[20])
10 print('Die before 70:', 1 - mt.lx[70]/mt.lx[20])
11 print('Die in the tenth decade of life:', (mt.lx[90]-mt.lx[100])/
12       mt.lx[20])
13
14 print('Probability of death between 50 and 51:',
15       (mt.lx[50] - mt.lx[51]) / mt.lx[50])
16 print('Likewise with qx:', mt.qx[50] / 1000)
17 print('Ultimate age:', mt.w)
```

```
Live to 100: 0.0118
Die before 70: 0.30178
Die in the tenth decade of life: 0.1330
```

Additional exercises not included in book:

```
print('Probability of death between 50 and 51:',
      (mt.lx[50] - mt.lx[51]) / mt.lx[50])
print('Likewise with qx:', mt.qx[50] / 1000)
print('Ultimate age:', mt.w)
```

```
Probability of death between 50 and 51: 0.0058999
Likewise with qx: 0.0058999
Ultimate age: 109
```

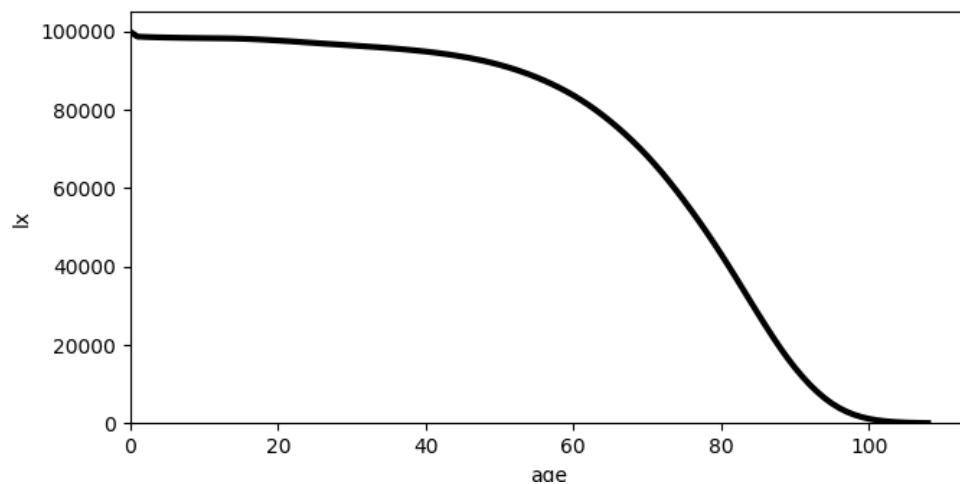
Figure 3.3.3 from Actuarial Mathematics, by Newton L. Bowers (et al.), 1997 (pg. 65)

```
#!/usr/bin/python
import pandas as pd
from pyliferisk import MortalityTable
import matplotlib.pyplot as plt

USLife1979 = pd.read_csv('uslife79.csv')
USLife79lx = USLife1979['lx'].tolist()

mt = MortalityTable(lx=USLife79lx)

x = range(0, mt.w)
y = mt.lx[:mt.w]
plt.plot(x, y, linewidth=3, color='0')
plt.ylabel('lx')
plt.xlabel('age')
plt.show()
```



## 4.7 Example 7. Cashflows in pandas

Replicating Example 1 with Pandas.

Life annuity immediate geometrically increasing for a male 67 years-old, as single premium (PASEM2010, interest 5%).

```

1  from pyliferisk import *
2  from pyliferisk.mortalitytables import SPAIN_PASEM2010M
3  import pandas as pd
4
5  mt = MortalityTable(nt=SPAIN_PASEM2010M)
6
7  age = 67
8  init_payment = 8000
9  incr = 0.03          # increment annuitie
10 i = 0.05             # interest rate
11
12 period = len(range(0, mt.w - age))
13
14 df = pd.DataFrame(pd.date_range('1/1/2023', periods=period, freq='
    Y'), columns=['date'])
15 df['t'] = df.index
16 df['age'] = pd.Series(list(range(age, mt.w)))
17 df['px'] = df['t'].apply(lambda t: 1 - mt.qx[age+t] / 1000)
18 df['px_cumprod'] = df['px'].cumprod()
19 df['disc_factor'] = df['t'].apply(lambda t: 1 / (1 + i) ** (t + 1)
    )
20 df['capital'] = df['t'].apply(lambda t: init_payment * ((1 + incr)
    ** t))
21 df['payments'] = df['t'].apply(lambda t: init_payment if t == 0
    else
22         df['capital'][t] * df['px_cumprod'][t-1])
23 df['apv_payments'] = df['payments'] * df['disc_factor']
24 premium = df['apv_payments'].sum().round(2)
25
26 print(df.head())
27 print('Premium:', premium)
28
29 df.to_excel('output.xlsx')

```

	date	t	age	px	...	disc_factor	capital	payments
0	2023-12-31	0	67	0.984336	...	0.952381	8000.0000	8000.0000
								7619.047619
1	2024-12-31	1	68	0.982438	...	0.907029	8240.0000	8110.6320
								7356.582313
2	2025-12-31	2	69	0.980193	...	0.863838	8487.2000	8207.1224
								7089.620905
3	2026-12-31	3	70	0.977540	...	0.822702	8741.8160	8286.3674
								6817.214967
4	2027-12-31	4	71	0.974395	...	0.783526	9004.0705	8343.1717
								6537.093338
								Premium: 96718.36

Export the DataFrame to a MS Excel file:

```
df.to_excel('output.xlsx')
```

H6			<b>f<sub>x</sub> Σ - =</b>	9004,07048							
	A	B	C	D	E	F	G	H	I	J	
1		date	t	age	px	px_cumprod	disc_factor	capital	payments	apv_payments	
2	0	31/12/23	0	67	0,98434	0,98434	0,95238	8000,00	8000,00	7619,05	
3	1	31/12/24	1	68	0,98244	0,96705	0,90703	8240,00	8110,93	7356,85	
4	2	31/12/25	2	69	0,98019	0,94789	0,86384	8487,20	8207,54	7089,98	
5	3	31/12/26	3	70	0,97754	0,92661	0,82270	8741,82	8286,32	6817,18	
6	4	31/12/27	4	71	0,9744	0,90288	0,78353	9004,07	8343,22	6537,13	
7	5	31/12/28	5	72	0,97065	0,87638	0,74622	9274,19	8373,48	6248,42	
8	6	31/12/29	6	73	0,96617	0,84673	0,71068	9552,42	8371,51	5949,48	
9	7	31/12/30	7	74	0,9608	0,81353	0,67684	9838,99	8330,93	5638,70	
10	8	31/12/31	8	75	0,95436	0,77641	0,64461	10134,16	8244,47	5314,46	
11	9	31/12/32	9	76	0,94666	0,73499	0,61391	10438,19	8104,26	4975,31	
12	10	31/12/33	10	77	0,93745	0,68901	0,58468	10751,33	7902,10	4620,19	
13	11	31/12/34	11	78	0,92647	0,63835	0,55684	11073,87	7630,02	4248,68	
14	12	31/12/35	12	79	0,91345	0,58310	0,53032	11406,09	7281,03	3861,29	
15	13	31/12/36	13	80	0,90319	0,52665	0,50507	11748,27	6850,41	3459,92	
16	14	31/12/37	14	81	0,89182	0,46968	0,48102	12100,72	6372,81	3065,43	
17	15	31/12/38	15	82	0,87931	0,41299	0,45811	12463,74	5853,91	2681,74	
18	16	31/12/39	16	83	0,86558	0,35748	0,43630	12837,65	5301,83	2313,17	
19	17	31/12/40	17	84	0,85052	0,30404	0,41552	13222,78	4726,85	1964,10	
20	18	31/12/41	18	85	0,83395	0,25356	0,39573	13619,46	4140,87	1638,68	
21	19	31/12/42	19	86	0,81581	0,20685	0,37689	14028,05	3556,88	1340,55	
22	20	31/12/43	20	87	0,79608	0,16467	0,35894	14448,89	2988,77	1072,80	
23	21	31/12/44	21	88	0,77485	0,12759	0,34185	14882,36	2450,67	837,76	
24	22	31/12/45	22	89	0,7523	0,09599	0,32557	15328,83	1955,87	636,77	
25	23	31/12/46	23	90	0,72864	0,06994	0,31007	15788,69	1515,53	469,92	
26	24	31/12/47	24	91	0,70418	0,04925	0,29530	16262,35	1137,41	335,88	
27	25	31/12/48	25	92	0,67915	0,03345	0,28124	16750,22	824,97	232,01	
28	26	31/12/49	26	93	0,65376	0,02187	0,26785	17252,73	577,08	154,57	
29	27	31/12/50	27	94	0,62817	0,01374	0,25509	17770,31	388,59	99,13	
30	28	31/12/51	28	95	0,60245	0,00828	0,24295	18303,42	251,42	61,08	

## 5 Books

The author has checked the library with examples from the following textbooks:

- Actuarial Mathematics for Life Contingent Risks (David C. M. Dickson, Mary R. Hardy and Howard R. Waters) Cambridge University Press, 2009.
- Actuarial Mathematics (2nd Edition), Newton L. Bowers (et al.), Society of Actuaries, 1997.
- Matemática de los Seguros de Vida, (Gil Fana J.A., Heras Matínez A. and Vilar Zanón J.L.) Fundación Mapfre Estudios, 1999.

It will be documented in the examples folder. Contributions are greatly appreciated.

## 6 Acknowledgements

My special thanks for all the contributions, suggestions and discussion to Florian Pons (*France*), Mason Borda (*US*), Sunil Subbakrishna (*US*), Arief Anbiya (*Indonesia*) and Marios Cheristanidis.