Package **Pyliferisk**

June 22, 2017

Type:	Python Package
Version:	1.9
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, a user guide and tutorial all-in-one. Also include a sort description of other useful libraries (NumPy, pandas, matplotlib...) widely used in actuarial calculations.

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 (`lifecontingencies.py`) and has no dependencies other than the Python Standard Library. Additionally, the package includes several life mortality tables (`mortalitytables.py`), mainly extracted from academic textbooks. Nevertheless, other libraries as Numpy or Datatime are required for increasing functionality.

Moreover, additional libraries (such as NumPy) can be included in order to increase the functionality, such as cash flow operations, random number generation, interpolation, etc.

You can find also examples for different contracts in the /examples/ folder.

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, it should be an internal process used extensively where all parts must walk hand in hand.

I highly recommend reading the official introduction to python:
`http://www.python.org/about/`

Potential uses

Python is used by several well-known banks companies for asset valuations. The exact search on Google for "financial modelling in Python" shows more than 65.000 results.

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++. For reusing R implementations, with the library `rpy2` is possible run applications built in R.

Take a look at application domains where Python is used: `http://www.python.org/about/apps/`

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 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 begining 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 begining 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[]`

Description	Returns the curtate expectation of life. Life expectancy
Actuarial notation	e_x
Usage	`mt.ex[]`
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:

```python
#!/usr/bin/python
from pyliferisk.lifecontingencies 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 suvival 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	${}_qA_{x:\overline{n}\rceil}$
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 a endowment insurance. An endowment insurance provides a combination of a term insurance and a pure endowment
Actuarial notation	$A_{x:\overline{n}\rceil}$
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)`
$A^1_{x:\overline{n}\rceil}$	Term insurance	`Axn(nt, x, n)`
$A_{x:\overline{n}\rceil}$	Endowment insurance	`AExn(nt, x, n)`
${}_qA_x$	Increasing whole-life	`qAx(nt, x, n)`
${}_qA_{x:\overline{n}\rceil}$	Increasing Term insurance	`qAxn(nt, x, n, q)`

Examples:

1) A whole-life single premium:

```python
#!/usr/bin/python
from pyliferisk.lifecontingencies 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)
```

```
589.080442399
```

2) A term insurance single premium:

```python
from pyliferisk.lifecontingencies 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)
```

```
646.148639826
```

3) Example 2 with cashflow approach:

```python
from pyliferisk.lifecontingencies import MortalityTable
from pyliferisk.mortalitytables import GKM95
import numpy as np

mt = 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))
```

```
646.148639826
```

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 refer 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 inmediately 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 an 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
	`'w'`	whole-life
4.	0	annuity-immediate
	1	annuity-due
optional args:		
–	m:	Number of fractional payments per period. If missing, m is set as 0.
–	[`'a'`, q]	Arithmetically increasing
	[`'g'`, q]	Geometrically increasing
–	-d	The deferring period in years

Examples:

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

```python
#!/usr/bin/python
from pyliferisk.lifecontingencies import Actuarial, annuity
from pyliferisk.mortalitytables import FIN # FIN: Financial table

mt=Actuarial(nt=FIN,i=0.12/12)

n = 5*12
C = 100

print(annuity(mt,0,n,1) * C)    #replace age 'x' by 0
```

Returns:

```
4495.50384062
```

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

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

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

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

```
1.39826671516
```

3) Reserving a life risk insurance with regular premium. Applying the equivalence principe, 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: ${}_tV_x = A^{1}_{40+t:\overline{10-t|}} - \pi \cdot \ddot{a}_{40+t:\overline{10-t|}}$

```python
#!/usr/bin/python
from pyliferisk.lifecontingencies 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 Cm * Axn(nt,x+t,n-t) - Premium * annuity(nt,x+t,n-t,0)

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

```
0 0.0
1 257.114710498
2 509.39822131
3 755.013754508
4 991.913730685
5 1217.65123453
6 1429.30662857
7 1623.49086908
8 1796.29664339
9 1943.33535386
10 2059.65197853
11 2139.73315443
12 2177.50066607
13 2166.19936261
14 2098.40421156
15 1966.05005343
16 1760.38440353
17 1471.8590438
18 1090.06689677
19 603.613648569
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)`
${}_{n\vert}\ddot{a}_x$	d-year deferred whole life annuity-due	`annuity(nt,x,n,0,-d)`
${}_{n\vert}a_x$	d-year deferred whole life annuity	`annuity(nt,x,n,1,-d)`
${}_{n\vert}\ddot{a}^{(m)}_{x:\overline{n}\rceil}$	d-year deferred n-year temporal annuity-due m-monthly payments	`annuity(nt,x,n,0,m,-d)`
${}_{n\vert}\ddot{a}_x$	d-year deferred whole life annuity-due	`annuity(nt,x,'w',0,-d)`
${}_{n\vert}a_x$	d-year deferred whole life annuity	`annuity(nt,x,'w',1,-d)`
	Increasing annuities (a sample of them)	
$(Ia)_x$	arithmetically increasing whole-life annuity	`annuity(nt,x,'w',1,['a',c])`
${}^{q}(I\ddot{a})_{x:\overline{n}\rceil}$	arithmetically increasing n-year temporal annuity-due	`annuity(nt,x,n,0,['a',c])`
${}^{q}\ddot{a}_x$	geometrically increasing whole-life annuity-due	`annuity(nt,x,'w',0,['g',c])`
${}_{n\vert}a_x$	d-year deferred geometrically increasing whole-life annuity-due	`annuity(nt,x,'w',0,['g',c],-d)`
${}^{q}_{n\vert}\ddot{a}^{(m)}_{x:\overline{n}\rceil}$	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
$A_{x:\overline{n}\vert}^{\ \ 1}$	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:

```python
#!/usr/bin/python
from pyliferisk.lifecontingencies 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:

```
58.8146911381
```

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 Cactus) to Python, or for simple calculations (where your model shouldn't be subject to future changes). If your goal is reduce the timing, they are a lot of other aspect where you can save time.

```python
#!/usr/bin/python
from pyliferisk.lifecontingencies import Actuarial
from pyliferisk.mortalitytables import SPAININE2004

tariff=Actuarial(nt=SPAININE2004,i=0.02)

print(tariff.Dx[50])
print(tariff.Nx[50])
print(tariff.Cx[50])
print(tariff.Mx[50])
```

Returns:

```
35633.8739667
844266.356048
109.835333846
19269.4834488
```

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).

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 import tables of an external source, i.e. txt or csv files. In that case, you can follow the instructions from section 4 Reading Data.

To use this tables is necessary import all of them (`import *`) or which you will use. Example:

```
from mortalitytables import GKM95 , UK43
```

Notes:

- The probability is qx * 1000.
- The first item indicate 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.
- 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).

4 Reading data

4.1 Data in list

First of all, it's necessary to introduce the list in Python. Lists are mutable sequences, typically used to store collections of homogeneous items. Lists may be constructed in several ways, mainly:

- Using a pair of square brackets to denote the empty list: []
- Using square brackets, separating items with commas: [a, b, c]
- Using the type constructor: list()

```
squares = [0, 1, 4, 16, 25]
squares[5]
```

```
25
```

Sometimes, it's necessary create a range of data (ages, months, years...). For these cases exists the **ranges**. The range type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in for loops.

Syntax: `class range(start, stop [, step])`

```
list(range(10))
list(range(0, 30, 5))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 5, 10, 15, 20, 25]
```

Once create a list, is possible extended, append or remove items:

`list.append(x)`	Add an item to the end of the list
`list.extend(L)`	Extend the list by appending all the items in the given list.
`list.remove(x)`	Remove the first item from the list whose value is x.

A lot of functions uses lists as input (similar than ranges in Excel): `sum([list])` or `len([list])`.

Example:

```
#!/usr/bin/python
from pyliferisk.lifecontingencies import *
from pyliferisk.mortalitytables import *

mt = MortalityTable(nt=USLIFE2002F)

list = []
for age in range(50,65):
        list.append(qx(mt,age))

print(list)
print(list[3])
print(sum(list))
print(len(list))
```

```
[3.194, 3.522, 3.634, 4.142, 4.434, 5.1, 5.006, 5.886, 6.441, 7.266, 7.576, 8.476, 9.201,
    10.101, 11.149]
4.142
95.128
15
```

More info at: *`https://docs.python.org/2/tutorial/datastructures.html`*

4.2 Open basic datafiles with standard library

CSV (comma-separated values) format is the most common import and export format for spreadsheets and databases. It's not necessary to install any library to read and write text files, you only need to import `csv` and the instruction `open()` and the path of the file.

The `csv` library implements classes to read and write tabular data in CSV format. The csv module's `reader` and `writer` objects read and write sequences. Programmers can also read and write data in dictionary form using the `DictReader` and `DictWriter` classes.

`csv.reader(csvfile , [dialect='excel', args])`:
Return a reader object which will iterate over lines in the given csvfile.

`csv.writer(csvfile , [dialect='excel', args])`:
Return a writer object responsible for converting the user's data into delimited strings.

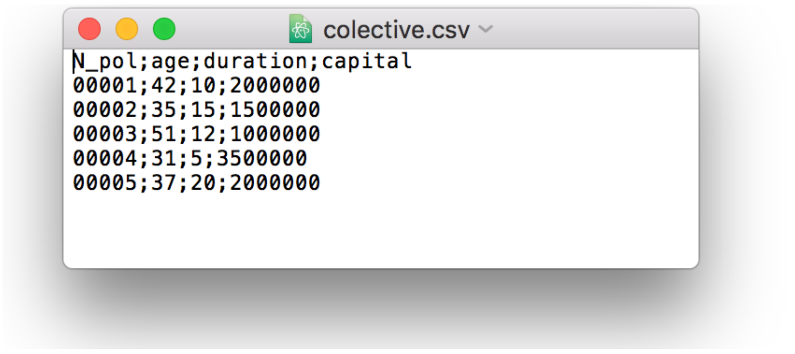
`csv.DictReader(csvfile, [fieldnames=None, restkey=None, restval=None, dialect='excel'])`.
In case of fieldnames parameter in the columns, create an object which operates like a regular reader but maps the information read into a dict whose keys are given by the optional fieldnames parameter.

`class csv.DictWriter(csvfile, [fieldnames, restval='', extrasaction='raise', dialect='excel'])`.
Create an object which operates like a regular writer but maps dictionaries onto output rows.

The mode argument is optional; 'r' will be assumed if it's omitted. The modes can be:

- `'r'` when the file will only be read (by default).
- `'w'` for only writing (an existing file with the same name will be erased).
- `'a'` opens the file for appending; any data written to the file is automatically added to the end.
- `'r+'` opens the file for both reading and writing.

Example:



```
N_pol;age;duration;capital
00001;42;10;2000000
00002;35;15;1500000
00003;51;12;1000000
00004;31;5;3500000
00005;37;20;2000000
```

```python
import csv
with open('colective.csv') as csvfile:
        reader = csv.DictReader(csvfile,delimiter=';')
        for row in reader:
                print(row['age'], row['duration'])
```

```
42 10
35 15
51 12
31 5
37 20
```

More info at: *`https://docs.python.org/2/library/csv.html`*

4.3 MS Excel: pandas and openpyxl libraries

It's highly recommend use csv files (comma-separated values) as your default format and use the `csv` library. Nevertheless, MS Excel (xlsx format) could be an essential need for a lot of actuaries. In this case, you can use `openpyxl` or `pandas` libraries for Excel interaction.

`openpyxl` library allows reading and writing Excel 2010 documents (`https://openpyxl.readthedocs.io`). In the example 4 (in section 6) you can find how to work this library. Anyway, other very best option for working with MS Excel, is `pandas` library. Please, see section 5.3 about `pandas`.

4.4 Database connectors

Python is perfect for risk analysis in big data, since is not limited by database size and is able to access libraries for working with any database is very easy (as SQL, DB2, Oracle, Hadoop, Apache Beam, MongoDB or CDH).

4.4.1 Connecting to MySQL using connector

The following example shows how to connect to the MySQL server:

```
#!/usr/bin/python
import mysql.connector
cnx = mysql.connector.connect(user='scott', password='tiger',
                              host='127.0.0.1',
                              database='employees')
cnx.close()
```

4.4.2 Connecting to Oracle Database 11g using cx_Oracle

While csv files or mysql databases are included in the standard library of Python, other database connectors must be done using external libraries. Each of database has its own library. The following example shows an Oracle connection, which has been obtained from the Oracle oficial site. [1]

```
#!/usr/bin/python
import cx_Oracle
con = cx_Oracle.connect('pythonhol/welcome@127.0.0.1/orcl')
cur = con.cursor()
cur.execute('select * from departments order by department_id')
for result in cur:
    print result
cur.close()
con.close()
```

```
(10, 'Administration', 200, 1700)
(20, 'Marketing', 201, 1800)
(30, 'Purchasing', 114, 1700)
(40, 'Human Resources', 203, 2400)
...
```

[1] `http://www.oracle.com/technetwork/articles/dsl/python-091105.html`

5 Other libraries

The pyliferisk library should to cover all the biometric aspects. Nevertheless, the actuarial calculation requires other addicional functions in order to able to modeling the rest of non-biometric aspects such as insured behavior, economic or operational assumptions (randoms distribution, Economic Scenario Generator, interpolation, etc.)

Despite Python follows the philosophy of "batteries included", having a rich and versatile standard library which is immediately available, without making the user download separate packages, it's highly recommended to use other mathematical libraries (as NumPy) since they are better for this purposes.

The following libraries provides the best tools to help to discount future cashflows, to build assumptions, to correlate risk, Interpolate data, etc.

5.1 NumPy and SciPy

Numpy and SciPy are the must-have libraries for actuarial calculations.

`NumPy` (Numeric Python) is a library for the Python, adding support for large, multi-dimensional arrays and matrices (numerical processing, incluiding array indexing, math operations, etc) along with a large collection of high-level mathematical functions to operate on these arrays. `http://www.numpy.org`

`SciPy` (Scientific Python) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. Advanced mathematical functions such as optimization, interpolation, integration, statistics and other scientific purposes.`https://www.scipy.org`

Below, it included a list of the necessary functions to be able to perform better in any cashflow projection or temporal series:

5.1.1 Statistics functions

A list of the most known distributions:

`beta(a, b [, size])`	Draw samples from a Beta distribution.
`binomial(n, p [, size])`	Draw samples from a binomial distribution.
`chisquare(df[, size])`	Draw samples from a chi-square distribution.
`f(dfnum, dfden[, size])`	Draw samples from an F distribution.
`gamma(shape [, scale, size])`	Draw samples from a Gamma distribution.
`lognormal([mean, sigma, size])`	Draw samples from a log-normal distribution.

More info at: *`https://docs.scipy.org/doc/numpy/reference/routines.statistics.html`*

5.1.2 Random functions

Stochastic modelling is an actuarial technique increasingly used in several fields (Stochastic pricing, quantile reserving, investments and asset liability matching) due to the ease of computer calculation, which allows simulate future lifetimes and future rates of interest using a given set of data.

`rand([size])`	Random values in a given shape.
`randn([size])`	Return a sample (or samples) from the "standard normal" distribution.
`randint(low[, high, size, dtype])`	Return random integers from low (inclusive) to high (exclusive).
`random([size])`	Return random floats in the half-open interval [0.0, 1.0).
`seed([seed])`	Seed the generator.

More info at: *`https://docs.scipy.org/doc/numpy/reference/routines.random.html`*

5.1.3 Financial: Discounting

Discounting cashflows is the core actuarial skill to calculate present values of life payment contingent.

- **np.npv:** Calculates the net present value of a series of payments using a single discount rate. It's the equivalent to NPV in MS Excel.

  Syntax: `numpy.npv(rate, payments)`

  Example:

```
#!/usr/bin/python
import numpy as np
payments = [100, 100, 100, 100, 100, 100, 100, 100, 100, 100]
net_present_value = np.npv(0.02,payments)
print(net_present_value)
```

```
916.223670637
```

- **np.pv:** Calculates the net present value of a series of payments using a single discount rate. It's the equivalent to NPV in MS Excel.

  Syntax: `numpy.npv(rate, n° period, payment, when='end')`
  When payments are due ('begin' (1) or 'end' (0))

  Example:

```
#!/usr/bin/python
import numpy as np
present_value = np.pv(0.02,10,100,when=1)
print(present_value)
```

```
-916.223670637
```

- **np.dot:** Multiplies components of two arrays, and returns the sum of those products. The array must have the same dimensions. It's the equivalent to SUMPRODUCT in MS Excel.

  Syntax: `dot(a, b)`

  Example:

```
#!/usr/bin/python
import numpy as np
payments = [100, 100, 100, 100, 100, 100, 100, 100, 100, 100]
discount = [1, 0.980392, 0.961168, 0.942322, 0.923845, 0.90573,
    0.887971, 0.87056, 0.85349, 0.836755]
sumproduct = np.dot(payments,discount)
print(sumproduct)
```

```
916.2233
```

Please, see example 1 to check differences between discounting with `np.dot` and `np.pv` (mainly if first year must or not to be discounted).

5.1.4 Interpolations

`np.interp1d(x, y, kind='linear')` Interpolate a 1-D function. *(see Example 6.)*

5.2 Date and Datetime library

Although it's possible operate with dates without any specific libraries, it's recommendable to parser with a defined date format due to the different computer regional date settings. For that, the use of the `time` or `datetime` libraries are the best options (both included in the Python Standard Library).

Examples:

1) Return a datetime corresponding to date_string, parsed according to a defined format:

```
from time import strptime
birth_date = strptime('19-03-1979','%d-%m-%Y')
```

2) Return the age using `datetime` library:

```
from datetime import date
from numpy import rint
birth_date = date(1979, 3, 19)
start_date = date(2017, 1, 1)

age = rint(int((start_date - birth_date).days)/365.25)
print age
```

NOTE: rint() rounds to the nearest integer

```
38.0
```

Please, check the library documentation for more information:
`https://docs.python.org/2/library/datetime.html`

`dateutil` library is an advanced extension to the standard Python datetime module, useful to do:

- Generic parsing of dates in almost any string format,
- Computing of relative deltas (next month, next year, next monday, last week of month, etc);
- Computing of relative deltas between two given date and/or datetime objects;
- More info at: `https://dateutil.readthedocs.io`

5.3 Pandas

pandas is also a must-have for actuaries, especially for those with experience in R, since its DataFrame is very much like R-philosophy. 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

5.3.1 Series

pandas.Series is a one-dimensional labeled array. Holding any data type (integers, strings, floating point numbers, Python objects, etc.) The basic method to create a Series is:
s = pandas.Series(data *[, index=index]*)

```
import pandas as pd
s = pd.Series([0,1,4,16,25])
s
```

```
0     0
1     1
2     4
3    16
4    25
```

.values: Returns Series as ndarray

```
s.values
```

```
array([ 0,  1,  4, 16, 25])
```

Series works just like a list.

```
s[2]
```

```
4
```

Vectorized operations and label alignment with Series:

```
import pandas as pd
exit_age = pd.Series([65, 70, 60, 75],
                      index =['group_1','group_2','group_3','group_4'])
exit_age['group_1']
```

```
65
```

```
exit_age[exit_age > 65]
```

```
group_2    70
group_4    75
```

5.3.2 DataFrame

DataFrame is a two-dimensional labeled data structure with labeled axes (rows and columns) of potentially different types. Arithmetic operations align on both row and column labels. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. DataFrame accepts many different kinds of input:

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- structured or record ndarray
- a series
- another DataFrame

Along with the data, you can optionally pass index (row labels) and columns (column labels) arguments.

`df = pandas.DataFrame(data` *`[, index, columns, dtype, copy]`*`)`

```
import pandas as pd
age = range(35,41)
lx = pd.DataFrame([100000, 99737.15, 99455.91, 99154.72, 98831.91,
    98485.68], index=age, columns=['males'])
lx
```

```
        males
35  100000.00
36   99737.15
37   99455.91
38   99154.72
39   98831.91
40   98485.68
```

Indexing The basics of indexing are as follows:

Operation	Syntax	Result
Select column	`df[col]`	Series
Select row by label	`df.loc[label]`	Series
Select row by integer location	`df.iloc[loc]`	Series
Slice rows	`df[5:10]`	DataFrame
Select row and column	`df[col][i]`	item

```
lx['males'][37]
```

```
99737.15
```

Attributes:

`DataFrame.ndim`	Number of axes/array dimensions
`DataFrame.values`	Numpy representation of NDFrame
`DataFrame.size`	number of elements in the NDFrame

Iteration:

`DataFrame.describe()`	Analyzes numeric and object series as well as DataFrame column, such as percentiles, max, min, mean, etc..
`DataFrame.head([n])`	Returns first (or last) n rows (5 by default)
`DataFrame.T`	Transposing your data
`DataFrame.as_matrix([columns])`	Convert the frame to its Numpy-array representation.
`DataFrame.insert(loc, column, value[])`	Insert column into DataFrame at specified location.
`DataFrame.assign`	Assign new columns to a DataFrame, returning a new object (a copy) with all the original columns
`DataFrame.append`	Append rows of other to the end of this frame, returning a new object.

Add the values of a Series as a new column of a DataFrame:

```
lx['female'] = pd.Series([100000, 99763.44, 99489.66, 99196.89, 98880.33,
      98537.61], index=age)
```

```
        males    females
35  100000.00  100000.00
36   99737.15   99737.15
37   99455.91   99455.91
38   99154.72   99154.72
39   98831.91   98831.91
40   98485.68   98485.68
```

More info: ***https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html***

5.3.3 Time Series

`pandas` is perfect for working with date series, useful for discounting cashflows and financial data analysis.

```
import pandas as pd

df = pd.DataFrame(pd.date_range('1/1/2016', periods=5*12,
                                  freq='M'), columns=['month'])
df.head()
```

```
       month
0 2016-01-31
1 2016-02-29
2 2016-03-31
3 2016-04-30
4 2016-05-31
```

More info at: ***https://pandas.pydata.org/pandas-docs/stable/timeseries.html***

5.3.4 Read and write files

One of the best utility of pandas is to be able to work easily with any type of external files (csv, hdf, sql, SAS, excel, json, html, stata, etc..). See all formats and instructions at: `https://pandas.pydata.org/pandas-docs/stable/api.html#input-output`

```
expenses = pd.read_csv('expenses.csv')
cashflow = pd.read_excel('cashflows.xlsx')
```

Writing to an excel file (see example 8).

```
df.to_excel('report.xlsx', sheet_name='Sheet1')
```

More info: ***https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_excel.html***

5.3.5 A perfect match: NumPy and pandas

NumPy functions may be applied to any Series or DataFrame pandas. In the following (nonsense) example shows how to obtain the net present value of the normal random 3x3 matrix diagonal.

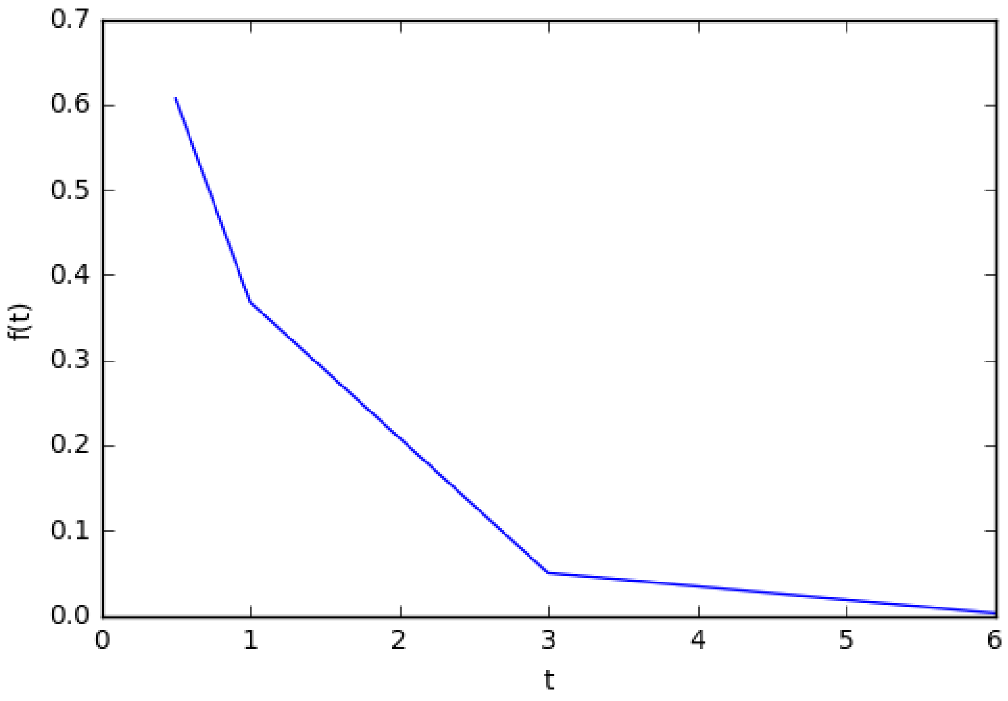
```
import pandas as pd
import numpy as np
np.random.seed([99])
df = pd.DataFrame(np.random.randn(3,3)*100)
np.npv(0.05,pd.Series(np.diag(df)))
```

```
-140.5085
```

5.4 Plotting library: Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code.
`http://matplotlib.org`

```python
import matplotlib.pyplot as plt
t = [0.5, 1, 3, 6]
f = [0.6065, 0.3679, 0.0498, 0.0025]
plt.plot(t,f)
plt.xlabel('t')
plt.ylabel('f(t)')
```



5.5 Other useful libraries

Other libraries to known how to increase the funcionalities (such as import results in MS Excel, ESG, C++ integration, etc...)

- **statsmodels**:
  Statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models (Linear Regression, GLM, ANOVA, etc.), as well as for conducting statistical tests, and statistical data exploration. An extensive list of result statistics are available for each estimator.
  `http://www.statsmodels.org/stable/`
- **rpy2**:
  RPy2 allows you to call implementations written in R within a Python process. There are similar libraries for other languages like C, C ++ or Fortran. R also has a package to execute code written in python (rPython).
  `https://rpy2.readthedocs.io`

```
#!/usr/bin/python
import rpy2.robjects as ro
print(ro.r('pi'))
```

```
3.141593
```

- **ipython**:
  This package allows the functionality of displaying and executing code in a local web environment like a notebook or desktop window. IPython is part of Jupyter, which includes other languages like R, Julia and Scala. Especially useful for academic environments or for testing without installation:
  `https://try.jupyter.org`
  `https://ipython.org`
- **python-xbrl**:
  Library to generate XBRL (metalanguage similar to xml) documents, standard used by regulators and governmental agencies for the reporting of financial information.
  Arelle (`http://arelle.org/`) is an open-source platform that provides a python API used for financial information validations, with support for the DPM database and XBLR used by EIOPA.
  `https://github.com/greedo/python-xbrl`
- **dora** This library contains the main functions for proper data cleaning, in case of errors or omissions in data.
  `https://github.com/NathanEpstein/Dora`
- **cx_Freeze**:
  Python does not require to be compiled to run, but code can be packaged to be distributed or closed code for future modifications. In this case, this library allows you to make executables compatible with Windows.
  `https://github.com/anthony-tuininga/cx_Freeze`
  For similar purposes, see also Tkinter library.

6 Examples

This section includes a list of examples using `pyliferisk` with the libraries seen previously.

6.1 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  #!/usr/bin/python
 2  from pyliferisk.lifecontingencies import *
 3  from pyliferisk.mortalitytables import *
 4  import numpy as np
 5
 6  mt=MortalityTable(nt=SPAIN_PASEM2010M)
 7
 8  age = 67
 9  initial_payment = 80000
10  incr = 0.03                              # increment
11  i = 0.05                                 # interest rate
12
13  discount_factor = []
14  for y in range(0, mt.w-age):
15          discount_factor.append(1/(1+i)**(y+1))
16
17  payments = [initial_payment]
18  for x in range(0, mt.w-age-1):
19          payments.append(payments[x] * (1 + incr) * (1-mt.qx[age+x] /
                1000))
20
21  print('Premium:', np.dot(payments,discount_factor))
```

```
Premium: 967183.635
```

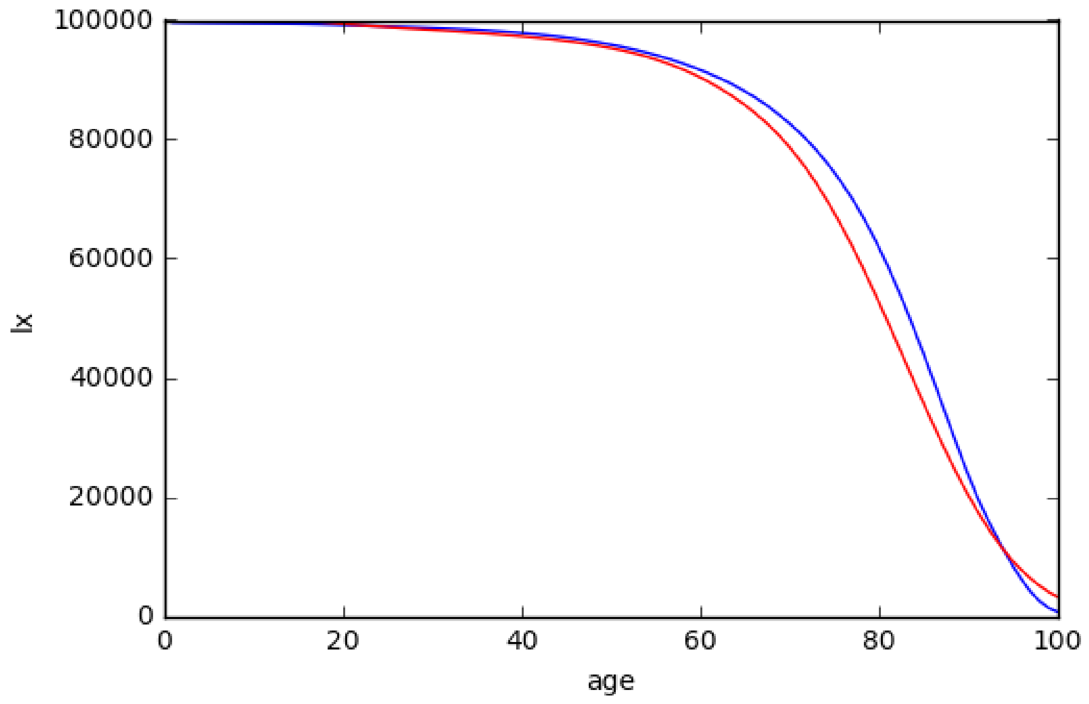
May be different ways of discounting (see previous line 15):

- In case of not discount first year, the exponent should be only `y`.
- In case of discount first year, the exponent should be `y+1`.
- In case of discount half year, the exponent should be `y+0.5`.

6.2 Example 2. Drawing graph with matplotlib

Plotting a surviving graph with pyplot library:

```python
#!/usr/bin/python
import matplotlib.pyplot as plt
import pyliferisk.lifecontingencies as lc
from pyliferisk.mortalitytables import SPAININE2004, GKM95
import numpy as np
tariff=lc.MortalityTable(nt=SPAININE2004)
estimate=lc.MortalityTable(nt=GKM95,perc=75)
x = np.arange(tariff.w)
y = tariff.lx[:tariff.w]
z = estimate.lx[:tariff.w]
plt.plot(x,y, color = 'blue')
plt.plot(x,z, color = 'red')
plt.ylabel('lx')
plt.xlabel('age')
```



6.3 Example 3. Obtain data from plain text file

Term Life with maturity capital benefit and similar death capital. The datafile used is described in the section 4. Reading Data.

```python
#!/usr/bin/python
from pyliferisk.lifecontingencies import *
from pyliferisk.mortalitytables import *
import csv

mt = Actuarial(nt=GKM80,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 1361233.52   163613.5
00002    35         15  1500000  842665.62    73865.3
00003    51         12  1000000  643890.86   69150.37
00004    31          5  3500000 2878725.12  623332.65
00005    37         20  2000000  941452.67   68273.98
----------------------------------------------------
Total Annual Premium: 998235.8
```

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

6.4 Example 4. Obtain the Risk free rate from MS Excel

1) Obtain the risk free rate from the EIOPA Excel file at 31-12-2016.

```python
#!/usr/bin/python
from openpyxl import load_workbook
filename='~/EIOPA_RFR_20161231_Term_Structures.xlsx'
wb = load_workbook(filename, read_only=True)
ws = wb['RFR_spot_no_VA'] # sheet: RFR curve no VA
wr = 'C11:C30' # range: the fist 20 years

for row in ws[wr]:
    for cell in row:
        print(cell.value)
```

```
-0.00302
-0.00261
-0.00208
-0.00123
-0.00024
0.00092
0.00215
0.00341
0.00461
0.00571
0.00671
0.0076
0.00841
0.00908
0.00958
0.00993
0.01019
0.01046
0.01077
0.01117
```

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

6.5 Example 5. Cashflow Calculation Reserving

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

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

Using lineal interest rate

```python
#!/usr/bin/python
from pyliferisk.lifecontingencies import *
from pyliferisk.mortalitytables import *
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+1))

        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)
            )

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

```
820.796050246
1088.37413953
```

Including risk free rate curve

```python
#!/usr/bin/python
from pyliferisk.lifecontingencies import *
from pyliferisk.mortalitytables import *
import numpy as np
from openpyxl import load_workbook

filename='/Users/panna/Proyectos/Ejemplos/
    EIOPA_RFR_20161231_Term_Structures.xlsx'
wb = load_workbook(filename, read_only=True)
ws = wb['RFR_spot_no_VA'] # sheet: RFR curve no VA
wr = 'C11:C160'
rfr_vector = []
for row in ws[wr]:
    for cell in row:
        rfr_vector.append(cell.value)

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) #fixed at age 25

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+1))
                elif i == 'rfr':
                        discount_factor.append(1/(1+rfr_vector[y])**(y+1)
                            )

        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)
            )

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

```
820.796050246
1088.37413953
544.308400948
```

6.6 Example 6. Interpolations with SciPy and pandas

Actuaries often have to interpolate data to obtain or estimate new data points. Mainly to construct assumptions to be used (such as lapses, surrenders, persistency, etc.) within the range of a discrete set of known data points.

Note: The following examples have been done by John Kitchin[2] for the Kitchin Research Group. This data that describes the value of f at time t with the formula `f(t) = exp(-t)`.

1) Lineal interpolation: Estimate the value of f at t=2.

```
from scipy.interpolate import interp1d
t = [0.5, 1, 3, 6]
f = [0.6065, 0.3679, 0.0498, 0.0025]

g = interp1d(t, f) # default is linear interpolation
print g(2)
print g([2, 3, 4])
```

```
0.20885
[ 0.20885     0.0498      0.03403333]
```

Remember that the function used above is actually f(t) = exp(-t), i.e. exp(-2) = 0.135335283237. So, the linearly interpolated example is not too accurate.

2) Cubic Spline interpolation. Estimate the value of f at t=2.

While linear interpolation uses a linear function for each of intervals, Spline interpolation uses low-degree polynomials in each of data, incurring in a smaller error than linear interpolation since the interpolant is smoother.

```
g2 = interp1d(t, f, 'cubic') #cubic refer to a spline interpolation of
    zeroth
print g2(2)
print g2([2, 3, 4])
```

```
0.108481818182
[ 0.10848182  0.0498      0.08428727]
```

Using pandas:

```
import pandas as pd
s = pd.Series([10, 20, np.nan, 40, 50, np.nan, 30])
s.interpolate()
```

```
0    10.0
1    20.0
2    30.0
3    40.0
4    50.0
5    40.0
6    30.0
```

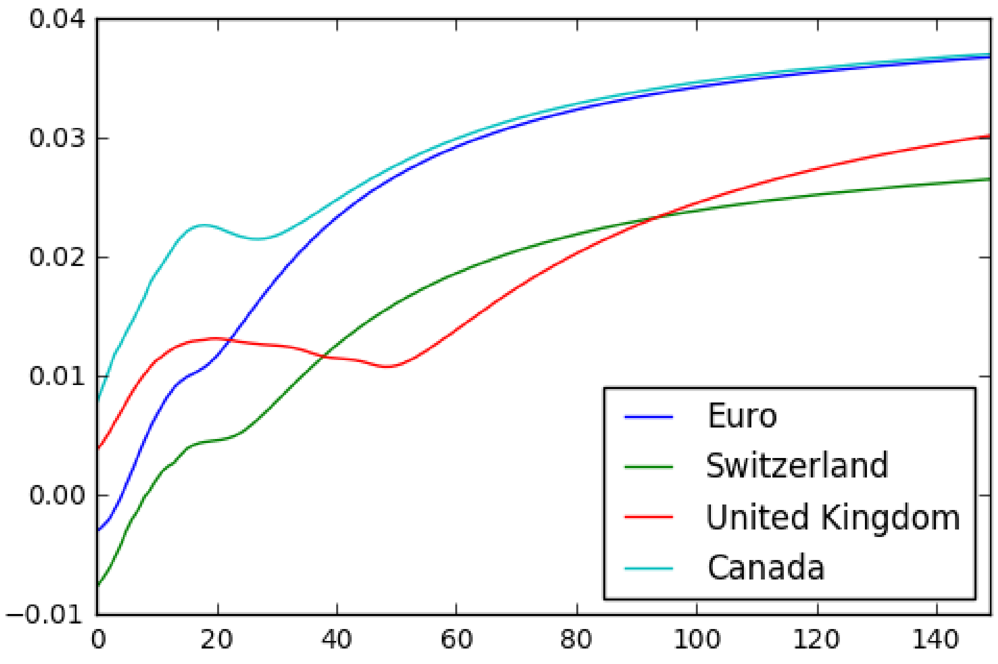
Available interpolate methods: linear (by defaul), time, index, values, nearest, zero, slinear, quadratic, cubic, barycentric, krogh, polynomial, spline, piecewise_polynomial, from_derivatives, pchip, akima.

[2] http://kitchingroup.cheme.cmu.edu/blog/2013/02/02/Better-interpolate-than-never/

6.7 Example 7. Plotting with pandas

Draw the risk-free-rate curves for Euro, Switzerland, United Kingdom and Canada (from EIOPA Excel file at 2016-12-31).

```
import pandas as pd
filename='~/EIOPA_RFR_20161231_Term_Structures.xlsx'
sheetname='RFR_spot_no_VA'
df = pd.read_excel(filename, sheetname=sheetname, skiprows=list(range
    (1,10)))  # skips the 10 first rows
df.drop(['Unnamed: 0'], axis=1, inplace=True)   #delete first column
df.plot(y=['Euro','Switzerland','United Kingdom','Canada'])
```



6.8 Example 8. Cashflows in pandas

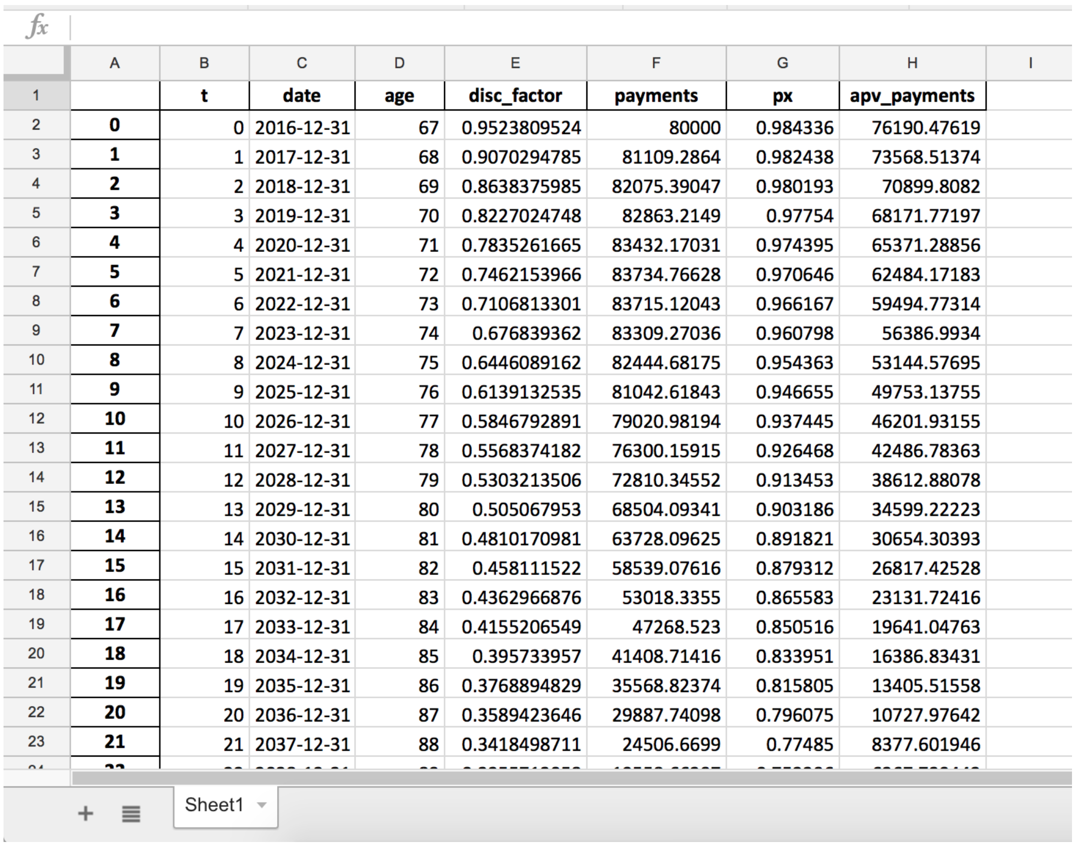
Replicating Example 1 with pandas: Life annuity immediate geometrically increasing for a male 67 years-old, as single premium (PASEM2010, interest 5%).

```python
#!/usr/bin/python
from pyliferisk.lifecontingencies import MortalityTable
from pyliferisk.mortalitytables import SPAIN_PASEM2010M
import pandas as pd
import numpy as np

mt=MortalityTable(nt=SPAIN_PASEM2010M)

age = 67
initial_payment = 80000
incr = 0.03                                # increment
i = 0.05                                   # interest rate

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

df = pd.DataFrame(period, columns=['t'])
df['date'] = pd.DataFrame(pd.date_range('1/1/2016', periods=len(period),
    freq='A'))
df['age'] = pd.Series(list(range(age,mt.w)))
df['disc_factor']= pd.Series(np.zeros(len(period)))

for t in period:
        df['disc_factor'][t] = 1/(1+i)**(t+1)

df['payments']= pd.Series(np.zeros(len(period)))
df.loc[0,'payments'] = initial_payment

for t in period:
    df['payments'][t+1] = df['payments'][t] * (1 - mt.qx[age+t] / 1000) *
      (1 + incr)

df['px']= pd.Series(np.zeros(len(period)))
for t in period:
    df['px'][t] =  1 - mt.qx[age+t] / 1000

df['apv_payments'] = df['payments'] * df['disc_factor']
premium = sum(df['apv_payments'])
premium
```

```
967183.635
```

Export the DataFrame to a MS Excel file:

```
df.to_excel(pd.ExcelWriter('output.xlsx'),'Sheet1')
writer.save()
```



	t	date	age	disc_factor	payments	px	apv_payments
0	0	2016-12-31	67	0.9523809524	80000	0.984336	76190.47619
1	1	2017-12-31	68	0.9070294785	81109.2864	0.982438	73568.51374
2	2	2018-12-31	69	0.8638375985	82075.39047	0.980193	70899.8082
3	3	2019-12-31	70	0.8227024748	82863.2149	0.97754	68171.77197
4	4	2020-12-31	71	0.7835261665	83432.17031	0.974395	65371.28856
5	5	2021-12-31	72	0.7462153966	83734.76628	0.970646	62484.17183
6	6	2022-12-31	73	0.7106813301	83715.12043	0.966167	59494.77314
7	7	2023-12-31	74	0.676839362	83309.27036	0.960798	56386.9934
8	8	2024-12-31	75	0.6446089162	82444.68175	0.954363	53144.57695
9	9	2025-12-31	76	0.6139132535	81042.61843	0.946655	49753.13755
10	10	2026-12-31	77	0.5846792891	79020.98194	0.937445	46201.93155
11	11	2027-12-31	78	0.5568374182	76300.15915	0.926468	42486.78363
12	12	2028-12-31	79	0.5303213506	72810.34552	0.913453	38612.88078
13	13	2029-12-31	80	0.505067953	68504.09341	0.903186	34599.22223
14	14	2030-12-31	81	0.4810170981	63728.09625	0.891821	30654.30393
15	15	2031-12-31	82	0.458111522	58539.07616	0.879312	26817.42528
16	16	2032-12-31	83	0.4362966876	53018.3355	0.865583	23131.72416
17	17	2033-12-31	84	0.4155206549	47268.523	0.850516	19641.04763
18	18	2034-12-31	85	0.395733957	41408.71416	0.833951	16386.83431
19	19	2035-12-31	86	0.3768894829	35568.82374	0.815805	13405.51558
20	20	2036-12-31	87	0.3589423646	29887.74098	0.796075	10727.97642
21	21	2037-12-31	88	0.3418498711	24506.6699	0.77485	8377.601946

7 Installation & IDE

7.1 Python installation

Python can be used on 21 different operating systems and environments. There are even versions that run on .NET and the Java virtual machine: `https://www.python.org/downloads/`

The Python implementation is under an open source license that makes it freely usable and distributable, even for commercial use. The Python license (`http://www.python.org/psf/license/`) is administered by the Python Software Foundation.

Python 2.7 or 3.0? Python 3.0 was released in 2008. The final 2.x version 2.7 release came out in mid-2010, with a statement of extended support for this end-of-life release. For beginners, the most recommendable is Python 3. Pyliferisk was written for Python, anyway should not be any issue to be used under Python 3.0. More info: `https://wiki.python.org/moin/Python2orPython3`

7.2 Library installation

Once pyhon is running, just install this library with `pip install pyliferisk` or download the source code at github (git clone): `https://github.com/franciscogarate/pyliferisk`

```
> pip install pyliferisk
```

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

```python
import pyliferisk.lifecontingencies as lc
```

or, if only like to use specific functions:

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

Update library: Run this command from terminal:

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

Desktop Software IDE

I highly recommended **Rodeo** (`http://rodeo.yhat.com/`)for desktop or **Sublime Text 2**: (`http://www.sublimetext.com/`): minimal and non-necessary settings. Ideal for testing. Sublime Text uses a custom UI toolkit, optimized for speed and beauty, and may be downloaded and evaluated for free.

Eclipse (or Aptana Studio 3 -based on Eclipse-) is an integrated development environment (IDE) recommended especially for python projects with a lot of files. Both are open-source and multi-platform. Please check the respective tutorials for installation.
For professional use, **Rodeo** or **Canopy** platforms: both are comprehensive Python analysis environments.

Additionally, **Anaconda** (`https://www.continuum.io/downloads`) is a distribution of Python for large-scale data processing, predictive analytics, and scientific computing:

- It includes a collection of about 200 open source packages and includes Intel MKL optimizations throughout the scientific Python stack.
- These packages include NumPy, Pandas, SciPy, Matplotlib, and Jupyter.
- Additional packages are available through contributed channels or through installation with pip.

8 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), 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.

9 Acknowledgements

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

10 To-Do's

- A example using a lx given
- A example with Generation Mortality Table with pandas
- Check full compatibility with Python 3.0