Package **Pyliferisk**

September 22, 2020

Type:	Python Package
Version:	1.11
Author:	Francisco Gárate
Contact:	fgaratesantiago@gmail.com
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.

Contents

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:

```python
from pyliferisk import *
from pyliferisk.mortalitytables import *

tariff = MortalityTable(nt=GKM95)
```

Option 2 (using alias):

```python
import pyliferisk as lf
import pyliferisk.mortalitytables as mort

tariff = lf.MortalityTable(nt=mort.GKM95)
```

Option 3, if only like to use specific functions:

```python
from pyliferisk import MortalityTable
from pyliferisk.mortalitytables import GKM95

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 (qx, lx...) 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:

```
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:

```
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]`

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

2.1.2 *method* `.lx[x]`

Description	Returns the number of survivors at beginning of age x
Actuarial notation	l_x
Usage	`mt.lx[x]`
Args	x: the age as integer number.
Example	`tariff.lx[50]`

2.1.3 *method* `.w`

Description	ultimate age (lw = 0)
Actuarial notation	w
Usage	`mt.w`
Example	`tariff.w`

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

Description	Returns the number of dying at beginning of age x
Actuarial notation	d_x
Usage	`mt.dx[x]`
Args	x: the age as integer number.
Example	`tariff.dx[50]`

2.1.5 *method* `.ex[x]`

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

Example:

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

```
from pyliferisk import MortalityTable
from pyliferisk.mortalitytables import SPAININE2004, GKM95

tariff = MortalityTable(nt=SPAININE2004)
experience = MortalityTable(nt=GKM95, perc=85)

# Print the omega (limiting age) of the both tables:
print(tariff.w)
print(experience.w)

# Print the qx at 50 years old:
print(tariff.qx[50] / 1000)
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.

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

2.2.1 *formula* `Ax()`

Description	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.
Actuarial notation	A_x
Usage	`Ax(mt, x)`
Args	mt: Mortality table. x: the age as integer number.
Example	`mt = Actuarial(nt=SPAININE2004, i=0.02)` `Ax(mt, 50)`

2.2.2 *formula* `Axn()`

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

2.2.3 *formula* `qAx()`

Description This function evaluates the APV of a geometrically increasing annual annuity-due.

Actuarial notation ${}_qA_x$

Usage `Axn(mt, x, q)`

Args
mt: Mortality table.
x: the age as integer number.
q: increase rate.

Example
```
mt = Actuarial(nt=SPAININE2004, i=0.02)
Axn(mt, 50, 0.03)
```

2.2.4 *formula* `qAxn()`

Description This function evaluates the APV of a geometrically increasing Term insurance.

Actuarial notation ${}_qA_{x:\overline{n}|}$

Usage `qAxn(nt, x, n, q)`

Args
mt: Mortality table.
x: the age as integer number.
n: period in years.
q: increase rate.

Example
```
mt = Actuarial(nt=SPAININE2004, i=0.02)
Axn(mt, 50, 10, 0.03)
```

2.2.5 *formula* `AExn()`

Description Returns the EPV of an endowment insurance.
An endowment insurance provides a combination of a term insurance
and a pure endowment

Actuarial notation $A_{x:\overline{n}|}$

Usage `AExn(mt, x, n)`

Args
mt: Mortality table.
x: the age as integer number.
n: period in years.

Example
```
mt = Actuarial(nt=SPAININE2004, i=0.02)
AExn(mt, 50, 10)
```

Syntax:

Notation	Description	Syntax
A_x	whole-life death insurance	`Ax(nt, x)`
$A^1_{x:\overline{n}|}$	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:

```
from pyliferisk import Actuarial, Ax
from pyliferisk.mortalitytables import GKM95

mt = Actuarial(nt=GKM95, i=0.02)
x = 50          #age
C = 1000        #capital

print(Ax(mt, x) * C)
```

```
589.0804423991423
```

2) A term insurance single premium:

```
from pyliferisk import Actuarial, Axn
from pyliferisk.mortalitytables import GKM95

mt = Actuarial(nt=GKM95, i=0.03)
x = 40          #age
n = 20          #horizon
C = 10000       #capital

print(Axn(mt, x, n) * C)
```

```
646.1486398262324
```

3) Example 2 with cashflow approach:

```python
import numpy as np
import pyliferisk as lf
from pyliferisk.mortalitytables import GKM95


mt = lf.MortalityTable(nt=GKM95)
x = 40          #age
n = 20          #horizon
C = 10000       #capital
i = 0.03        #interest rate

payments = []
for t in range(0, n):
   payments.append((mt.lx[x+t] - mt.lx[x+t+1]) / mt.lx[x] * C)

discount_factor = []
for y in range(0, n):
   discount_factor.append(1 / (1 + i) ** (y + 0.5))

print(np.dot(discount_factor, payments).round(2))
```

```
646.1486398262326
```

To print the results year by year:

```python
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()`

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

```
import numpy as np
import pyliferisk as lf
from pyliferisk.mortalitytables import FIN

mt = lf.Actuarial(nt=FIN, i=0.12/12) #FIN = Financial table

n = 5 * 12
C = 100

print(lf.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 (GKM95, interest 6%):

Actuarial equivalence: $\pi^{1}_{30:\overline{10|}} = 1000 \cdot \dfrac{A^{1}_{30:\overline{10|}}}{\ddot{a}_{30:\overline{10|}}}$

```
import pyliferisk as lf
from pyliferisk.mortalitytables import GKM95

nt = lf.Actuarial(nt=GKM95, i=0.06)
x = 30
n = 10
C = 1000

print(C * (lf.Axn(nt, x, n) / lf.annuity(nt, x, n, 0)))
```

```
1.398266715155939
```

3) Reserving a life risk insurance with regular premium. Applying the equivalence principle, where: ${}_0V_x = A_x - \pi \cdot \ddot{a}_x = 0$ and: $\pi = \dfrac{A^{1}_{x:\overline{n|}}}{\ddot{a}_{x:\overline{n|}}}$. At time t: ${}_tV_x = A^{\;1}_{40+t:\overline{10-t|}} - \pi \cdot \ddot{a}_{40+t:\overline{10-t|}}$

```
from pyliferisk import *
from pyliferisk.mortalitytables import GKM95

nt = Actuarial(nt=GKM95, i=0.03)
x = 40
n = 20
Cm = 100000
Premium = Cm * Axn(nt, x, n) / annuity(nt, x, n, 0)  #fixed premium

def Reserve(t):
    return round(Cm * Axn(nt, x+t, n-t)
                                    - Premium * annuity(nt, x+t, n-t, 0),2)

for t in range(0, n+1):
    print(t, Reserve(t))
```

```
0 0.0
1 257.11
2 509.4
3 755.01
4 991.91
5 1217.65
6 1429.31
7 1623.49
8 1796.3
9 1943.34
10 2059.65
11 2139.73
12 2177.5
13 2166.2
14 2098.4
15 1966.05
16 1760.38
17 1471.86
18 1090.07
19 603.61
20 0.0
```

Actuarial notation vs. Syntax formula

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

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

Syntax:

Notation	Description	Syntax
$A_{x:\overset{1}{\overline{n}|}}$	Pure endowment (Deferred capital)	`nEx(nt, x, n)`
${}_nE_x$	EPV of a pure endowment (deferred capital)	`nEx(nt, x, n)`

Example:

A deferred capital premium calculation:

```
from pyliferisk import *
from pyliferisk.mortalitytables import GKM80

C = 1000
x = 60
n = 25
exp = 0.2 / 100           # expenses over capital
com =  0.10               # commision over premium

mt = Actuarial(nt=GKM80, i=0.025)

def Premium(mt, x, n):
  return (nEx(mt, x, n) + Axn(mt, x, n)) / annuity(mt, x, n, 0) * C

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.

```python
import pyliferisk as lf
from pyliferisk.mortalitytables import GKM95

tariff = lf.Actuarial(nt=GKM95, i=0.02)

print(tariff.Dx[50])
print(tariff.Nx[50])
print(tariff.Cx[50])
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):

```
from pyliferisk import *

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:

```python
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:

```python
from pyliferisk import MortalityTable

lx_sample = [100000,99500,99250,99200,99000,98900]
mt = MortalityTable(lx=lx_sample)

print(mt.qx[0]/1000)
print(mt.qx[1]/1000)
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:

```
mt = MortalityTable(nt=INM05)
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):

```
mt = MortalityTable(nt=INM05)
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 (GKM95, interest 5%):

```
1  import pyliferisk as lf
2  from pyliferisk.mortalitytables import GKM95
3  import numpy as np
4
5  mt = lf.MortalityTable(nt=GKM95)
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)
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+x] /
               1000))
19
20 print('Premium:', np.dot(payments, discount_factor).round(2))
```

```
Premium: 100488.39
```

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

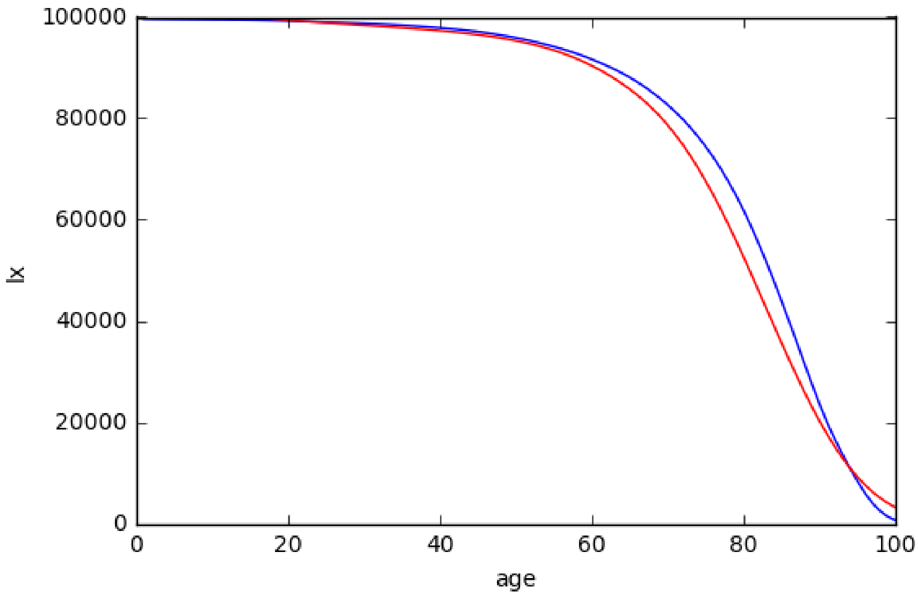
- not discount first year: the exponent should be only `y` and 100 488.39 as result.
- discount first year: the exponent should be `y+1` and 95 703.22 as result.
- discount half year: the exponent should be `y+0.5` and 98 066.62 as result.

4.2 Example 2. Drawing graph with matplotlib

Plotting a surviving graph with pyplot library:

```python
import matplotlib.pyplot as plt
import pyliferisk as lf
from pyliferisk.mortalitytables import SPAININE2004, GKM95

tariff = lf.MortalityTable(nt=SPAININE2004)
experience = lf.MortalityTable(nt=GKM95, perc=75)
x = range(0, tariff.w)
y = tariff.lx[:tariff.w]
z = experience.lx[:tariff.w]
plt.plot(x,y, color = 'blue')
plt.plot(x,z, color = 'red')
plt.ylabel('lx')
plt.xlabel('age')
plt.show()
```



4.3 Example 3. Obtain data from plain text file

Term Life with maturity capital benefit and similar death capital.

```
from pyliferisk import *
from pyliferisk.mortalitytables import SPAININE2004, GKM95
import csv

mt = Actuarial(nt=GKM95, i=0.04)

def single_risk_premium(x, n):
   return nEx(mt, x, n) + Axn(mt, x, n)

def annual_risk_premium(x, n):
   return (single_risk_premium(x, n) / annuity(mt, x, n, 0))

SingleRiskPrem=[]
AnnualRiskPrem=[]

columns = '{0:8} {1:2} {2:9} {3:8} {4:10} {5:10}'
print(columns.format('Contract', 'Age', 'Duration', 'Capital', 'Single Pr
    ', 'Annual Pr'))
print('--' * 26)

with open('colective.csv', 'r') as file:
     colective = csv.DictReader(file, delimiter=';')
     for row in colective:
          age = int(row['age'])
          dur = int(row['duration'])
          capital = int(row['capital'])
          single_premium = round(capital * single_risk_premium(age, dur),
              2)
          annual_premium = round(capital * annual_risk_premium(age, dur),
              2)
          AnnualRiskPrem.append(annual_premium)
          print(columns.format(row['N_pol'], age, dur, capital,
              single_premium, annual_premium))

print('--'*26)
print('Total Annual Premium:', sum(AnnualRiskPrem))
```

```
Contract Age Duration  Capital  Single Pr   Annual Pr
----------------------------------------------------
00001    42        10  2000000 1358356.77   162845.92
00002    35        15  1500000  840608.56    73547.58
00003    51        12  1000000  637123.78    67529.26
00004    31         5  3500000 2878340.45   623281.52
00005    37        20  2000000  934625.98    67482.69
----------------------------------------------------
Total Annual Premium: 994686.97
```

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 31-12-2019 with Pandas. `Pandas` (Python Data Analysis) is an open source library providing high-performance, easy-to-use data structures, and data analysis tools. `http://pandas.pydata.org`

```
import pandas as pd

df = pd.read_excel('EIOPA_RFR_20191231_Term_Structures.xlsx', sheet_name=
    'RFR_spot_no_VA', skiprows=9, usecols='C:C', names=['Euro'])
df['Disc_factor'] = (1 + df['Euro']) ** df.index

print(df.head(20))
```

```
       Euro  Disc_factor
0  -0.00421     1.000000
1  -0.00391     0.996090
2  -0.00338     0.993251
3  -0.00285     0.991474
4  -0.00229     0.990871
5  -0.00164     0.991827
6  -0.00084     0.994971
7  -0.00018     0.998741
8   0.00047     1.003766
9   0.00113     1.010216
10  0.00164     1.016522
11  0.00213     1.023681
12  0.00268     1.032638
13  0.00321     1.042543
14  0.00362     1.051890
15  0.00389     1.059966
16  0.00409     1.067486
17  0.00431     1.075852
18  0.00460     1.086118
19  0.00500     1.099399
```

Even is possible to read directly from any site with the urllib2, Scrapy or BeautifulSoup4 (a screen-scraping libraries for parsing HTML and XML).

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

```
from pyliferisk import *
from pyliferisk.mortalitytables import INM05
import numpy as np

tariff  = Actuarial(nt=INM05, i=0.05)
reserve = MortalityTable(nt=INM05)
age = 32                        # age
Cd = 3000                       # capital death
Premium = Cd * Ax(tariff, 25) / annuity(tariff, 25, 'w', 0) #fixed at age 25

qx_vector = []
px_vector=[]
for i in range(age, reserve.w + 1):
        qx = ((reserve.lx[i] - reserve.lx[i+1]) / reserve.lx[age])
        qx_vector.append(qx)
        qx_sum = sum(qx_vector)
        px_vector.append(1 - qx_sum)

def Reserve(i):
        discount_factor = []
        for y in range(0, reserve.w-age + 1):
                discount_factor.append(1 / ( 1 + i) ** y)

        APV_Premium = np.dot(Premium, px_vector)
        APV_Claims = np.dot(Cd, qx_vector)
        # Reserve = APV(Premium) - APV(Claim)
        return np.dot(discount_factor, np.subtract(APV_Claims, APV_Premium)).round(2)

print(Reserve(0.0191))
print(Reserve(0.0139))
```

```
847.8
1116.07
```

Including risk free rate curve

```
#!/usr/bin/python
from pyliferisk import *
from pyliferisk.mortalitytables import INM05
import numpy as np
import pandas as pd

rfr = pd.read_excel('EIOPA_RFR_20191231_Term_Structures.xlsx', sheet_name
    ='RFR_spot_no_VA', skiprows=9, usecols='C:C', names=['Euro'])

tariff  = Actuarial(nt=INM05, i=0.05)
reserve = MortalityTable(nt=INM05)
x = 32                     # age
Cd = 3000                  # capital death
Premium = Cd * Ax(tariff, 25) / annuity(tariff, 25, 'w', 0)

qx_vector = []
px_vector = []
for i in range(x,reserve.w + 1):
        qx = ((reserve.lx[i]-reserve.lx[i+1]) / reserve.lx[x])
        qx_vector.append(qx)
        qx_sum = sum(qx_vector)
        px_vector.append(1 - qx_sum)

def Reserve(i):
        discount_factor = []
        for y in range(0, reserve.w - x + 1):
                if isinstance(i,float):
                        discount_factor.append(1 / (1 + i) ** y)
                elif i == 'rfr':
                        discount_factor.append(1 / (1 + rfr['Euro'][y])
                            ** y)

        APV_Premium = np.dot(Premium, px_vector)
        APV_Claims = np.dot(Cd, qx_vector)
        return np.dot(discount_factor, np.subtract(APV_Claims,
            APV_Premium)).round(2)

print(Reserve(0.0191))
print(Reserve(0.0139))
print(Reserve('rfr'))
```

```
847.8
1116.07
737.41
```

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)

```
import pandas as pd
import pyliferisk as lf

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

mt = lf.MortalityTable(lx=USLife79lx)

print('Live to 100:', mt.lx[100]/mt.lx[20])
print('Die before 70:', 1 - mt.lx[70]/mt.lx[20])
print('Die in the tenth decade of life:', (mt.lx[90]-mt.lx[100])/mt.lx[20])


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)
```

```
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
```

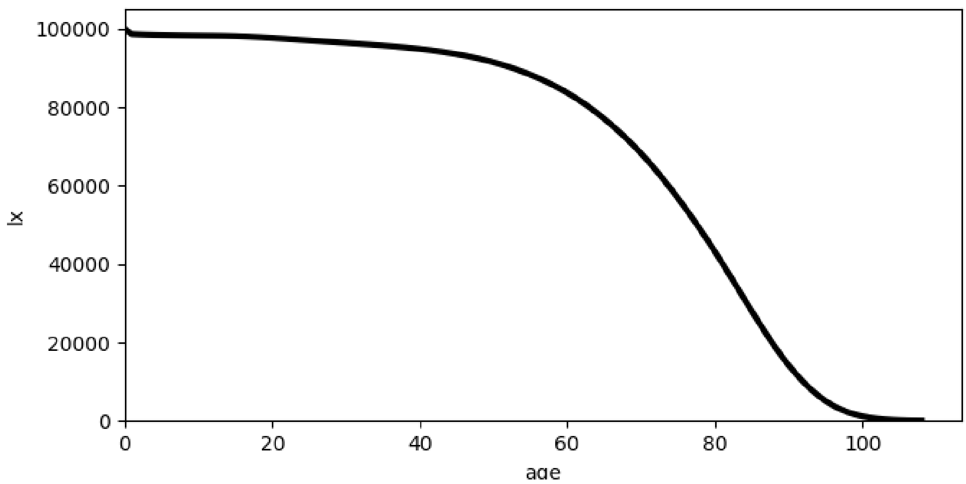
Figure 3.3.3 from Actuarial Mathematics, by Newton L. Bowers (et al.), 1997 (pg. 65)

```
#!/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%).

```python
import pandas as pd
import pyliferisk as lf
from pyliferisk.mortalitytables import GKM95

mt = lf.MortalityTable(nt=GKM95)

age = 67
init_payment = 8000
incr = 0.03                # increment annutie
i = 0.05                   # interest rate

period = len(range(0, mt.w - age))

df = pd.DataFrame(pd.date_range('2019-12-31', periods=period, freq='Y'),
    columns=['date'])
df['t'] = df.index
df['age'] = pd.Series(list(range(age, mt.w)))
df['px'] = df['t'].apply(lambda t: 1 - mt.qx[age+t] / 1000)
df['px_cumprod'] = df['px'].shift(1).fillna(1).cumprod() # px next year
df['disc_factor'] = df['t'].apply(lambda t: 1 / (1 + i) ** t)
df['capital'] = df['t'].apply(lambda t: init_payment * ((1 + incr) ** t))
df['payments'] = df['t'].apply(lambda t: init_payment if t == 0 else
                df['capital'][t] * df['px_cumprod'][t])
df['apv_payments'] = df['payments'] * df['disc_factor']
premium = df['apv_payments'].sum().round(2)

print(df.head())
print('Premium:', premium)
```

```
        date  t  age  ...     capital     payments  apv_payments
0 2019-12-31  0   67  ...  8000.00000  8000.000000   8000.000000
1 2020-12-31  1   68  ...  8240.00000  8055.905216   7672.290682
2 2021-12-31  2   69  ...  8487.20000  8090.127878   7337.984469
3 2022-12-31  3   70  ...  8741.81600  8099.414932   6996.579144
4 2023-12-31  4   71  ...  9004.07048  8080.684695   6647.999296

Premium: 100488.39
```

Export the DataFrame to a MS Excel file:

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

	date	t	age	px	px_cumprod	disc_factor	capital	payments	apv_payments
0	31/12/2019	0	67	0,9776584	1	1	8.000	8.000	8.000
1	31/12/2020	1	68	0,9749982	0,9776584	0,952380952	8.240	8.055,91	7.672,29
2	31/12/2021	2	69	0,9719883	0,95321518	0,907029478	8.487,2	8.090,13	7.337,98
3	31/12/2022	3	70	0,9686286	0,926514003	0,863837599	8.741,82	8.099,41	6.996,58
4	31/12/2023	4	71	0,9649192	0,897447961	0,822702475	9.004,07	8.080,68	6.648,00
5	31/12/2024	5	72	0,96086	0,865964769	0,783526166	9.274,19	8.031,12	6.292,60
6	31/12/2025	6	73	0,956451	0,832070908	0,746215397	9.552,42	7.948,29	5.931,14
7	31/12/2026	7	74	0,9516922	0,795835052	0,71068133	9.838,99	7.830,21	5.564,79
8	31/12/2027	8	75	0,9465837	0,757390011	0,676839362	10.134,16	7.675,51	5.195,09
9	31/12/2028	9	76	0,9411255	0,716933039	0,644608916	10.438,19	7.483,48	4.823,92
10	31/12/2029	10	77	0,9353174	0,674723965	0,613913254	10.751,33	7.254,18	4.453,44
11	31/12/2030	11	78	0,9291596	0,631081065	0,584679289	11.073,87	6.988,51	4.086,04
12	31/12/2031	12	79	0,922652	0,58637503	0,556837418	11.406,09	6.688,24	3.724,26
13	31/12/2032	13	80	0,9157947	0,541020094	0,530321351	11.748,27	6.356,05	3.370,75
14	31/12/2033	14	81	0,9085876	0,495463334	0,505067953	12.100,72	5.995,46	3.028,12
15	31/12/2034	15	82	0,9010307	0,450171842	0,481017098	12.463,74	5.610,82	2.698,90
16	31/12/2035	16	83	0,893124	0,40561865	0,458111522	12.837,65	5.207,19	2.385,47
17	31/12/2036	17	84	0,8848676	0,362267751	0,436296688	13.222,78	4.790,19	2.089,94
18	31/12/2037	18	85	0,8762614	0,320558995	0,415520655	13.619,46	4.365,84	1.814,10
19	31/12/2038	19	86	0,8673055	0,280893474	0,395733957	14.028,05	3.940,39	1.559,35

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)*, Marios Cheristanidis and Amit Agarwal.