Library lifeactuary

Pedro Corte Real & Gracinda Rita Guerreiro

January 28, 2023

Type: Python Package

Version: 1.2

Authors: Pedro Corte Real & Gracinda R. Guerreiro

Contact: parcr@fct.unl.pt & grg@fct.unl.pt

Repository: https://github.com/parcr/lifeactuary_1.2

Abstract

lifeactuary is a Python library to perform actuarial mathematics on life contingencies and classical financial mathematics computations. Versatile, simple and easy to use. The main functions are implemented using the usual actuarial approach, making it a natural choice for the life actuary.

This document is produced as a descriptive tool on how to use the package and as a user guide for the developed actuarial functions. For each actuarial function, an illustrative example is provided.

The package uses Python version 3.7 or higher.

This package and functions herein are provided as is, without any guarantee regarding the accuracy of calculations. It's distributed using the MIT License and the authors disclaim any liability arising by any losses due to direct or indirect use of this package.

This package is still under development and further useful and interesting functions will be available any time soon.

Version 1.2 includes a new class *CommutationTableFrac*, which computes actuarial tables for non-integer ages and includes some improvements on the previous version. Namely:

- We solve the problem of reading the tables_manual.xlsx and having to deal with the "nan" produced by pandas when reading from excel columns with different number of rows.
- We change the way we deal with the "ω" in *mortality_table.py* so that we are able to deal with fractional ages and produce fractional commutation tables.

Contents

1 Introduction

The *lifeactuary* library for Python aims to provide a wide range of actuarial functions for life contingencies. The names of the functions follow the International Actuarial Notation and are intuitive (qx, p_x, lx, an, axn, nEx, Ax, …) and the common parameters are set as usual (x for actuarial age, n for term of the contract, p for term of payments, …). This aims to provide with an easy to use and "guessable" list of functions.

Using mortality tables, the library provides functions for computing (a) survival probabilities for integer and non-integer ages and terms, (b) life expectancy for both integer and non integer ages and terms, (c) expected present value of life annuities and (d) expected present value of traditional life insurances. All these features allow for the development of simple or more complicated actuarial evaluations and product developments.

This library can be used for academic or professional purposes. The library contains the functions of main traditional products in Life Insurance, allows for an easy computation of actuarial tables but also provides the tools for designing new products, building tariffs, computing reserves.

It incorporates a very generic and simple to use function to compute the Expected Present Value (aka, actuarial expected present value) what is becoming a very relevant tool in the solvency analysis.

A set of examples is presented in the end of this manual, showing some potential uses of this library in life contingencies products and evaluations.

This package is still under development and further useful functions will be available any time soon. In the next release, the package will include functions that allow the computation of: (1) variance of life annuities, (2) variance of classical life insurances and (3) expected present value of annuities for multiple lives.

2 Mortality Tables

The functions developed on this section are related to common actuarial biometric functions, computed upon a mortality table.

2.1 Class MortalityTable

---

class *MortalityTable*
This class instantiates a life table. Data can be provided in the form of the l_x, q_x or p_x. Note that the first value is the first age considered in the table. The life table will be complete, that is, from age 0 to age ω (the last age where $l_x > 0$). It includes the computation of common biometric functions: l_x, p_x, q_x, d_x, e_x for integer ages and for non-integers ages, using methods as Uniform Distribution of Death (udd), Constant Force of Mortality (cfm) and Balducci approximation (bal).

Usage

```
MortalityTable(data_type='q', mt=None, perc=100, last_q=1)
```

Description
Initializes the MortalityTable class so that we can construct a mortality table with the usual fields.

Parameters

data_type	Use 'l' for l_x, 'p' for p_x and 'q' for q_x.
mt	The mortality table, in array format, according to the data_type defined
perc	The percentage of q_x to use, e.g., use 50 for 50%.
last_q	The value for q_ω.

2.2 Importing Mortality Tables

The package includes a wide number of mortality tables and allows for the inclusion of any other mortality tables extracted from SOA, in xml format, or by importing other ones in usual formats, such as xlsx, csv, txt. For instance, in the manual, one of the tables that we will be using is the TV7377 in the xml format supported by the SOA.

2.2.1 Reading from lifeactuary Package

Example

```
from lifeactuary import mortality_table as mt, read_soa_table_xml as rst

# reads TV7377 mortality table from SOA table
soa = rst.SoaTable('soa_tables/' + 'TV7377' + '.xml')

# creates mortality table from qx of SOA table
tv7377 = mt.MortalityTable(data_type='q', mt=soa.table_qx, perc=100, last_q=1)
```

2.2.2 Importing from File

When building a new mortality table to import from a file, please note that the first value of the table corresponds to the first age considered in the table. For instance, if the first value of the table is 20, it means that $l_x = 0$, for $x = 0, \ldots, 19$.

Usage

```
from lifeactuary import mortality_table as mt
import pandas as pd

# reads manually imported mortality table
table_manual_qx = pd.read_excel('soa_tables/' + 'tables_manual' + '.xlsx', sheet_name='qx')
table_manual_lx = pd.read_excel('soa_tables/' + 'tables_manual' + '.xlsx', sheet_name='lx')

# creates mortality table from lx of a xlsx file
grf95 = mt.MortalityTable(data_type='q', mt=list(table_manual_qx['GRF95']), perc=80)
grm95 = mt.MortalityTable(data_type='l', mt=list(table_manual_lx['GRM95']), perc=80)
```

2.3 Demographic Functions

After the mortality table is instantiated, the common demographic functions are available in the package, such as l_x (expected number of subjects alive at age x), d_x (expected number of deaths with age x), q_x (mortality rate at age x), e_x (complete life expectancy at age x), ω (terminal age of the mortality table):

2.3.1 lx[x]

Actuarial Notation	l_x
Usage	mt.lx[x]
Args	x: age as an integer number
Example	tv7377.lx[50]
Result	94055.99997478718

2.3.2 dx[x]

Actuarial Notation	d_x
Usage	mt.dx[x]
Args	x: age as an integer number
Example	tv7377.dx[50]
Result	353.99999675630613

2.3.3 qx[x]

Actuarial Notation	q_x
Usage	mt.qx[x]
Args	x: age as an integer number
Example	tv7377.qx[50]
Result	0.0037637152

2.3.4 px[x]

Actuarial Notation	p_x
Usage	mt.qx[x]
Args	x: age as an integer number
Example	tv7377.px[50]
Result	0.9962362848

2.3.5 ex[x]

Actuarial Notation	e_x
Usage	mt.ex[x]
Args	x: age as an integer number
Example	tv7377.ex[50]
Result	30.07981415164423

2.3.6 w

Actuarial Notation	ω
Usage	mt.w
Example	tv7377.w
Result	106

Other Examples

```
## Consulting information from an object

# Outputs the information necessary to clone the object
tv7377

# consults the lx of TV7377
tv7377.lx

# Consults the ex of GRF95
grf95.ex

# extracts all methods from the object grm95
grm95.__dict__
```

2.4 Survival Probabilities Functions

The package also allows for the direct computation of survival probabilities for an aged x individual. Focusing on the common actuarial probabilities, some functions are available for the computations of the following probabilities: ${}_nq_x$, ${}_np_x$, ${}_{t|n}q_x$ for integer and non-integer ages and periods. In fact, in this library, the non-integer ages and periods are just a particular case when using any method.

2.4.1 nqx

Actuarial Notation: ${}_nq_x$

Usage

```
nqx(x, n=1, method='udd')
```

Parameters

x	age at the beginning
n	period
method	For non-integer ages and periods, use 'udd' for *Uniform Distribution of Death*, 'cfm' for *Constant Force of Mortality* and 'bal' for *Balducci approximation*
return	probability of (x) dying before age $x + n$

Examples

```
# probability that (50) dies before age 52.
tv7377.nqx(50, 2) # 0.0078038614928698236

# probability that an aged 50.5 individual dies before age 53.
tv7377.nqx(50.5, 2.5 ,method='udd') # 0.010321797187509807
tv7377.nqx(50.5, 2.5 ,method='cfm') # 0.010320038151286903
tv7377.nqx(50.5, 2.5 ,method='bal') # 0.010318279111937612
```

2.4.2 npx

Actuarial Notation: ${}_np_x$

Usage

```
npx(x, n=1, method='udd')
```

Parameters

x	age at the beginning
n	period
method	For non-integer ages and periods, use 'udd' for *Uniform Distribution of Death*, 'cfm' for *Constant Force of Mortality* and 'bal' for *Balducci approximation*
return	probability of (x) surviving beyond age $x + n$

Examples

```
# probability that (80) reaches age 82.
tv7377.npx(80, 2) # 0.8563257446904969

# probability that an aged 80.5 individual reaches age 85.
tv7377.npx(80.5, 4.5 ,method='udd') # 0.3512032870461814
tv7377.npx(80.5, 4.5 ,method='cfm') # 0.3507713780990377
tv7377.npx(80.5, 4.5 ,method='bal') # 0.3503391816267956 7
```

2.4.3 t_nqx

Actuarial Notation: ${}_{t|n}p_x$

Usage

```
t_nqx(x, t=1, n=1, method='udd')
```

Parameters

x	age at the beginning
t	deferment period
n	period
method	For non-integer ages and periods, use 'udd' for *Uniform Distribution of Death*, 'cfm' for *Constant Force of Mortality* and 'bal' for *Balducci approximation*
return	probability of (x) surviving beyond age $x + t$ and die before age $x + t + n$

Examples

```
# probability that (80) reaches age 82.
tv7377.npx(80, 2) # 0.8563257446904969

# probability that an aged 80.5 individual reaches age 85.
tv7377.npx(80.5, 4.5 , 'udd') # 0.3512032870461814
tv7377.npx(80.5, 4.5 , 'cfm') # 0.3507713780990377
tv7377.npx(80.5, 4.5 , 'bal') # 0.35033918162679567
```

2.5 Life Expectancy Function

The library allows for the computation of Complete Life Expectancy for integer and non-integer ages and periods.

Usage

```
exn(x, n, method='udd')
```

Parameters

x	age at the beginning
n	period
method	For non-integer ages, use 'udd' for *Uniform Distribution of Death*, 'cfm' for *Constant Force of Mortality* and 'bal' for *Balducci approximation*
return	life expectancy for (x) over the next n years

Examples

```
# complete life expectancy for (60) over the next 10 years
tv7377.exn(60, 10)       # 9.498277332706456
tv7377.exn(60, 10,'cfm') # 9.498146560076156
tv7377.exn(60, 10,'bal') # 9.498015788406414

# complete life expectancy for (60.1) over the next 10.2 years
tv7377.exn(60.1, 10.2)       # 9.673511678284852
tv7377.exn(60.1, 10.2,'cfm') # 9.67338786347054
tv7377.exn(60.1, 10.2,'bal') # 9.673264041773342
```

2.6 Actuarial Tables

From a given Mortality Table, the library allows for the construction of an Actuarial Table (or Commutation Table) providing and allowing to access the values of the common commutation symbols.
This functions are useful for academic purposes, or to implement "old" life contingency products where commutation symbols were commonly used.
At the moment, actuarial evaluation should use cashflow projections, for which interest rate curves should be considered instead of a fixed one, as in the case of actuarial tables.
Despite this, this library allows for the computation of traditional methods in life insurance.

2.6.1 Class CommutationTable

---

class *CommutationTable*
This class instantiates, for a specific mortality table and interest rate, all the usual commutation functions: D_x, N_x, S_x, C_x ,M_x and R_x.

Usage

```
CommutationFunctions(i=None, g=0, data_type='q', mt=None, perc=100, app_cont=False)
```

Description
Initializes the CommutationTable class so that we can construct an actuarial table with the usual fields.

Parameters

i	Interest Rate, in percentage. For instance, use 5 for 5%.
g	Rate of growing (in percentage), for capitals evolving geometrically. $g > 0$ for increasing capitals and $g < 0$ for decreasing capitals
data_type	Use 'l' for l_x, 'p' for p_x and 'q' for q_x.
mt	The mortality table, in array format, according to the data_type defined
perc	The percentage of q_x to use, e.g., use 50 for 50%.
app_cont	Use True for continuous approach (deaths occur, in average, in the middle of the year) or False for considering death payments are considered in the end of the year.

Examples

```
from lifeactuary import commutation_table as ct, read_soa_table_xml as rst

# reads SOA table
soa = rst.SoaTable('soa_tables/' + 'TV7377' + '.xml')

# creates an actuarial table from qx of SOA table
tv7377_ct = ct.CommutationFunctions(i=2, g=0, data_type='q', mt=soa.table_qx, perc=100,
    app_cont=False)

# creates an actuarial table from the lx of an Excel file, with death payments to be processed
     in the moment of death
grm95_ct = ct.CommutationFunctions(i=1.5, g=0, data_type='l', mt=list(table_manual_lx['GRM95'
    ]), perc=100, app_cont=False)
```

With the construction of the commutation table (ct), the class computes methods that are useful when computing actuarial evaluations, which are described in subsections 2.6.4 to 2.6.10.

2.6.2 Class CommutationTableFrac

---

class *CommutationTableFrac*
This class instantiates, for a specific mortality table and interest rate, all the usual commutation functions: D_x, N_x, S_x, C_x, M_x and R_x, for ages $x + \frac{1}{frac} \in \left\{0, \frac{1}{frac}, \ldots, \omega + \frac{frac-1}{frac}\right\}$.

Usage

```
CommutationFunctions(i=None, g=0, data_type='q', mt=None, perc=100, frac=2, method='udd')
```

Description
Initializes the CommutationTableFrac class so that we can construct an actuarial table with the usual fields, for all indented fractional ages, considering that payments are made in the end of the fractional times.

Parameters

i	Interest Rate, in percentage. For instance, use 5 for 5%.
g	Rate of growing (in percentage), for capitals evolving geometrically. $g > 0$ for increasing capitals and $g < 0$ for decreasing capitals. For instance, use 2 for 2%.
data_type	Use 'l' for l_x, 'p' for p_x and 'q' for q_x.
mt	The mortality table, in array format, according to the data_type defined
perc	The percentage of q_x to use, e.g., use 50 for 50%.
frac	Number of fractional ages for each age x
method	Approximation method for non-integer ages. Use 'udd' for *Uniform Distribution of Death*, 'cfm' for *Constant Force of Mortality* and 'bal' for *Balducci approximation*.

Examples

```
from lifeactuary import mortality_table as mt, commutation_table as ct, commutation_table_frac
    , read_soa_table_xml as rst

# reads SOA table
soa = rst.SoaTable('soa_tables/' + 'TV7377' + '.xml')

# creates an actuarial table from qx of SOA table, for ages x+k*0.5, x=0,...,w, k=0,1.
tv7377_ct_f2 = CommutationFunctionsFrac(i=2, g=0, data_type='q', mt=soa.table_qx, perc=100,
    frac=2, method='udd')

# creates an actuarial table from qx of SOA table, for ages x+k*0.25, x=0,...,w, k=0,1,2,3
tv7377_ct_f4 = CommutationFunctionsFrac(i=2, g=0, data_type='q', mt=soa.table_qx, perc=100,
    frac=4, method='udd')

# creates an actuarial table from qx of SOA table, for ages x+k*1/6, x=0,...,w, k=0,1,...,5
tv7377_ct_f6 = CommutationFunctionsFrac(i=2, g=0, data_type='q', mt=soa.table_qx, perc=100,
    frac=6, method='udd')

```

```
# creates an actuarial table from qx of SOA table, for ages x+k*1/365, x=0,...,w, k=0,1,...,364
tv7377_ct_f365 = CommutationFunctionsFrac(i=2, g=0, data_type='q', mt=soa.table_qx, perc=100, frac=365, method='udd')

# creates an actuarial table from qx of SOA table, for ages x+k*0.5, x=0,...,w, k=0,1 with rate of growing of 1%
tv7377_ct_f2_g1 = CommutationFunctionsFrac(i=2, g=1, data_type='q', mt=soa.table_qx, perc=100, frac=2, method='udd')
```

With the construction of the fractional commutation table, the class computes methods that may be useful when computing actuarial evaluations.

Observation: Naturally, the *CommutationTableFrac* class with *frac*=1 produces the same results as the class *CommutationTable*.

2.6.3 Function age_to_index()

The *CommutationTableFrac* class produces, for each commutation symbol, a vector with $\omega + (frac - 1) \times \omega$ components. To simplify the access of each method to each fractional age, the user should use the following function:

Usage

```
age_to_index(age_int, age_frac)
```

Description: Returns the index of the vector position in a given method of an actuarial table, corresponding to the age_int+age_frac position.

Parameters

age_int	integer part of the age
age_frac	fractional part of the age

Examples

```
tv7377_ct_f2.age_to_index(50, 0.5)   # 101
tv7377_ct_f4.age_to_index(50, 0.75)  # 203
tv7377_ct_f6.age_to_index(50, 5/6)   # 305
```

2.6.4 v

Actuarial Notation	v
Definition	$\frac{1}{1+i}$
Usage	ct.v
Example	tv7377_ct.v
Result	0.9803921568627451

2.6.5 Dx and Dx_frac

Actuarial Notation	D_x
Usage	ct.Dx[x]
Args	x: age as an integer
Example	tv7377_ct.Dx[50]
Result	34944.42647196618

As for the Dx_frac method, the following examples illustrate the use of the method:

Examples

```
x=50.5
index_age=tv7377_ct_f2.age_to_index(int(x), x-int(x))  #101
a=tv7377_ct_f2.Dx_frac[index_age]
print(f'For age {x} with index {index_age}, Dx={a}')
# For age 50.5, with index 101, Dx=34535.02547926754
```

2.6.6 Nx and Nx_frac

Actuarial Notation	N_x
Usage	ct.Nx[x]
Args	x: age as an integer
Example	tv7377_ct.Nx[50]
Result	788151.7176774722

As for the Nx_frac method, the following examples illustrates the use of the method:

Examples

```
x=35+1/6
index_age=tv7377_ct_f6.age_to_index(int(x), x-int(x))  # 211
a=tv7377_ct_f6.Nx_frac[index_age]
print(f'For age {x} with index {index_age}, Nx={a}')
# For age 35.166666666666664, with index 211, Nx=8333822.587495911


## Present value of an unitary whole life annuity due, paid quarterly to an individual aged
    x=65.25
x=65.25
index_age=tv7377_ct_f4.age_to_index(int(x), x-int(x))  # 261
a=tv7377_ct_f4.Nx_frac[index_age]/tv7377_ct_f4.Dx_frac[index_age]
print(f'For age {x} with index {index_age}, ax={a}')
# For age 65.25, with index 261, ax=56.9123257953868


## Present value of an unitary whole life annuity due, paid annually to an individual aged
    x=65.25
print(f'For age {x}, with index {index_age}, ax(4)={a/4}')
# For age 65.25, with index 261, ax(4)=14.2280814488467


```

```python
## Present value of an unitary whole life annuity immediate, paid daily to an individual aged
       x=66+120/365
x=66+120/365
index_age=tv7377_ct_f365.age_to_index(int(x), x-int(x))  # 24210
a=tv7377_ct_f4.Nx_frac[index_age]/tv7377_ct_f4.Dx_frac[index_age]
print(f'For age {x} with index {index_age}, ax={a}')
# For age 66.32876712328768, with index 24210, ax=4572.698998279401


## Present value of an unitary whole life annuity immediate, paid annually to an individual
    aged x=66+120/365
print(f'For age {x}, with index {index_age}, ax(365)={b/365}')
# For age 66.32876712328768, with index 24210, ax(365)=12.527942461039455
```

2.6.7 Sx and Sx_frac

Actuarial Notation	S_x
Usage	ct.Sx[x]
Args	x: age as an integer
Example	tv7377_ct.Sx[50]
Result	12024274.4751688

Examples for using Sx_frac are similar to the previous methods and will be omitted.

In the following methods, the choice between payments made in the “end of the year” or in the “moment of death” must be performed when constructing the commutation table (False or True in the app_cont parameter, respectively). In that sense, the actuarial notation in each of the following methods is given for both scenarios. As for the computation of these commutation symbols for non-integer ages, the reasoning is the same as the previous sub-sections: define a fractional actuarial table and use the methods for non-integer ages, according the defined fractions of the year. In that sense, examples will not the included.

2.6.8 Cx and Cx_frac

Actuarial Notation	C_x or $\overline{C}_x$
Usage	ct.Cx[x]
Args	x: age as an integer number
Example	tv7377_ct.Cx[50]
Result	128.94202849786421

2.6.9 Mx and Mx_frac

Actuarial Notation	M_x or $\overline{M}_x$
Usage	ct.Mx[x]
Args	x: age as an integer number
Example	tv7377_ct.Mx[50]
Result	19490.471223388264

2.6.10 Rx and Rx_frac

Actuarial Notation	R_x or $\overline{R}_x$
Usage	ct.Rx[x]
Args	x: age as an integer number
Example	tv7377_ct.Rx[50]
Result	552381.6299290637

Examples

```python
# Actuarial present value of a unitary due whole life annuity for (50) paid annually
tv7377_ct.Nx[50]/tv7377_ct.Dx[50]    # 22.55443277

# Actuarial present value of a whole life insurance for (50) with 100.000 m.u. capital. Payment is made in the moment of death
tv7377_ct_md = ct.CommutationFunctions(2, 0, 'q', soa.table_qx, 100, True)
100000*tv7377_ct_md.Mx[50]/tv7377_ct_md.Dx[50]     # 56330.616995

# Present value of an unitary whole life annuity due, paid quarterly to an individual aged x=65.25
x = 65.25
index_age = tv7377_ct_f4.age_to_index(int(x), x - int(x))  # 261
a = tv7377_ct_f4.Nx_frac[index_age] / tv7377_ct_f4.Dx_frac[index_age]
print(f'For age {x}, with index {index_age}, ax(4)={a}')
# For age 65.25, with index 261, ax(4)=56.9123257953868

## Actuarial present Value of unitary annuity due, paid semiannually with 10 terms for (35.5)
x = 35.5
n=10/2
index_age = tv7377_ct_f2.age_to_index(int(x), x - int(x))
index_age_end = tv7377_ct_f2.age_to_index(int(x+n), x+n - int(x+n))
b = (tv7377_ct_f2.Nx_frac[index_age] - tv7377_ct_f2.Nx_frac[index_age_end])/tv7377_ct_f2.Dx_frac[index_age]
print(f'For age {x}, with index {index_age}, with semiannual payments until age {x+n}, with index {index_age_end}, ax:n={b}')
# For age 35.5, with index 71, with semiannual payments until age 40.5, with index 81, ax:n=9.542714980644465

## Actuarial present value of a whole life insurance for (50.75) with 100.000 m.u. capital.
x=50.75
index_age=tv7377_ct_f4.age_to_index(int(x), x-int(x))  # 203
100000*tv7377_ct_f4.Mx_frac[index_age]/tv7377_ct_f4.Dx_frac[index_age]
print(f'For age {x}, with index {index_age}, the risk premium is Ax={b}')
# For age 50.75, with index 203, the risk premium is Ax=56909.96956118816

## Actuarial present Value of a whole life annuity paid semiannually to an individual aged x = 35.5. Payments have a growth rate of 1%
x = 35.5
index_age = tv7377_ct_f2_g1.age_to_index(int(x), x - int(x))  # 151
a = tv7377_ct_f2_g1.Nx_frac[index_age] / tv7377_ct_f2_g1.Dx_frac[index_age]
print(f'For age {x}, with index {index_age}, Gax(2)={a}')
# For age 35.5, with index 71, Gax(2)=70.31380781464682
```

3 Export Actuarial Table to Excel

Actuarial Tables may be exported to Excel files, using the standard Python functions.
For example, the following instruction produces a xlsx file with actuarial commutation symbols for all integer and half year ages, for TV7377 mortality table:

```
tv7377_ct_f2.df_commutation_table_frac().to_excel(excel_writer='frac2' + '.xlsx',
     sheet_name='frac2', index=False, freeze_panes=(1, 1))
```

Figure 1 illustrates an excerpt of the the produced excel file for TV7377 actuarial table for years fractioned in semesters.

x	lx	dx	qx	px	Dx	Nx	Sx	Cx	Mx	Rx
0	100000	584	0,00584	0,99416	100000	7794804	464818252,1	578,2462	23202,03	3215202
0,5	99416	584	0,005874	0,994126	98436,51	7694804	457023448,5	572,549	22623,79	3192000
1	98832	48	0,000486	0,999514	96894,12	7596367	449328644,9	46,59518	22051,24	3169376
1,5	98784	48	0,000486	0,999514	95892,88	7499473	441732277,7	46,1361	22004,64	3147325
2	98736	29,5	0,000299	0,999701	94901,96	7403580	434232804,7	28,07512	21958,51	3125320
2,5	98706,5	29,5	0,000299	0,999701	93938,87	7308678	426829224,6	27,79851	21930,43	3103362
3	98677	23	0,000233	0,999767	92985,54	7214739	419520546,4	21,45988	21902,63	3081431
3,5	98654	23	0,000233	0,999767	92047,95	7121754	412305807,1	21,24845	21881,17	3059529
...	...	...	...	...	...	...	...	...	...	...
104	12	4	0,333333	0,666667	1,530254	3,533177	7,265558967	0,505059	1,495443	3,461593
104,5	8	4	0,5	0,5	1,010118	2,002923	3,732382231	0,500083	0,990384	1,96615
105	4	1,5	0,375	0,625	0,500083	0,992805	1,729458998	0,185683	0,490301	0,975766
105,5	2,5	1,5	0,6	0,4	0,309472	0,492723	0,736653597	0,183854	0,304618	0,485465
106	1	0,5	0,5	0,5	0,122569	0,18325	0,24393104	0,060681	0,120764	0,180847
106,5	0,5	0,5	1	0	0,060681	0,060681	0,060680858	0,060083	0,060083	0,060083
107	0	0	1	0	0	0	0	0	0	0

Figure 1: Excerpt of TV7377 semiannually fractional actuarial table

4 Some lifeactuary Functions and Syntax

4.1 Life Annuities

Table 1: Actuarial Notation and Syntax Formula for Life Annuities

Notation	**Description**	**Syntax**
a_x	whole life annuity	ax(x,1)
$\ddot{a}_x$	whole life annuity due	aax(x,1)
${}_{t\vert}a_x$	t years deferred whole life annuity	t_ax(x,1,t)
${}_{t\vert}\ddot{a}_x$	t years deferred whole life annuity due	t_aax(x,1,t)
$a_x^{(m)}$	whole life annuity payable m times per year	ax(x,m)
$\ddot{a}_x^{(m)}$	whole life annuity due payable m times per year	aax(x,m)
${}_{t\vert}a_x^{(m)}$	t years deferred whole life annuity payable m times per year	t_ax(x,m,t)
${}_{t\vert}\ddot{a}_x^{(m)}$	t years deferred whole life annuity due payable m times per year	t_aax(x,m,t)
$a_{x:\overline{n}\vert}$	n year temporary life annuity	nax(x,n,1)
$\ddot{a}_{x:\overline{n}\vert}$	n year temporary life annuity due	naax(x,n,1)
${}_{t\vert}a_{x:\overline{n}\vert}$	t year deferred n year temporary life annuity	t_nax(x,n,1,t)
${}_{t\vert}\ddot{a}_{x:\overline{n}\vert}$	t year deferred n year temporary life annuity due	t_naax(x,n,1,t)
$a_{x:\overline{n}\vert}^{(m)}$	n year temporary life annuity payable m times per year	nax(x,n,m)
$\ddot{a}_{x:\overline{n}\vert}^{(m)}$	n year temporary life annuity due payable m times per year	naax(x,n,m)
${}_{t\vert}a_{x:\overline{n}\vert}^{(m)}$	t year deferred n year temporary life annuity payable m times per year	t_nax(x,n,m,t)
${}_{t\vert}\ddot{a}_{x:\overline{n}\vert}^{(m)}$	t year deferred n year temporary life annuity due payable m times per year	t_naax(x,n,m,t)

Table 2: Actuarial Notation and Syntax Formula for Increasing Life Annuities

Notation	**Description**	**Syntax**
${}_{t\vert}Ia_{x:\overline{n}\vert}^{(m)r}$	t-years deferred n-year temporary increasing life annuity, payable m times per year. First payment C and increasing/decreasing amount r	t_nIax(x,n,m,t,C,r)
${}_{t\vert}I\ddot{a}_{x:\overline{n}\vert}^{(m)r}$	t-years deferred n-year temporary increasing life annuity, payable m times per year. First payment C and increasing/decreasing amount r	t_nIaax(x,n,m,t,C,r)

For life annuities with terms varying geometrically, the Actuarial Table must be built with an increasing rate g and the functions from Table 1 are applied.

4.2 Life Insurances

The following table resumes the available function for life insurances. As usual in the acturial notation, the capital letters with bar refer to payments due in the moment of death and the absense of bar refers to payments due in the end of the year in which the death occurs.

Table 3: Actuarial Notation and Syntax Formula for Life Insurances - fixed capitals

Notation	**Description**	**Syntax**
${}_nE_x$	pure endowment	nEx(x,n)
A_x	whole life insurance (end of the year)	Ax(x)
$\bar{A}_x$	whole life insurance (moment of death)	Ax_(x)
${}_{t\vert}A_x$	t years deferred whole life insurance (end of the year)	t_Ax(x,t)
${}_{t\vert}\bar{A}_x$	t years deferred whole life insurance (moment of death)	t_Ax_(x,t)
$A^1_{x:\overline{n}\vert}$	term life insurance (end of the year)	nAx(x,n)
$\bar{A}^1_{x:\overline{n}\vert}$	term life insurance (moment of death)	nAx_(x,n)
${}_{t\vert}A^1_{x:\overline{n}\vert}$	t years deferred term life insurance (end of the year)	t_nAx(x,n,t)
${}_{t\vert}\bar{A}^1_{x:\overline{n}\vert}$	t years deferred term life insurance (moment of death)	t_nAx_(x,n,t)
$A_{x:\overline{n}\vert}$	endowment insurance (end of the year)	nAEx(x,n)
$\bar{A}_{x:\overline{n}\vert}$	endowment insurance (moment of death)	nAEx_(x,n)
${}_{t\vert}A_{x:\overline{n}\vert}$	t-years deferred endowment insurance (end of the year)	t_nAEx(x,n,t)
${}_{t\vert}\bar{A}_{x:\overline{n}\vert}$	t-years deferred endowment insurance (moment of death)	t_nAEx_(x,n,t)

Table 4: Actuarial Notation and Syntax Formula for Life Insurances - variable capitals

Notation	**Description**	**Syntax**
$(IA)_x$	whole life insurance with arithmetically increasing capitals (end of the year)	IAx(x)
$(I\bar{A})_x$	whole life insurance with arithmetically increasing capitals (moment of death)	IAx_(x)
$(IA)_{x:\overline{n}\vert}$	term life insurance with arithmetically increasing capitals (end of the year)	nIAx(x,n)
$(I\bar{A})_{x:\overline{n}\vert}$	term life insurance with arithmetically increasing capitals (moment of death)	nIAx_(x,n)
${}_{t\vert}(IA)^r_{x:\overline{n}\vert}$	t-years deferred term life insurance with capitals evolving arithmetically (increasing or decreasing). First Capital C and increase amount r (end of the year)	nIArx(x,n,t,C,r)
${}_{t\vert}(I\bar{A})^r_{x:\overline{n}\vert}$	t-years deferred term life insurance with capitals evolving arithmetically (increasing or decreasing). First Capital C and increase amount r (moment of death)	nIArx_(x,n,t,C,r)

4.3 Financial Annuities

Table 5: Actuarial Notation and Syntax Formula for Financial Annuities

Notation	Description	Syntax
$a_{\overline{n}\rvert}$	n-year immediate financial annuity	an(n)
$\ddot{a}_{\overline{n}\rvert}$	n-year due financial annuity	aan(n)
a_{∞}	perpetual immediate financial annuity	a(None)
$\ddot{a}_{\infty}$	perpetual due financial annuity	aa(None)
${}_{r}(Ia)^{(m)}_{\overline{n}\rvert}$	n-year immediate financial annuity with first payment C and evolving arithmetically (increasing [$r > 0$] or decreasing [$r < 0$]). Payment increases in each period of the interest rate.	Ian(n,C,r)
${}_{r}(I\ddot{a})^{(m)}_{\overline{n}\rvert}$	n-year due financial annuity with first payment C and evolving arithmetically (increasing [$r > 0$] or decreasing [$r < 0$]). Payment increases in each period of the interest rate.	Iaan(n,C,r)
${}_{r}(I^{(m)}a)^{(m)}_{\overline{n}\rvert}$	n-year immediate financial annuity with first payment C and evolving arithmetically (increasing [$r > 0$] or decreasing [$r < 0$]). Payments increase in each payment period.	Iman(n,C,r)
${}_{r}(I^{(m)}\ddot{a})^{(m)}_{\overline{n}\rvert}$	n-year due financial annuity with first payment C and evolving arithmetically (increasing [$r > 0$] or decreasing [$r < 0$]). Payments increase in each payment period.	Imaan(n,C,r)
${}_{g}(Ga)^{(m)}_{\overline{n}\rvert}$	n-year immediate financial annuity with first payment C and evolving geometrically with rate g. Payments change in each period of the interest rate.	Gan(n,C,g)
${}_{g}(G\ddot{a})^{(m)}_{\overline{n}\rvert}$	n-year due financial annuity with first payment C and evolving geometrically with rate g. Payments change in each period of the interest rate.	Gaan(n,C,g)
${}_{g}(G^{(m)}a)^{(m)}_{\overline{n}\rvert}$	n-year immediate financial annuity with first payment C and evolving geometrically with rate g. Payments change in each payment period.	Gman(n,C,g)
${}_{g}(G^{(m)}\ddot{a})^{(m)}_{\overline{n}\rvert}$	n-year due financial annuity with first payment C and evolving geometrically with rate g. Payments change in each payment period.	Gmaan(n,C,g)

For annuities paid m times per year, the class Annuities Certain must be initiated with the correspondent frequency m. Examples are presented in section 7.

5 Life Annuities

A life annuity corresponds to a series of payments paid as long as an individual is alive on the payment date. The life annuity can be temporary or payable for whole life, the payments are due in the beginning (annuity due) or at the end of the periods (annuity immediate) and starts immediately in the next period or after some delay (deferred annuities). The payments are constant through the all term of contract or are variable (with or without a mathematical regularity). The number of payments in each period of the interest rate may also be defined.

The computation of the present value of all these life annuities is available in the library, and are presented in this chapter. If using the *CommutationTable* class, defined in section 2.6.1, life annuities functions are available for integer ages and terms. If using *CommutationTableFrac* class, see section 2.6.2, life annuities function will be available for fractional ages and non-integer terms.

For a more general approach, the library includes a function, see subsection 5.3.4, which computes the present value of a given series of cash-flows, with a given set of interest rates and a defined set of probabilities.

5.1 Whole Life Annuities

5.1.1 ax

Actuarial Notation: a_x and $a_x^{(m)}$

Usage

```
ax(x, m=1)
```

Description: Returns the actuarial present value of a whole life annuity of 1 per time period. Payments of $1/m$ are made m times per year at the end of the periods.

Parameters

x	age at the beginning of the contract
m	number of payments in each period of the interest rate

Examples

```
tv7377_ct.ax(50, 1)  # 21.554432773700235
tv7377_ct.ax(50, 4)  # 21.929432773700235
```

5.1.2 aax

Actuarial Notation: $\ddot{a}_x$ and $\ddot{a}_x^{(m)}$

Usage

```
aax(x, m=1)
```

Description: Returns the actuarial present value of a whole life annuity due of 1 per time period. The payments of $1/m$ are made m times per year at the beginning of the periods.

Parameters

x	age at the beginning of the contract
m	number of payments in each period of the interest rate

Examples

```
tv7377_ct.aax(50, 1)  # 22.55443277370024
tv7377_ct.aax(50, 4)  # 22.17943277370024
```

5.1.3 t_ax

Actuarial Notation: ${}_{t|}a_x$ and ${}_{t|}a_x^{(m)}$

Usage

```
t_ax(x, m=1, defer=0)
```

Description: Returns the actuarial present value of a immediate whole life annuity of 1 per time period, deferred t periods. The payments of $1/m$ are made m times per year at the end of the periods.

Parameters

x	age at the beginning of the contract
m	number of payments in each period of the interest rate
defer	number of deferment years

Observation: t_ax(x, m, defer=0) = ax(x, m)

Examples

```
tv7377_ct.t_ax(50, 1, 5)  # 16.899196591768252
tv7377_ct.t_ax(50, 4, 5)  # 17.231374204075433
```

5.1.4 t_aax

Actuarial Notation: ${}_{t|}\ddot{a}_x$ and ${}_{t|}\ddot{a}_x^{(m)}$

Usage

```
t_aax(x, m=1, defer=0)
```

Description: Returns the actuarial present value of a whole life annuity due of 1 per time period, deferred t periods. The payments of $1/m$ are made m times per year at the beginning of the periods.

Parameters

x	age at the beginning of the contract
m	number of payments in each period of the interest rate
defer	number of deferment years

Observation: t_aax(x, m, defer=0) = aax(x, m)

Examples

```
tv7377_ct.t_aax(50, 1, 5)  # 17.78500355792074
tv7377_ct.t_aax(50, 4, 5)  # 17.45282594561355
```

5.2 Temporary Life Annuities

5.2.1 nax

Actuarial Notation: $a_{x:\overline{n}|}$ and $a^{(m)}_{x:\overline{n}|}$

Usage

```
nax(x, n, m=1)
```

Description: Returns the actuarial present value of an immediate n term life annuity of 1 per time period. The payments of $1/m$ are made m times per year at the end of the periods.

Parameters

x	age at the beginning of the contract
n	number of periods until the end of the contract (measured in periods of the interest rate)
m	number of payments in each period of the interest rate

Examples

```
tv7377_ct.nax(50, 10, 1)  # 8.756215803256639
tv7377_ct.nax(50, 10, 4)  # 8.839775242816884
```

5.2.2 naax

Actuarial Notation: $\ddot{a}_{x:\overline{n}|}$ and $\ddot{a}^{(m)}_{x:\overline{n}|}$

Usage

```
naax(x, n, m=1)
```

Description: Returns the actuarial present value of a n term life annuity due of 1 per time period. The payments of $1/m$ are made m times per year at the beginning of the periods.

Parameters

x	age at the beginning of the contract
n	number of periods until the end of the contract (measured in periods of the interest rate)
m	number of payments in each period of the interest rate

Examples

```
tv7377_ct.naax(50, 10, 1)  # 8.979040975417291
tv7377_ct.naax(50, 10, 4)  # 8.895481535857046
```

5.2.3 t_nax

Actuarial Notation: ${}_{t|}a_{x:\overline{n}|}$ and ${}_{t|}a^{(m)}_{x:\overline{n}|}$

Usage

```
t_nax(x, n, m=1, defer=0)
```

Description: Returns the actuarial present value of a immediate n term life annuity of 1 per time period, deferred t periods. The payments of $1/m$ are made m times per year at the end of the periods.

Parameters

x	age at the beginning of the contract
n	number of periods until the end of the contract (measured in periods of the interest rate)
m	number of payments in each period of the interest rate
defer	number of deferment years

Observation: t_nax(x, m, defer=0) = nax(x, m)

Examples

```
tv7377_ct.t_nax(50, 10, 1, 5)  # 7.670292001795834
tv7377_ct.t_nax(50, 10, 4, 5)  # 7.750622055449898
```

5.2.4 t_naax

Actuarial Notation: ${}_{t|}\ddot{a}_{x:\overline{n}|}$ and ${}_{t|}\ddot{a}^{(m)}_{x:\overline{n}|}$

Usage

```
t_naax(x, m=1, defer=0)
```

Description: Returns the actuarial present value of a n term life annuity due of 1 per time period, deferred t periods. The payments of $1/m$ are made m times per year at the beginning of the periods.

Parameters

x	age at the beginning of the contract
n	number of periods until the end of the contract (measured in periods of the interest rate)
m	number of payments in each period of the interest rate
defer	number of deferment years

Observation: t_naax(x, m, defer=0) = naax(x, m)

Examples

```
tv7377_ct.t_naax(50, 10, 1, 5)  # 7.8845054782066715
tv7377_ct.t_naax(50, 10, 4, 5)  # 7.804175424552608
```

5.3 Life Annuities with variable terms

In this section, are presented functions that compute the actuarial present value of life annuities whose payments are not constant overtime.
As special regularities, life annuities with terms evolving in arithmetic or geometric progression (increasing or decreasing) are well known and easily computed and this library presents easy to use solutions and functions.
As a general case, the library allows for the computation of the actuarial present value for a given set of cash-flows, interest rates and , that can vary without any regularity. The set of parameters should be provided in vector formats.

5.3.1 t_nIax

Actuarial Notation: ${}_{t|}(Ia)_{x:\overline{n}|}$ and ${}_{t|}(Ia)^{(m)}_{x:\overline{n}|}$

Usage

```
t_nIax(x, n, m=1, defer=0, first_amount=1, increase_amount=1)
```

Description: Returns the actuarial present value of an immediate n term life annuity, deferred t periods, with payments evolving in arithmetic progression. Payments of $1/m$ are made m times per year at the end of the periods. First amount and Increase amount may be different. For decreasing life annuities, the Increase Amount should be negative.

Parameters

x	age at the beginning of the contract
n	number of periods of the contract (measured in periods of the interest rate)
m	number of payments in each period of the interest rate
defer	number of deferment years
first_amount	amount of the first payment
increase_amount	amount of the increase amount increasing life annuities: increase_amount > 0 decreasing life annuities: increase_amount < 0

Observation: t_nIax(x, n, m, defer, 1, increase_amount=0) = t_nax(x, n, m, defer)

Examples

```
tv7377_ct.t_nIax(x=50, n=10, m=1, defer=0, first_amount=1, increase_amount=1)    # 46.33017
tv7377_ct.t_nIax(x=50, n=10, m=1, defer=0, first_amount=1, increase_amount=5)    # 196.62599
tv7377_ct.t_nIax(x=50, n=10, m=1, defer=0, first_amount=100, increase_amount=-5) # 687.75180
```

5.3.2 t_nIaax

Actuarial Notation: ${}_{t|}(I\ddot{a})_{x:\overline{n}|}$ and ${}_{t|}(I\ddot{a})^{(m)}_{x:\overline{n}|}$

Usage

```
t_nIaax(x, n, m=1, defer=0, first_amount=1, increase_amount=1)
```

Description: Returns the actuarial present value of a n term life annuity due, deferred t periods, with payments evolving in arithmetic progression. The payments are made m times per year at the beginning of the periods and the payment of each period is divided into m equal payments. First amount and Increase amount may be different. For decreasing life annuities, the *Increase Amount* should be negative.

Parameters

x	age at the beginning of the contract
n	number of periods until the end of the contract (measured in periods of the interest rate)
m	number of payments in each period of the interest rate
defer	number of deferment years
first_amount	amount of the first payment
increase_amount	amount of the increase amount increasing life annuities: increase_amount > 0 decreasing life annuities: increase_amount < 0

Observation: t_nIaax(x, n, m, defer, 1, increase_amount=0) = t_naax(x, n, m, defer)

Examples

```
tv7377_ct.t_nIaax(x=50, n=10, m=1, defer=0, first_amount=1, increase_amount=1)    # 47.5374643
tv7377_ct.t_nIaax(x=50, n=10, m=1, defer=0, first_amount=1, increase_amount=5)    # 201.771158
tv7377_ct.t_nIaax(x=50, n=10, m=1, defer=0, first_amount=100, increase_amount=-5) # 705.111980
```

5.3.3 Geometric Life Annuities

For life annuities with payments evolving in geometric progression (increasing or decreasing) the growth rate (g) should be included when computing the actuarial table (see section 2.6.1) and then use the life annuities functions presented in the previous sections.

Examples

```
from lifeactuary import commutation_table as ct

# reads SOA table
soa = rst.SoaTable('/soa_tables/' + 'TV7377' + '.xml')

# creates an actuarial table from qx of SOA table with geometric increase of 5% on payments
tv7377_ctg_inc = ct.CommutationFunctions(i=2, g=5, data_type='q', mt=soa.table_qx, perc=100,
    app_cont=False)


```

```
# creates an actuarial table from qx of SOA table with geometric decrease of 5% on payments
tv7377_ctg_dec = ct.CommutationFunctions(i=2, g=-5, data_type='q', mt=soa.table_qx, perc=100,
    app_cont=False)

# actuarial present value of a geometrically evolving life annuity for a 50 years old
    indidivual, for 10 years period
tv7377_ctg_inc.naax(50,10,1)  # 11.18091822195998
tv7377_ctg_dec.naax(50,10,1)  # 7.281682932595854
```

5.3.4 Present_Value Function

This function generalizes any of the above, returning the present value of a series of cash-flows (introduced in vector mode), where the interest rate for each period may differ as well as the probability assigned to each payment/benefit. According to the defined probabilities, the function returns the present value (for probabilities equal to 1) or the actuarial present value of the series of cash-flows.

Usage

```
present_value(probs, age, spot_rates, capital)
```

Description: This function computes the expected present value of a cash-flow, that can be contingent on some probabilities. The payments are considered at the end of the period.

Parameters

probs	vector of probabilities. For using the instantiated actuarial table, introduce probs=None.
age	age at the beginning of the contract
spot_rates	vector of interest rates for the considered time periods
capital	vector of cash-flow amounts

Examples

```
pv1 = tv7377_ct.present_value(probs=1, age=None,
      spot_rates=[1.2, 1.4, 1.8, 1.6, 1.9], capital=[100, -25, 120, 300, -50])
print('Present Value:', pv1)
# 425.750701233034

pv2 = tv7377_ct.present_value(probs=None, age=35,
      spot_rates=[1.2, 1.4, 1.8, 1.6, 1.9], capital=[100, -25, 120, 300, -50])
print('Present Value:', pv2)
# 424.2408517830521
```

6 Pricing Life Insurance

When pricing life insurance, there is a need to compute the present value of the benefit payment, for a given mortality table and an interest rate.
This library includes functions that allow for pricing the most common contracts in life insurance, such as Pure Endowment, Whole Life and Temporary Insurances, Endowment Insurance as well as the traditional life insurance with increasing (or decreasing) capitals.
The available functions allow for the pricing of any other type of life insurance, whose actuarial evaluation makes use of the functions available in this library.
All the functions are available for non-integer ages and terms, if the *CommutationTableFrac* is used for building the Actuarial Table.

6.1 Pure Endowment / Deferred Capital / Expected Present Value

Actuarial Notation: ${}_nE_x$

Usage

```
nEx(x, n)
```

Description: Returns the present value of a Pure Endowment of 1 for an aged x individual, paid at age $x+n$.

Parameters

x	age at the beginning of the contract
m	years until payment, if (x) is alive

Examples

```
tv7377_ct.nEx(50, 5)  # 0.8858069661524854
tv7377_ct.nEx(50, 10) # 0.7771748278393479
tv7377_ct.nEx(80, 10) # 0.2283081320230277
```

6.2 Whole Life Insurance

6.2.1 Ax

Actuarial Notation: A_x

Usage

```
Ax(x)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a whole life insurance (i.e. net single premium), that pays 1 at the end of the year of death.

Parameters

x	age at the beginning of the contract

Examples

```
tv7377_ct.Ax(50)  # 0.5577562201235239
```

6.2.2 Ax_

Actuarial Notation: $\bar{A}_x$

Usage

```
Ax_(x)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a whole life insurance (i.e. net single premium), that pays 1 at the moment of death.

Parameters

x	age at the beginning of the contract

Examples

```
tv7377_ct.Ax_(50)  # 0.5633061699539695
```

6.2.3 t_Ax

Actuarial Notation: ${}_{t|}A_x$

Usage

```
t_Ax(x, defer=0)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a whole life insurance (i.e. net single premium), that pays 1 at the end of year of death. The contract is deferred t years.

Parameters

x	age at the beginning of the contract
t	deferment period (in years)

Observation: t_Ax(x, defer=0) = Ax(x)

Examples

```
tv7377_ct.t_Ax(50,2)  # 0.550183040772438
```

6.2.4 t_Ax_

Actuarial Notation: ${}_{t|}\overline{A}_x$

Usage

```
t_Ax_(x, defer=0)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a whole life insurance (i.e. net single premium), that pays 1 at the moment of death. The contract is deferred t years.

Parameters

x	age at the beginning of the contract
defer	deferment period (in years)

Observation: t_Ax_(x, defer=0) = Ax_(x)

Examples

```
tv7377_ct.t_Ax(50,2)  # 0.5556576337284301
```

6.3 Temporary Life Insurance

6.3.1 nAx

Actuarial Notation: $A^1_{x:\overline{n}|}$

Usage

```
nAx(x, n)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a term (temporary) life insurance (i.e. net single premium), that pays 1, at the end of the year of death.

Parameters

x	age at the beginning of the contract
n	number of years of the contract

Examples

```
tv7377_ct.nAx(50,10)  # 0.046765545191685375
```

6.3.2 nAx_

Actuarial Notation: $\bar{A}^1_{x:\overline{n}|}$

Usage

```
nAx_(x, n)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a term (temporary) life insurance (i.e. net single premium), that pays 1, at the moment of death.

Parameters

x	age at the beginning of the contract
n	number of years of the contract

Examples

```
tv7377_ct.t_Ax_(50,10)  # 0.047230885460862126
```

6.3.3 t_nAx

Actuarial Notation: ${}_{t|}A^{1}_{x:\overline{n}|}$

Usage

```
t_nAx(x, n, defer=0)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a term (temporary) deferred life insurance (i.e. net single premium), that pays 1, at the end of the year of death.

Parameters

x	age at the beginning of the contract
n	number of years of the contract
defer	number of years of deferment

Observation: t_nAx(x, n, defer=0) = nAx(x, n)

Examples

```
tv7377_ct.t_nAx(50,10,5)  # 0.059615329779334834
```

6.3.4 t_nAx_

Actuarial Notation: ${}_{t|}\bar{A}^{1}_{x:\overline{n}|}$

Usage

```
t_nAx_(x, n, defer=0)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a term (temporary) deferred life insurance (i.e. net single premium), that pays 1, at the moment of death.

Parameters

x	age at the beginning of the contract
n	number of years of the contract
defer	number of years of deferment

Observation: t_nAx_(x, n, defer=0) = nAx_(x, n)

Examples

```
tv7377_ct.t_nAx_(50,10,5)  # 0.060208531750847685
```

6.4 Endowment Insurance

6.4.1 nAEx

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

Usage

```
nAEx(x, n)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of an Endowment life insurance (i.e. net single premium), that pays 1, at the end of year of death or 1 if (x) survives to age $x+n$.

Parameters

x	age at the beginning of the contract
n	number of years of the contract

Examples

```
tv7377_ct.nAEx(50,10)  # 0.8239403730310333
```

6.4.2 nAEx_

Actuarial Notation: $\bar{A}_{x:\overline{n}|}$

Usage

```
nAEx(x, n)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of an Endowment life insurance (i.e. net single premium), that pays 1, at the moment of death or 1 if (x) survives to age $x+n$.

Parameters

x	age at the beginning of the contract
n	number of years of the contract

Examples

```
tv7377_ct.nAEx_(50,10)  # 0.8244057133002101
```

6.4.3 t_nAEx

Actuarial Notation: ${}_{t|}A_{x:\overline{n}|}$

Usage

```
t_nAEx(x, n, defer=0)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a deferred Endowment life insurance (i.e. net single premium) that pays 1, at the end of year of death or 1 if (x) survives to age $x + t + n$.

Parameters

x	age at the beginning of the contract
n	number of years of the contract
defer	number of years of deferment

Examples

```
tv7377_ct.t_nAEx(50,10,2)  # 0.7863040688470341
```

6.4.4 t_AEx_

Actuarial Notation: ${}_{t|}\bar{A}_{x:\overline{n}|}$

Usage

```
t_nAEx_(x, n, defer=0)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a deferred Endowment life insurance (i.e. net single premium) that pays 1, at the moment of death or 1 if (x) survives to age $x + t + n$.

Parameters

x	age at the beginning of the contract
n	number of years of the contract
defer	number of years of deferment

Examples

```
tv7377_ct.t_nAEx_(50,10,2)  # 0.7868146552887256
```

6.5 Temporary Life Insurance with variable Capitals

6.5.1 IAx

Actuarial Notation: $(IA)_x$

Usage

```
IAx(x)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a Term Life Insurance (i.e. net single premium), that pays $1+k$, at the end of year of death, if death occurs between ages $x+k$ and $x+k+1$, for k=0, 1, ... The capital of the first year is equal to the rate of the progression.

Parameters

x	age at the beginning of the contract

Examples

```
tv7377_ct.IAx(50)  # 15.807431562003355
```

6.5.2 nIAx

Actuarial Notation: $(IA)_{x:\overline{n}|}$

Usage

```
nIAx(x,n)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of an arithmetically increasing life insurance (i.e. net single premium), that pays $1+k$, at the end of the year if death happens between age $x+k$ and $x+k+1$, k=0,. . . , $n-1$.

Parameters

x	age at the beginning of the contract
n	number of years of the contract

Examples

```
tv7377_ct.nIAx(50,10)  # 0.2751855520152587
```

6.5.3 nIArx

This function computes the actuarial present value of a term insurance whose capitals increase arithmetically, allowing for different amounts of initial capital and increase amount. It corresponds to a generalization of function nIAx, where the first amount is equal to increase amount.

Actuarial Notation: ${}_r(IA)_{x:\overline{n}|}$

Usage

```
nIArx_ct(x,n, defer=0, first_amount=1, increase_amount=1)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a term life insurance (i.e. net single premium), that pays (first_amount + $k\times$ increase_amount), at the end of the year if death occurs between ages $x+k$ and $x+k+1$, for $k = 0,\ldots,n-1$. Allows the computation for decreasing capitals. The first capital may differ from the increasing/decreasing amount.

Parameters

x	age at the beginning of the contract
n	number of years of the contract
defer	number of deferment years
first_amount	insured amount in the first year of the contract
increase_amount	rate of increasing (if > 0) or decreasing (if < 0)

Examples

```
tv7377_ct.nIArx(50,10,0,1000,50)     # 58.18654553286392
tv7377_ct.nIArx(50,10,10,1000,50)    # 133.3402470815534
tv7377_ct.nIArx(50,10,0,1000,-50)    # 35.344544850506836
tv7377_ct.nIArx(50,10,10,1000,-50)   # 28.027981923486493
```

Observation: nIAx(x, n) = nIArx(x, n, defer=0,first_amount=1, increase_amount=1)

6.5.4 nIArx_

This function computes the actuarial present value of a term insurance whose capitals increase arithmetically, allowing for different amounts of initial capital and increase amount. It corresponds to a generalization of function nIAx_, where the first amount is equal to increase amount.

Actuarial Notation: $(I\bar{A})^r_{x:\overline{n}|}$

Usage

```
nIArx_(x,n, defer=0, first_amount=1, increase_amount=1)
```

Description: Returns the Expected Present Value (EPV) [Actuarial Present Value (APV)] of a term life insurance (i.e. net single premium), that pays (first_amount + $k\times$ increase_amount), at the moment of death if death happens between age $x+k$ and $x+k+1$, for $k = 0,\ldots,n-1$.

Parameters

x	age at the beginning of the contract
n	number of years of the contract
defer	number of deferment years
first_amount	insured amount in the first year of the contract
increase_amount	rate of increasing (if > 0) or decreasing (if < 0)

Examples

```
tv7377_ct.nIArx_(50,10,0,1000,50)     # 58.765530395538875
tv7377_ct.nIArx_(50,10,10,1000,50)    # 134.66704838825683
tv7377_ct.nIArx_(50,10,0,1000,-50)    # 35.69624052618538
tv7377_ct.nIArx_(50,10,10,1000,-50)   # 28.306874184857506
```

7 Financial Annuities

7.1 Class Annuities_Certain

class *Annuities_Certain*
This class instantiates the methods for the computation of financial annuities, for a given interest rate and a chosen frequency of payments m (in each period of the interest rate).

Usage

```
AnnuitiesCertain(interest_rate, m=1)
```

Description

Initializes the AnnuitiesCertain class so that we can compute the present value of financial annuities.

Parameters

interest_rate	interest rate, in percentage (e.g. use 5 for 5%)
m	frequency of payments, in each period of the interest rate

Examples

```
from lifeactuary import annuities_certain as ac

# instantiates a methods for computing financial annuities with a 5% annual interest rate,
    with annual payments
i1=ac.Annuities_Certain(5,1)

# instantiates a methods for computing financial annuities with a 5% annual interest rate,
    with quarterly payments
i4=ac.Annuities_Certain(5,4)
```

The following methods are available after instantiating the class:

7.1.1 im

Actuarial Notation	i_m
Definition	$m\left[(1+i)^{1/m}-1\right]$
Usage	irate.im
Example	i4.im
Result	0.04908893771615741

7.1.2 vm

Actuarial Notation	v_m
Definition	$\frac{1}{1+\frac{i_m}{m}}$
Usage	irate.vm
Example	i4.vm
Result	0.9878765474230741

7.1.3 dm

Actuarial Notation	d_m
Definition	$\left(1-\frac{i}{1+i}\right)^{1/m}-1$
Usage	irate.dm
Example	i4.dm
Result	0.0484938103077039

7.2 Constant Terms Financial Annuities

7.2.1 an

Actuarial Notation: $a_{\overline{n|}}$ and $a^{(m)}_{\overline{n|}}$ or $a_{\overline{\infty|}}$ and $a^{(m)}_{\overline{\infty|}}$

Usage

```
an(terms)
```

Description: Returns the present value of an immediate n term financial annuity with payments equal to 1. Payments are made in the end of the periods. In fractional annuities, payments of $1/m$ are made m times per year at the end of the periods.

Parameters

terms	number of periods (measured in periods of the interest rate) (terms=None) or (terms=0) returns the present value of the perpetual annuity

Examples

```
i1.an(10) # 7.721734929184813
i4.an(10) # 7.86504586209782

i1.an(None) # 19.999999999999982
i4.an(0) # 20.371188429095998
```

7.2.2 aan

Actuarial Notation: $\ddot{a}_{\overline{n|}}$ and $\ddot{a}^{(m)}_{\overline{n|}}$ or $\ddot{a}_{\overline{\infty|}}$ and $\ddot{a}^{(m)}_{\overline{\infty|}}$

Usage

```
aan(terms)
```

Description: Returns the present value of a due n term financial annuity with payments equal to 1. Payments are made in the beginning of periods. In fractional annuities, payments of $1/m$ are made m times per year at the end of the periods.

Parameters

terms	number of periods (measured in periods of the interest rate) (terms=None) or (terms=0) returns the present value of the perpetual annuity

Examples

```
i1.aan(10) # 8.107821675644054
i4.aan(10) # 7.9615675487126305

i1.aan(None) # 20.999999999999982
i4.aan(0) # 20.621188429095998
```

7.3 Variable Terms Financial Annuities

In this section we present functions that allow the computation of the present value of financial annuities with variable terms, for the particular cases where the terms evolve in arithmetic or geometric progressions.
For both cases, functions are defined in a general way such that the first term may differ from the rate of growth and the same function allows for increasing and decreasing terms.
For annuities with payments more frequent than the interest rate period, two approaches are considered and corresponding functions are developed: (1) payments level within each interest period and increase/decrease from one interest period to the next and (2) payments increase in each payment period.

Annuities with terms evolving Arithmetically

7.3.1 Ian

Actuarial Notation: $(Ia)_{\overline{n}|}$ and $(Ia)^{(m)}_{\overline{n}|}$ or $(Da)_{\overline{n}|}$ and $(Da)^{(m)}_{\overline{n}|}$

Usage

```
Ian(terms, payment=1, increase=1)
```

Description: Returns the present value of an immediate n term financial annuity with payments increasing/decreasing arithmetically. Payments are made in the end of the periods. First payment and increase amount may differ. In fractional annuities, payments level within each interest period and increase/decrease from one interest period to the next.

Parameters

terms	number of periods (measured in periods of the interest rate)
payment	amount of the first payment
increase	increase amount of payments (>0 for increasing annuities and <0 for decreasing annuities)

Actuarial Formula
For C - first payment , r - rate of increasing/decreasing, i-annual interest rate, n - number of terms, m-frequency of payments

$$_{(C,r)}(Ia)^{(m)}_{\overline{n}|} = C\, a^{(m)}_{\overline{n}|} + r\, \frac{a_{\overline{n}|i} - nv^n}{i^{(m)}}$$

Examples

```python
a4 = ac.Annuities_Certain(interest_rate=5, m=12)
r4 = a4.Ian(terms=20, payment=2000 * 12, increase=400 * 12)
print(r4)
#789369.5624059099
```

7.3.2 Iaan

Actuarial Notation: $I\ddot{a}_{\overline{n}|}$ and $I\ddot{a}_{\overline{n}|}^{(m)}$ or $D\ddot{a}_{\overline{n}|}$ and $D\ddot{a}_{\overline{n}|}^{(m)}$

Usage

```python
Iaan(terms, payment=1, increase=1)
```

Description: Returns the present value of a due n term financial annuity with payments increasing/decreasing arithmetically. Payments are made in the beginning of the periods. First payment and increase amount may differ. In fractional annuities, payments level within each interest period and increase/decrease from one interest period to the next.

Parameters

terms	number of periods (measured in periods of the interest rate)
payment	amount of the first payment
increase	increase amount of payments (> 0 for increasing annuities and < 0 for decreasing annuities)

Examples

```python
a5 = ac.Annuities_Certain(interest_rate=2, m=2)
r5 = a5.Iaan(terms=2, payment=1, increase=1)
print(r5)
# 2.946198813622495
```

7.3.3 Iman

Actuarial Notation: $(I^{(m)}a)_{\overline{n}|}$ and $(I^{(m)}a)_{\overline{n}|}^{(m)}$ or $(D^{(m)}a)_{\overline{n}|}$ and $(D^{(m)}a)_{\overline{n}|}^{(m)}$

Usage

```python
Iman(terms, payment=1, increase=1)
```

Description: Returns the present value of an immediate n-term financial annuity with payments increasing/decreasing arithmetically. Payments are made in the end of the periods. First payment and increase amount may differ. In fractional annuities, payments increase in each payment period.

Parameters

terms	number of periods (measured in periods of the interest rate)
payment	amount of the first payment
increase	increase amount of payments (> 0 for increasing annuities and < 0 for decreasing annuities)

Actuarial Formula
For C - first payment , r - rate of increasing/decreasing, i-annual interest rate, n - number of terms, m-frequency of payments

$$_{(C,r)}(I^{(m)}a)^{(m)}_{\overline{n|}} = C\, a^{(m)}_{\overline{n|}} + rm\, \frac{a^{(m)}_{\overline{n|}i} - nv^n}{i^{(m)}}$$

Examples

```python
a3 = ac.Annuities_Certain(interest_rate=3.3, m=12)
r3 = a3.Iman(terms=8, payment=25 * 12, increase=2 * 12)
print(r3)
# 9781.284321297218
```

Annuities with terms evolving Geometrically

For computing the present value of geometric financial annuities, let us consider the interest rate i_g which reflects the annual interest rate of the annuity, i, and the growth rate g. For these cases, let us define

$$i_g = \frac{1-g}{1+i} \qquad \text{and} \qquad v_g = \frac{1+g}{1+i}.$$

7.3.4 Gan

Actuarial Notation: $_g(Ga)_{\overline{n|}}$ and $_g(Ga)^{(m)}_{\overline{n|}}$

Usage

```python
Gan(terms,payment=1, grow=0)
```

Description: Returns the present value of an immediate n term financial annuity with payments increasing/decreasing geometrically. Payments are made in the end of the periods. In fractional annuities, payments level within each interest period and increase/decrease from one interest period to the next.

Parameters

terms	number of periods (measured in periods of the interest rate)
payment	amount of the first payment
grow	rate of growing of payments, in percentage

Actuarial Formula

For C - first payment , g - rate of increasing/decreasing, i-annual interest rate, n - number of terms, m-frequency of payments

$$
{}_{(C,g)}(Ga)^{(m)}_{\overline{n}|} = \begin{cases} \frac{C}{m}\,\frac{1}{(1+g)^{1/m}}\, a^{(m)}_{\overline{n}|\, i_g} & , \quad i \neq g \\ \\ Cnv^{1/m} & , \quad i = g \end{cases}
$$

Examples

```
g1=ac.Annuities_Certain(interest_rate=5, m=2)
g1.Gan(terms=5,payment=10,grow=10)
# 108.60048398210647
```

7.3.5 Gaan

Actuarial Notation: ${}_{g}(G\ddot{a})^{(m)}_{\overline{n}|}$

Usage

```
Gaan(terms,payment=1, grow=0)
```

Description: Returns the present value of an immediate n term financial annuity with payments increasing/decreasing geometrically. Payments are made in the end of the periods. In fractional annuities, payments level within each interest period and increase/decrease from one interest period to the next.

Parameters

terms	number of periods (measured in periods of the interest rate)
payment	amount of the first payment
grow	rate of growing of payments, in percentage

Examples

```
g2=ac.Annuities_Certain(interest_rate=5, m=4)
g2.Gaan(terms=5,payment=100,grow=10)
# 2185.9539086346845
```

7.3.6 Gman

Actuarial Notation: ${}_{g}(G^{(m)}a)_{\overline{n}|}$ or ${}_{g}(G^{(m)}a)^{(m)}_{\overline{n}|}$

Usage

```
Gman(terms,payment=1, grow=0)
```

Description: Returns the present value of an immediate n term financial annuity with payments increasing/decreasing geometrically. Payments are made in the end of the periods. In fractional annuities, payments increase in each payment period.

Parameters

terms	number of periods (measured in periods of the interest rate)
payment	amount of the first payment
grow	rate of growing of payments, in percentage

Actuarial Formula

For C - first payment , g - rate of increasing/decreasing, i-annual interest rate, n - number of terms, m-frequency of payments

$$_{(C,g)}(Ga)^{(m)}_{\overline{n|}} = \begin{cases} C a^{(m)}_{\overline{1|}\,i} \times \ddot{a}_{\overline{n|}\,i_g} & , \quad i \neq g \\ \\ C\, n\, a^{(m)}_{\overline{1|}\,i} & , \quad i = g \end{cases}$$

Examples

```
g3=ac.Annuities_Certain(interest_rate=5, m=2)
g3.Gman(terms=5,payment=10,grow=10)
# 53.022051853437205
```

7.3.7 Gmaan

Actuarial Notation: $_g(G^{(m)}\ddot{a})_{\overline{n|}}$ or $_g(G^{(m)}\ddot{a})_{\overline{n|}}$

Usage

```
Gmaan(terms,payment=1, grow=0)
```

Description: Returns the present value of an immediate n term financial annuity with payments increasing/decreasing geometrically. Payments are made in the beginning of the periods. In fractional annuities, payments increase in each payment period.

Parameters

terms	number of periods (measured in periods of the interest rate)
payment	amount of the first payment
grow	rate of growing of payments, in percentage

Examples

```
g4=ac.Annuities_Certain(interest_rate=5, m=2)
g4.Gmaan(terms=5,payment=10,grow=10)
#  51.80299115517991
```

8 Examples

In this section we present some interesting and more complete examples of application, for which the use of lifeactuary package is helpful and provides "easy to use" solutions and functions.

8.1 Survival Probabilities

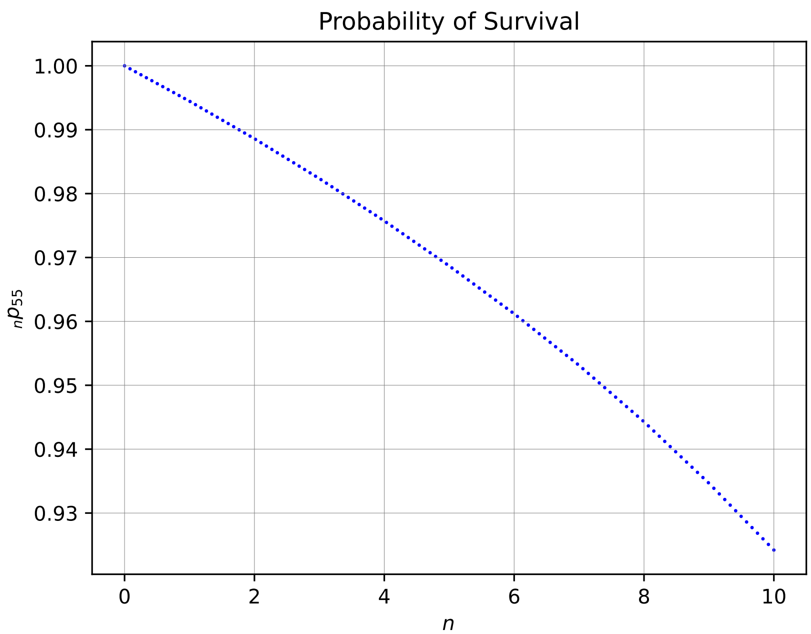
Example 1

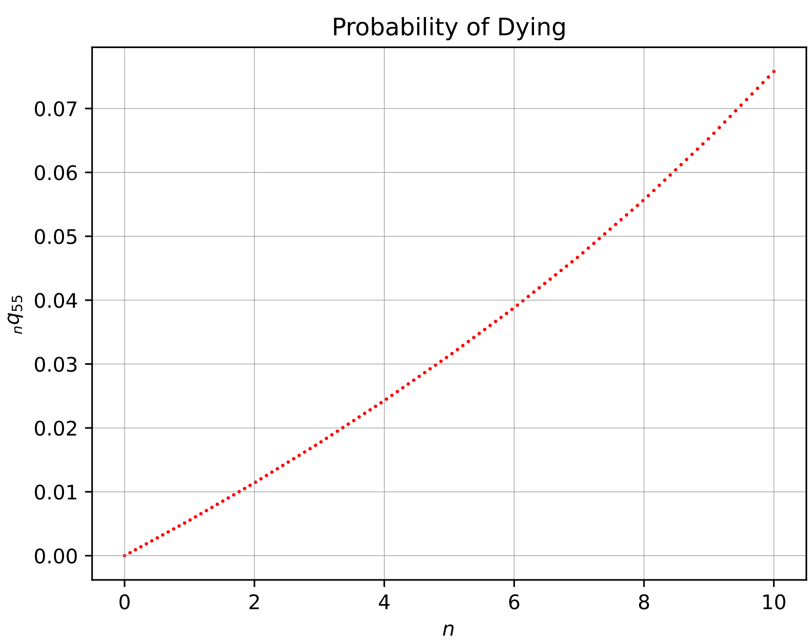
Consider a 50 year old individual and the TV7377 mortality table.

1. Determine the probabilities of (50) being alive in the end of each month of the following ten years, considering the Uniform Distribution of Death approximation.
2. Determine the probabilities of (50) not surviving up to the end of each month of the following ten years, considering the Uniform Distribution of Death approximation.
3. Build a dataframe with ages and estimated probabilities.
4. Export data to an Excel file.
5. Plot the estimated probabilities in a scatterplot.

```python
from lifeactuary import mortality_table as mtable, read_soa_table_xml as rst

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

mt = rst.SoaTable('/soa_tables/TV7377' + '.xml')
lt = mtable.MortalityTable(mt=mt.table_qx)

# Question 1
x = 55
n = np.linspace(0, 10, 10*12)
sprobs = [lt.npx(x=x, n=i, method='udd') for i in n]
dprobs = [lt.nqx(x=x, n=i, method='udd') for i in n]
ages = x + n
df = pd.DataFrame.from_dict({'n': n, 'x': ages, 'npx': sprobs, 'nqx': dprobs})

# Question 2
df.to_excel(excel_writer='example1.xlsx', sheet_name='example1', index=False, freeze_panes=(1,
    1))

# Question 3
plt.scatter(n, sprobs, s=.5, color='blue')

plt.xlabel(r'$n$')
plt.ylabel(r'${}_{n}p_{55}$')
plt.title('Probability of Survival')
plt.grid(visible=True, which='both', axis='both', color='grey', linestyle='-', linewidth=.1)
plt.savefig('example1s' + '.eps', format='eps', dpi=3600)
plt.show()

```

```
plt.scatter(n, dprobs, s=.5, color='red')

plt.xlabel(r'$n$')
plt.ylabel(r'${}_{n}q_{55}$')
plt.title('Probability of Dying')
plt.grid(visible=True, which='both', axis='both', color='grey', linestyle='-', linewidth=.1)
plt.savefig('example1d' + '.eps', format='eps', dpi=3600)
plt.show()
```





8.2 Life Tables and Life Annuities

Example 2

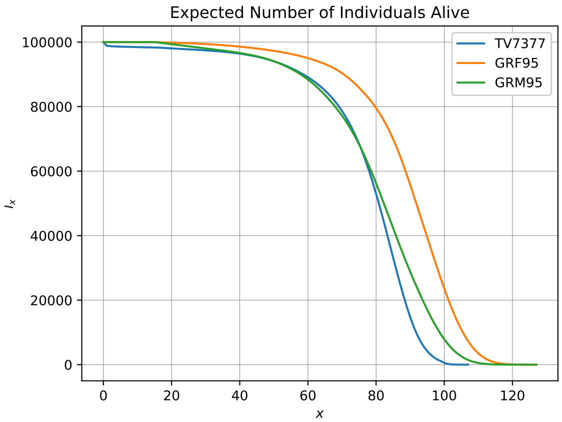
In this example, let us develop the Python code for answering the following questions:

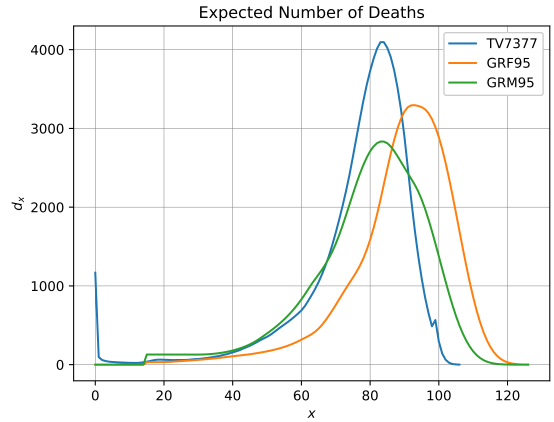
1. Import mortality tables TV7377, GRF95 and GRM95, from SOA Mortality Tables, with the columns x, l_x, p_x, q_x, e_x.
2. Construct an actuarial table considering a technical rate of interest of 4% *per annum*, and append the columns D_x and N_x to the tables produced in the previous question.
3. Plot the l_x, q_x, p_x, e_x and $\ln(D_x)$, comparing, in the same graph, the values of each mortality table.
4. Determine the net single premium (risk single premium) of a whole life annuity immediate, if someone 55 years old today, wants to receive 1000€ per year considering that:
   (a) The contract is paid at single premium.
   (b) The contract is paid at level premiums during 5 years.
5. Determine the net single premium (risk single premium) of a 10 years temporary due life annuity, if someone 55 years old today, wants to receive 1000 m.u. per year.

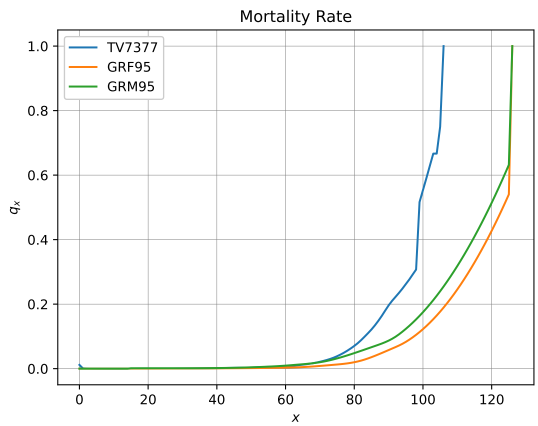
```python
from lifeactuary import mortality_table as mtable, commutation_table as ct, read_soa_table_xml
    as rst

import numpy as np
import os
import sys
import matplotlib.pyplot as plt

this_py = os.path.split(sys.argv[0])[-1][:-3]

def parse_table_name(name):
    return name.replace(' ', '').replace('/', '')

# Question 1
table_names = ['TV7377', 'GRF95', 'GRM95']
mt_lst = [rst.SoaTable('../../../soa_tables/' + name + '.xml') for name in table_names]
lt_lst = [mtable.MortalityTable(mt=mt.table_qx) for mt in mt_lst]


# Question 2
interest_rate = 4
ct_lst = [ct.CommutationFunctions(i=interest_rate, g=0, mt=mt.table_qx) for mt in mt_lst]

for idx, lt in enumerate(lt_lst):
    name = parse_table_name(mt_lst[idx].name)
    lt.df_life_table().to_excel(excel_writer=name + '.xlsx', sheet_name=name, index=False,
    freeze_panes=(1, 1))
    ct_lst[idx].df_commutation_table().to_excel(excel_writer=name + '_comm' + '.xlsx',
    sheet_name=name, index=False, freeze_panes=(1, 1))

```

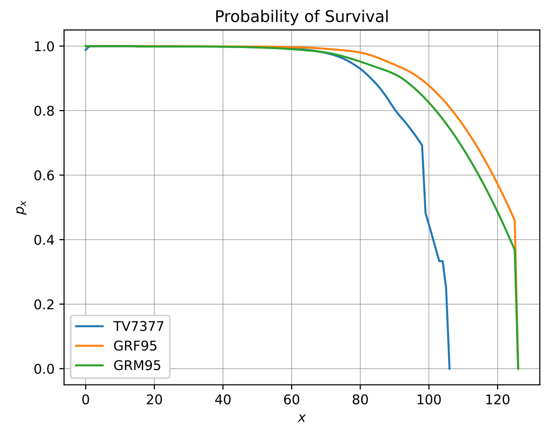
```python
# Question 3
'''
Plot lx
'''
fig, axes = plt.subplots()
for idx, lt in enumerate(lt_lst):
    ages = np.arange(0, lt.w + 2)
    plt.plot(ages, lt.lx, label=table_names[idx])
plt.xlabel(r'$x$')
plt.ylabel(r'$l_x$')
plt.title('Expected Number of Individuals Alive')
plt.grid(visible=True, which='both', axis='both', color='grey', linestyle='-', linewidth=.1)
plt.legend()
plt.savefig(this_py + 'lx' + '.eps', format='eps', dpi=3600)
plt.show()

'''
Plot dx
'''
fig, axes = plt.subplots()
for idx, lt in enumerate(lt_lst):
    ages = np.arange(0, lt.w + 1)
    plt.plot(ages, lt.dx, label=table_names[idx])
plt.xlabel(r'$x$')
plt.ylabel(r'$d_x$')
plt.title('Expected Number of Deaths')
plt.grid(visible=True, which='both', axis='both', color='grey', linestyle='-', linewidth=.1)
plt.legend()
plt.savefig(this_py + 'dx' +'.eps', format='eps', dpi=3600)
plt.show()

'''
Plot qx
'''
fig, axes = plt.subplots()
for idx, lt in enumerate(lt_lst):
    ages = np.arange(0, lt.w + 1)
    plt.plot(ages, lt.qx, label=table_names[idx])
plt.xlabel(r'$x$')
plt.ylabel(r'$q_x$')
plt.title('Mortality Rate')
plt.grid(visible=True, which='both', axis='both', color='grey', linestyle='-', linewidth=.1)
plt.legend()
plt.savefig(this_py + 'qx' +'.eps', format='eps', dpi=3600)
plt.show()

'''
Plot px
'''
fig, axes = plt.subplots()
for idx, lt in enumerate(lt_lst):
    ages = np.arange(0, lt.w + 1)
    plt.plot(ages, lt.px, label=table_names[idx])
plt.xlabel(r'$x$')
plt.ylabel(r'$p_x$')
plt.title('Probability of Survival')
plt.grid(visible=True, which='both', axis='both', color='grey', linestyle='-', linewidth=.1)
```

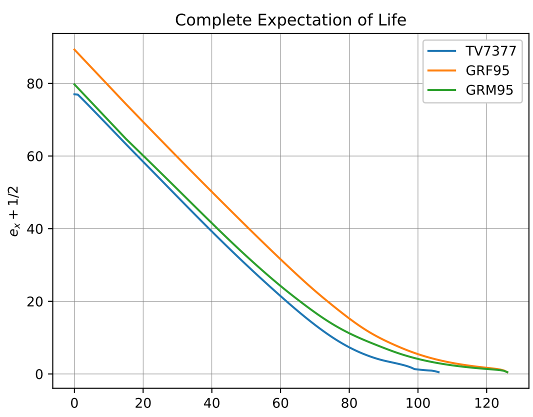
```
plt.legend()
plt.savefig(this_py + 'px' +'.eps', format='eps', dpi=3600)
plt.show()

'''
Plot ex
'''
fig, axes = plt.subplots()
for idx, lt in enumerate(lt_lst):
    ages = np.arange(0, lt.w + 1)
    plt.plot(ages, lt.ex, label=table_names[idx])
plt.xlabel(r'$x$')
plt.ylabel(r'${e}_{x}+1/2$')
plt.title('Complete Expectation of Life')
plt.grid(visible=True, which='both', axis='both', color='grey', linestyle='-', linewidth=.1)
plt.legend()
plt.savefig(this_py + 'ex' + '.eps', format='eps', dpi=3600)
plt.show()

'''
Plot ln(Dx)
'''
fig, axes = plt.subplots()
for idx, lt in enumerate(ct_lst):
    ages = np.arange(0, lt.w + 1)
    plt.plot(ages, np.log(lt.Dx), label=table_names[idx])
plt.xlabel(r'$x$')
plt.ylabel(r'$ln(D_x)$')
plt.title(r'ln(D_x)')
plt.grid(visible=True, which='both', axis='both', color='grey', linestyle='-', linewidth=.1)
plt.legend()
plt.savefig(this_py +'lnDx' + '.eps', format='eps', dpi=3600)
plt.show()

# Question 4
for idx, ct in enumerate(ct_lst):
    print(table_names[idx] + ": " + f'{round(1000 * ct.ax(x=55, m=1), 2):,}')
print()

# Question 5
for idx, ct in enumerate(ct_lst):
    print(table_names[idx]+":"+f'{round(1000 * ct.ax(x=55, m=1) / ct.naax(x=55, n=5, m=1), 2)
    :,}')

# Consult the values used each computation (Nx, Dx)
for idx, ct in enumerate(ct_lst):
    print(ct.msn)
    print()

# Consult the values used in the computation, only for TV7377
ct_lst[0].msn
```

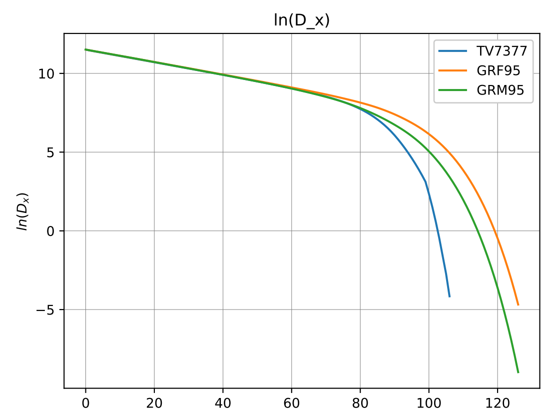












Question 4 a): $1000a_{55} = 1000\frac{N_{56}}{D_{55}}$

TV7377: 14,979.28 **GRF95:** 18,019.96 **GRM95:** 15,531.28

Question 4 b): $P\,\ddot{a}_{55:\overline{5}|} = 1000\frac{N_{56}}{D_{55}}$

TV7377: 3,272.3 **GRF95:** 3,910.88 **GRM95:** 3,398.66

Question 5: $1000a_{55:\overline{10}|} = 1000\frac{N_{56}-N_{66}}{D_{55}}$

TV7377: 3,272.3 **GRF95:** 3,910.88 **GRM95:** 3,398.66

8.3 Life Insurance

Example 3

Consider a Pure Endowment insurance with duration 10 years, if someone 55 years old today, subscribes 1000€, considering $i = 4\%/\text{year}$. Considering the mortatily tables TV7377, GRF95 and GRM95:

1. Determine the net single premium (risk single premium).
2. Determine the annual premium paid during 5 years if the insured is alive.
3. Evaluate the price of contract, including the refund of all premiums paid at the end of the year of death and at the end of the term.

```
from lifeactuary import mortality_table as mtable, commutation_table as ct, read_soa_table_xml
    as rst
import numpy as np

table_names = ['TV7377', 'GRF95', 'GRM95']
interest_rate = 4
mt_lst = [rst.SoaTable('soa_tables/' + name + '.xml') for name in table_names]
lt_lst = [mtable.MortalityTable(mt=mt.table_qx) for mt in mt_lst]
ct_lst = [ct.CommutationFunctions(i=interest_rate, g=0, mt=mt.table_qx) for mt in mt_lst]

# General Information
x = 55
capital = 1000
term = 10
term_annuity = 5
# pure endowment
pureEndow = [ct.nEx(x=x, n=term) for ct in ct_lst]
# temporary annuity due
tad = [ct.naax(x=x, n=term_annuity, m=1) for ct in ct_lst]

### Question (a)
print('\nnet single premium')
for idx, ct in enumerate(ct_lst):
    print(table_names[idx] + ": " + f'{round(capital * pureEndow[idx], 5):,}')

### Question (b)
print('\nlevel premium')
for idx, ct in enumerate(ct_lst):
    print(table_names[idx] + ":" + f'{round(capital * pureEndow[idx] / tad[idx], 5):,}')

# show the annuities
print('\nannuities')
for idx, ct in enumerate(ct_lst):
    print(table_names[idx] + ":" + f'{round(tad[idx], 5):,}')

### Question (c)

## Refund of Net Single Premium
print('\nSingle Net Risk Premium Refund at End of the Year of Death')
termLifeInsurance = [ct.nAx(x=x, n=term) for ct in ct_lst]
pureEndow_refund = [ct.nEx(x=x, n=term) / (1 - ct.nAx(x=x, n=term)) for ct in ct_lst]
```

```python

print('\nTerm Life Insurance')
for idx, ct in enumerate(ct_lst):
    print(table_names[idx] + ":" + f'{round(termLifeInsurance[idx], 5):,}')

print('\nSingle Net Premium Refund Cost at End of the Year of Death')
for idx, ct in enumerate(ct_lst):
    print(table_names[idx] + ":" + f'{round(capital * pureEndow[idx] / (1 - termLifeInsurance[idx]), 5):,}')

print('Refund Cost at End of the Year of Death')
for idx, ct in enumerate(ct_lst):
    print(table_names[idx] + ":" + f'{round(capital * (pureEndow_refund[idx] - pureEndow[idx]), 5):,}')

print('\nSingle Net Risk Premium Refund at End of the Term')
pureEndow_refund_eot = [ct.nEx(x=x, n=term) / (1 - (1 + interest_rate / 100) ** (-term) + ct.nEx(x=x, n=term))
    for ct in ct_lst]

for idx, ct in enumerate(ct_lst):
    print(table_names[idx] + ":" + f'{round(capital * pureEndow_refund_eot[idx], 5):,}')

print('Refund Cost at End of the the Term')
for idx, ct in enumerate(ct_lst):
    print(table_names[idx] + ":" + f'{round(capital * (pureEndow_refund_eot[idx] - pureEndow[idx]), 5):,}')


## Refund of Net Level Premiums

print('\nLeveled Net Risk Premium Refund at End of the Year of Death')
tli_increasing = [ct.nIAx(x=x, n=term_annuity) for ct in ct_lst]
tli_deferred = [ct.t_nAx(x=x, n=term - term_annuity, defer=term_annuity) for ct in ct_lst]
pureEndow_leveled_refund = [
    pureEndow[idx_ct] / (tad[idx_ct] - tli_increasing[idx_ct] - term_annuity * tli_deferred[idx_ct])
    for idx_ct, ct in enumerate(ct_lst)]

for idx, ct in enumerate(ct_lst):
    print(table_names[idx] + ":" + f'{round(capital * pureEndow_leveled_refund[idx], 5):,}')
```

Solutions:

Question 1:

$$PP = 1000\,{}_{10}E_{55}$$

TV7377: 624.36092 **GRF95:** 653.67485 **GRM95:** 615.65987

Question 2:

$$P\ddot{a}_{55} = 1000\,\frac{{}_{10}E_{55}}{\ddot{a}_{55:\overline{10}|}}$$

TV7377: 136.39495 **GRF95:** 141.86738 **GRM95:** 134.72276

Question 3:

Single Premium with Refund paid at the end of the Year of Death

$$P = 1000\,\frac{{}_{10}E_{55}}{1 - A^{1}_{55:\overline{10}|}}$$

TV7377: 664.2747 **GRF95:** 670.91998 **GRM95:** 662.20316

Single Premium with Refund paid at the end of the contract

$$P = 1000\,\frac{{}_{10}E_{55}}{1 - v^{10}\ {}_{10}q_{55}}$$

TV7377: 658.0555 **GRF95:** 668.30356 **GRM95:** 654.89063

Level Premium with Refund paid at the end of the year of death

$$P = 1000\,\frac{{}_{10}E_{55}}{\ddot{a}_{55:\overline{10}|} - (IA)_{55:\overline{10}|}}$$

TV7377: 144.14453 **GRF95:** 145.18891 **GRM95:** 143.8195

Example 4

For the contract in Example 3.2, estimate the net premium reserves until the end of the contract, export the estimates to and EXcel file and plot the evolution of the reserves in a graph.

```python

## Net premium reserves path

l0 = 1000
reserves_dict = {'table': [], 'x': [], 'insurer': [], 'insured': [], 'reserve': []}
fund_dict = {'lx': [], 'claim': [], 'premium': [], 'fund': []}

# Expected reserves value, that is, considering the survivorship of the group
expected_reserve_dict = {'insurer_exp': [], 'insured_exp': [], 'reserve_exp': []}
ages = range(x, x + term + 1)
print('\n\n Net Premium reserves \n\n')
for idx_clt, clt in enumerate(ct_lst):
    premium_unit = pureEndow[idx_clt]
    premium_capital = capital * premium_unit
    premium_unit_leveled = premium_unit / tad[idx_clt]
    premium_leveled = premium_unit_leveled * capital
    for age in ages:
        # reserves
        reserves_dict['table'].append(table_names[idx_clt])
        reserves_dict['x'].append(age)
        insurer_liability = clt.nEx(x=age, n=term - (age - x)) * \
                            capital
        reserves_dict['insurer'].append(insurer_liability)
        tad2 = clt.naax(x=age, n=term_annuity - (age - x))
        insured_liability = premium_leveled * tad2
        reserves_dict['insured'].append(insured_liability)
        reserve = insurer_liability - insured_liability
        reserves_dict['reserve'].append(reserve)

        prob_survival = clt.npx(x=x, n=age - x)
        lx = l0 * prob_survival
        expected_reserve_dict['insurer_exp'].append(insurer_liability*lx)
        expected_reserve_dict['insured_exp'].append(insured_liability*lx)
        expected_reserve_dict['reserve_exp'].append(reserve*prob_survival*lx)

        # fund
        fund_dict['lx'].append(lx)
        qx_1 = clt.nqx(x=age, n=1)
        claim = 0
        if age == x + term:
            claim = capital * lx
        fund_dict['claim'].append(claim)
        premium = 0
        if tad2 > 0:
            premium = premium_leveled*lx
        fund_dict['premium'].append(premium)
        if age == x:
            fund = lx * premium_leveled
        else:
            fund = fund_dict['fund'][-1] * (1 + interest_rate / 100) - claim + premium
        fund_dict['fund'].append(fund)
```

```python

reserves_df = pd.DataFrame(reserves_dict)
expected_reserve_df = pd.DataFrame(expected_reserve_dict)
fund_df = pd.DataFrame(fund_dict)
name = 'pureEndowment_55_1'
reserves_df.to_excel(excel_writer=name + '_netReserves' + '.xlsx', sheet_name=name, index=
    False, freeze_panes=(1, 1))

'''
plot the reserves
'''
for idx_clt, clt in enumerate(ct_lst):
    plt.plot(ages, reserves_df.loc[reserves_df['table'] == table_names[idx_clt]]['reserve'],
             label=table_names[idx_clt])

plt.xlabel(r'$x$')
plt.ylabel('Reserves')
plt.title('Net Premium Reserves Pure Endowment')
plt.grid(visible=True, which='both', axis='both', color='grey', linestyle='-', linewidth=.1)
plt.legend()
plt.show()

# save the graph
plt.savefig(this_py + '.eps', format='eps', dpi=3600)
```