Solving Actuarial Math with Python

Terence Lim

Jan 20, 2024

CONTENTS

1 Actuarial Python **5**
- 1.1 Installation . . . 5
- 1.2 Overview . . . 5
- 1.3 License . . . 6
- 1.4 Methods . . . 6
- 1.5 Examples . . . 7

2 Interest Theory **9**
- 2.1 Interest rates . . . 9
- 2.2 Methods . . . 10
- 2.3 Examples . . . 11

3 Life Contingent Risks **15**
- 3.1 Probability . . . 15
- 3.2 Portfolio Percentile . . . 16
- 3.3 Methods . . . 17
- 3.4 Examples . . . 17

4 Survival Models **21**
- 4.1 Lifetime distribution . . . 21
- 4.2 Survival function . . . 21
- 4.3 Force of mortality . . . 22
- 4.4 Actuarial notation . . . 22
- 4.5 Methods . . . 23
- 4.6 Examples . . . 24

5 Expected Future Lifetimes **27**
- 5.1 Complete expectation of life . . . 27
- 5.2 Curtate expectation of life . . . 27
- 5.3 Temporary expectation of life . . . 28
- 5.4 Methods . . . 28
- 5.5 Examples . . . 28

6 Fractional Ages **33**
- 6.1 Uniform distribution of deaths . . . 33
- 6.2 Constant force of mortality . . . 34
- 6.3 Methods . . . 34
- 6.4 Examples . . . 35

7 Insurance **39**
- 7.1 Present value of life insurance r.v. Z . . . 39

7.2 Whole life insurance . . . 39
7.3 Term insurance . . . 40
7.4 Deferred insurance . . . 40
7.5 Endowment insurance . . . 40
7.6 Pure endowment . . . 41
7.7 Variances . . . 41
7.8 Varying insurance . . . 42
7.9 Probabilities and percentiles . . . 43
7.10 Methods . . . 43
7.11 Examples . . . 44

8 Annuities **53**
8.1 Present value of life annuity r.v. Y . . . 53
8.2 Whole life annuity . . . 53
8.3 Temporary annuity . . . 54
8.4 Deferred whole life annuity . . . 54
8.5 Certain and life annuity . . . 54
8.6 Life insurance twin . . . 55
8.7 Variances . . . 55
8.8 Immediate life annuity . . . 56
8.9 Varying life annuities . . . 56
8.10 Probabilities and percentiles . . . 57
8.11 Methods . . . 57
8.12 Examples . . . 59

9 Premiums **63**
9.1 Present value of loss at issue r.v. ${}_0L$ . . . 63
9.2 Equivalence principle . . . 64
9.3 Net premium . . . 64
9.4 Gross premium . . . 65
9.5 Portfolio Percentile Premium . . . 66
9.6 Methods . . . 66
9.7 Examples . . . 67

10 Policy Values **71**
10.1 Net policy value . . . 71
10.2 Gross policy value . . . 72
10.3 Variance of future loss . . . 72
10.4 Expense reserve . . . 73
10.5 Probabilities and Percentiles . . . 74
10.6 Methods . . . 74
10.7 Examples . . . 76

11 Reserves **81**
11.1 Recursion . . . 81
11.2 Interim reserves . . . 81
11.3 Modified reserves . . . 82
11.4 Methods . . . 82
11.5 Examples . . . 83

12 Recursion **87**
12.1 Chain rule . . . 87
12.2 Expected future lifetime . . . 87
12.3 Life insurance . . . 88
12.4 Life annuities . . . 88

12.5 Methods . . . 89
12.6 Examples . . . 90

13 Life Table **97**
13.1 Methods . . . 97
13.2 Examples . . . 98

14 SULT **103**
14.1 Standard ultimate life table . . . 103
14.2 Pure endowment . . . 103
14.3 Term life insurance . . . 103
14.4 Methods . . . 104
14.5 Examples . . . 104

15 Select Life Table **115**
15.1 Select and ultimate life model . . . 115
15.2 Methods . . . 116
15.3 Examples . . . 117

16 Mortality Laws **123**
16.1 Uniform distribution . . . 123
16.2 Beta distribution . . . 124
16.3 Gompertz's Law . . . 124
16.4 Makeham's Law . . . 124
16.5 Methods . . . 125
16.6 Examples . . . 126

17 Constant Force of Mortality **129**
17.1 Expected future lifetime . . . 129
17.2 Pure endowment . . . 129
17.3 Life insurance . . . 130
17.4 Life annuities . . . 130
17.5 Net future loss . . . 130
17.6 Methods . . . 130
17.7 Examples . . . 131

18 Extra Risk **135**
18.1 Age rating . . . 135
18.2 Multiple of mortality rate . . . 135
18.3 Force of mortality . . . 135
18.4 Methods . . . 136
18.5 Examples . . . 136

19 1/M'thly **139**
19.1 Life Insurance . . . 139
19.2 Life Annuity . . . 140
19.3 Life Insurance Twin . . . 140
19.4 Methods . . . 140
19.5 Examples . . . 142

20 UDD M'thly **145**
20.1 Life insurance . . . 145
20.2 Continuous Life Insurance . . . 145
20.3 Interest functions . . . 146
20.4 Life annuities . . . 146

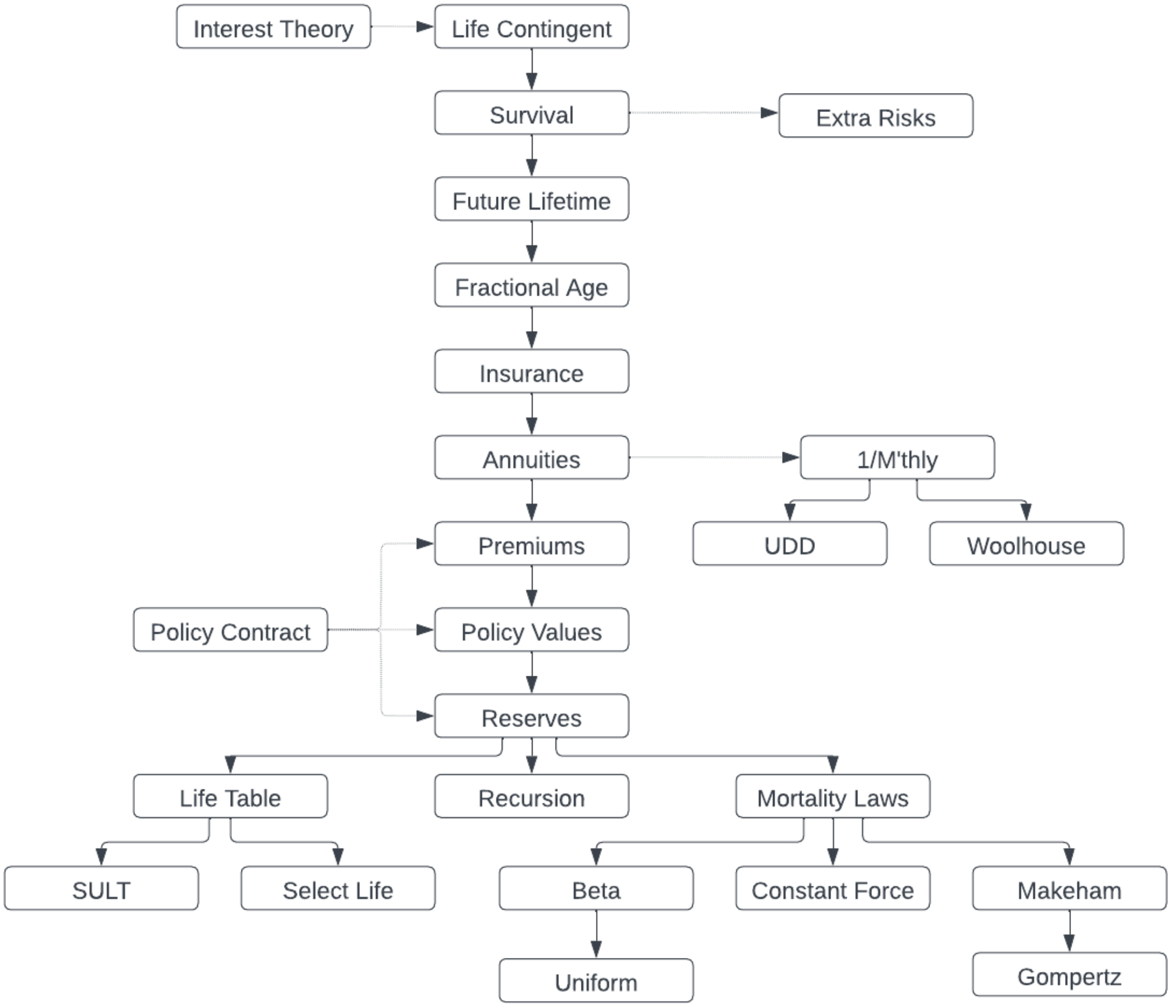
20.5 Methods . . . 146
20.6 Examples . . . 147

21 Woolhouse M'thly 151
21.1 Life Annuities . . . 151
21.2 Methods . . . 151
21.3 Examples . . . 152

22 FAM-L Exam Solutions and Hints 155
22.1 1 Tables . . . 157
22.2 2 Survival models . . . 158
22.3 3 Life tables and selection . . . 162
22.4 4 Insurance benefits . . . 170
22.5 5 Annuities . . . 181
22.6 6 Premium Calculation . . . 186
22.7 7 Policy Values . . . 214

actuarialmath – Solve Life Contingent Risks with Python

This Python package implements fundamental methods for modeling life contingent risks, and closely follows the coverage of traditional topics in actuarial exams and standard texts such as the "Fundamentals of Actuarial Math - Long-term" exam syllabus by the Society of Actuaries, and "Actuarial Mathematics for Life Contingent Risks" by Dickson, Hardy and Waters. The actuarial concepts, and corresponding Python classes, are introduced and modeled hierarchically.


Interest Theory
Life Contingent
Survival
Extra Risks
Future Lifetime
Fractional Age
Insurance
Annuities
1/M'thly
UDD
Woolhouse
Premiums
Policy Contract
Policy Values
Reserves
Life Table
Recursion
Mortality Laws
SULT
Select Life
Beta
Constant Force
Makeham
Uniform
Gompertz

Quick Start

1. `pip install actuarialmath`
    - also requires `numpy`, `scipy`, `matplotlib` and `pandas`.
2. Start Python (version >= 3.10) or Jupyter-notebook
    - Select a suitable subclass to initialize with your actuarial assumptions, such as `MortalityLaws` (or a special law like `ConstantForce`), `LifeTable`, `SULT`, `SelectLife` or `Recursion`.
    - Call appropriate methods to compute intermediate or final results, or to `solve` parameter values implicitly.
    - Adjust answers with `ExtraRisk` or `Mthly` (or its `UDD` or `Woolhouse`) classes

Examples

SOA FAM-L sample question 5.7:

Given $A_{35} = 0.188$, $A_{65} = 0.498$, $S_{35}(30) = 0.883$, calculate the EPV of a temporary annuity $\ddot{a}^{(2)}_{35:\overline{30|}}$ paid half-yearly using the Woolhouse approximation.

```
from actuarialmath import Recursion, Woolhouse
# initialize Recursion class with actuarial inputs
life = Recursion().set_interest(i=0.04)\
                  .set_A(0.188, x=35)\
                  .set_A(0.498, x=65)\
                  .set_p(0.883, x=35, t=30)
# modfy the standard results with Woolhouse mthly approximation
mthly = Woolhouse(m=2, life=life, three_term=False)
# compute the desired temporary annuity value
print(1000 * mthly.temporary_annuity(35, t=30)) #   solution = 17376.7
```

SOA FAM-L sample question 7.20:

For a fully discrete whole life insurance of 1000 on (35), you are given

- First year expenses are 30% of the gross premium plus 300
- Renewal expenses are 4% of the gross premium plus 30
- All expenses are incurred at the beginning of the policy year
- Gross premiums are calculated using the equivalence principle
- The gross premium policy value at the end of the first policy year is R
- Using the Full Preliminary Term Method, the modified reserve at the end of the first policy year is S
- Mortality follows the Standard Ultimate Life Table
- $i = 0.05$

Calculate R – S

```
from actuarialmath import SULT, Contract
life = SULT()
# compute the required FPT policy value
S = life.FPT_policy_value(35, t=1, b=1000)  # is always 0 in year 1!
# input the given policy contract terms
contract = Contract(benefit=1000,
                    initial_premium=.3,
                    initial_policy=300,
                    renewal_premium=.04,
                    renewal_policy=30)
# compute gross premium using the equivalence principle
G = life.gross_premium(A=life.whole_life_insurance(35), **contract.premium_terms)
# compute the required policy value
R = life.gross_policy_value(35, t=1, contract=contract.set_contract(premium=G))
print(R-S)    # solution = -277.19
```

Resources

1. Jupyter notebook or run in Colab, to solve all sample SOA FAM-L exam questions.
2. User Guide, or download pdf
3. API reference
4. Github repo and issues

Sources

- SOA FAM-L Sample Questions: copy retrieved Aug 2022
- SOA FAM-L Sample Solutions: copy retrieved Aug 2022
- Actuarial Mathematics for Life Contingent Risks, by David Dickson, Mary Hardy and Howard Waters, published by Cambridge University Press.

Contact

Github: https://terence-lim.github.io

Enjoy!

Terence Lim PhD ASA CFA CPA CFP CAIA

CHAPTER

ONE

ACTUARIAL PYTHON

The `actuarialmath` package is written in and requires Python (currently: version 3.10). Though the comparable R language possesses other desirable qualities, object-oriented programming is more straightforward in Python: since our sequence of actuarial concepts logically build upon each other, they are naturally developed as a hieararchy of Python classes with inherited methods and properties.

1.1 Installation

Install either by using pip:

- `pip install actuarialmath`

or cloning from github:

- `git clone https://github.com/terence-lim/actuarialmath.git`

1.2 Overview

Each section of this document introduces a class, along with the actuarial concepts it implements, arranged logically in three groups. To use the package, a suitable subclass should first be selected from the last group to load the given actuarial assumptions. Then the appropriate computational methods can be called, which may be inherited from the other general classes or make use of any shortcut formulas that can be obtained from the specific survival model assumed.

1. Implement general actuarial methods
    - Basic interest theory and probability laws
    - Survival functions, expected future lifetimes and fractional ages
    - Insurance, annuity, premiums, policy values, and reserves calculations
2. Adjust results for
    - Extra mortality risks
    - 1/mthly payment frequency using UDD or Woolhouse approaches
3. Specify survival models and assumptions, and implement associated shortcut formulas
    - Recursion inputs
    - Life table, select life table, or standard ultimate life table
    - Mortality laws, such as constant force of maturity, beta and uniform distributions, or Makeham's and Gompertz's laws

1.3 License

MIT License

Copyright (c) 2022-2023 Terence Lim

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.4 Methods

The `Actuarial` base class provides some common helpful utility functions and definitions of constants, that are needed by other classes in the package.

```python
from actuarialmath import Actuarial
import math
import describe
describe.methods(Actuarial)
```

```
class Actuarial - Define constants and common utility functions

    Constants:
      VARIANCE : select variance as the statistical moment to calculate

      WHOLE : indicates that term of insurance or annuity is Whole Life

    Methods:
    --------

    solve(fun, target, grid, mad):
      Solve root, or parameter that minimizes absolute value, of a function

    add_term(t, n):
      Add two terms, either term may be Whole Life

    max_term(x, t, u):
      Decrease term t if adding deferral period u to (x) exceeds maxage

    isclose(r, target, abs_tol):
      Is close to zero or target value
```

1.5 Examples

The `solve()` method is called to impute the value of a parameter such that its function output value is equal to a specified target value, by either returning the zero *root* (set argument `mad = False`, by default) or minimizing the absolute difference (set `mad = True`). As a simple example, to solve for the median of the exponential cumulative distribution:

```
Actuarial.solve(fun=lambda x: 1 - math.exp(-x), target=0.5, grid=[0, 2])
```

```
0.6931471805599453
```

The `add_term` method adds two terms, while handling the case where either may not be a fixed term, i.e. they may be *whole life* and indicated with the constant `WHOLE`. The `max_term` method trims the value of a term *t*, such that its sum with age *x* and deferral period *u* is no larger than the maximum age

```
actuarial = Actuarial()

def as_term(t): return "WHOLE_LIFE" if t == Actuarial.WHOLE else t

for a,b in [(3, Actuarial.WHOLE), (Actuarial.WHOLE, -1), (3, 2), (3, -1)]:
    print(f"{as_term(a)} + {as_term(b)} =", as_term(actuarial.add_term(a, b)))
print()
for t in [10, 50, Actuarial.WHOLE]:
    print(as_term(t), '->', actuarial.max_term(x=65, t=t, u=20))
```

```
3 + WHOLE_LIFE = WHOLE_LIFE
WHOLE_LIFE + -1 = WHOLE_LIFE
3 + 2 = 5
3 + -1 = 2

10 -> 10
50 -> 15
WHOLE_LIFE -> 15
```

CHAPTER TWO

INTEREST THEORY

Interest theory functions, that are in common actuarial and financial use, are reviewed. Interest rates are generally assumed to be fixed and constant.

2.1 Interest rates

i is the amount earned on $1 after one year

- effective annual *interest rate*

$i^{(m)}$ is the nominal interest rate compounded *m* times per year

- it is numerically equal to m times the effective interest rate over 1/m'th of a year.

$d = \frac{i}{1+i}$

- annual *discount rate* of interest

$d^{(m)}$ is the nominal discount rate compounded *m* times per year

- it is numerically equal to m times the discount rate over 1/m'th of a year.

$v = \frac{1}{1+i}$

- annual *discount factor*

$\delta = \log(1+i)$

- *continuosly-compounded rate* of interest, or *force of interest per year*

Relationships between interest rates

$$\begin{aligned} v^t &= e^{-\delta t} \\ &= (1-d)^t \\ &= (1+i)^{-t} \\ &= (1-\frac{d^{(m)}}{m})^{mt} \\ &= (1+\frac{i^{(m)}}{m})^{-mt} \end{aligned}$$

Doubling the force of interest

Note that doubling the force of interest is not the same as doubling the rate of interest.

$^2\delta \leftarrow 2\delta$

$^2i \leftarrow 2i + i^2$

$^2d \leftarrow 2d - d^2$

$^2v \leftarrow v^2$

Annuity certain

Present value of an annuity certain that pays at a rate of \$1 per year for n years

$$\ddot{a}_{\overline{n|}} = \frac{1 - v^n}{d}$$

- Annuity certain due: pays \$1 at the beginning of the year

$$a_{\overline{n|}} = \frac{1 - v^n}{i} = \ddot{a}_{\overline{n+1|}} - 1$$

- Immediate annuity certain: pays \$1 at the end of the year

$$\bar{a}_{\overline{n|}} = \frac{1 - v^n}{\delta}$$

- Continuous annuity certain: pays at a rate of \$1 per year continuously.

2.2 Methods

The `Interest` class implements methods to convert between nominal, discount, continuously-compounded and 1/m'thy rates of interest, and compute the value of an annuity certain.

```python
from actuarialmath import Interest
import describe
describe.methods(Interest)
```

```
class Interest - Store an assumed interest rate, and compute interest rate␣
↪functions

    Args:
      i : assumed annual interest rate
      d : or annual discount rate
      v : or annual discount factor
      delta : or continuously compounded interest rate
      v_t : or discount rate as a function of time
      i_m : or m-thly interest rate
      d_m : or m-thly discount rate
      m : m'thly frequency, if i_m or d_m are specified

    Methods:
    --------

    annuity(t, m, due):
      Compute value of the annuity certain factor

    mthly(m, i, d, i_m, d_m):
      Convert to or from m'thly interest rates

    double_force(i, delta, d, v):
      Double the force of interest
```

2.3 Examples

The `Interest` class can be initialized with an assumed annual interest rate expressed in any one of the forms `i, d, v, delta, i_m, d_m` (where the latter two forms also require the argument `m` to specify the number of times compounded in a year). Then that rate can be retrieved in any other annual form as an attribute.

```
interest = Interest(i=0.05)
print(interest.d, interest.v, interest.delta, interest.i)
```

```
0.047619047619047616 0.9523809523809523 0.04879016416943205 0.05
```

The `mthly` static method converts between annual-pay and m-thly pay interest rates in any form.

```
delta = 0.05
i = Interest(delta=delta).i      # convert to annual interest rate
d = Interest(delta=delta).d      # convert to annual discount rate
i_m = Interest.mthly(i=i, m=12)  # convert to annual interest rate monthly-pay
d_m = Interest.mthly(d=d, m=12)  # convert to annual discount rate monthly-pay
print('Continuously-compounded annual rate of interest:', delta)
print('  Convert to annual interest rate:              ', i, Interest.mthly(i_m=i_m,␣
 ↪m=12))
print('  Convert to annual discount rate:              ', d, Interest.mthly(d_m=d_m,␣
 ↪m=12))
print('  Convert to annual interest rate (monthly-pay):', i_m)
print('  Convert to annual discount rate (monthly-pay):', d_m)
```

```
Continuously-compounded annual rate of interest: 0.05
  Convert to annual interest rate:               0.05127109637602412 0.
 ↪051271096376023007
  Convert to annual discount rate:               0.04877057549928606 0.
 ↪04877057549928587
  Convert to annual interest rate (monthly-pay): 0.05010431149342143
  Convert to annual discount rate (monthly-pay): 0.049895977858680496
```

The `annuity` class method computes the present value of an annuity that pays at the rate of $1 per year continuously or discrete (due, immediate, or m'thly)

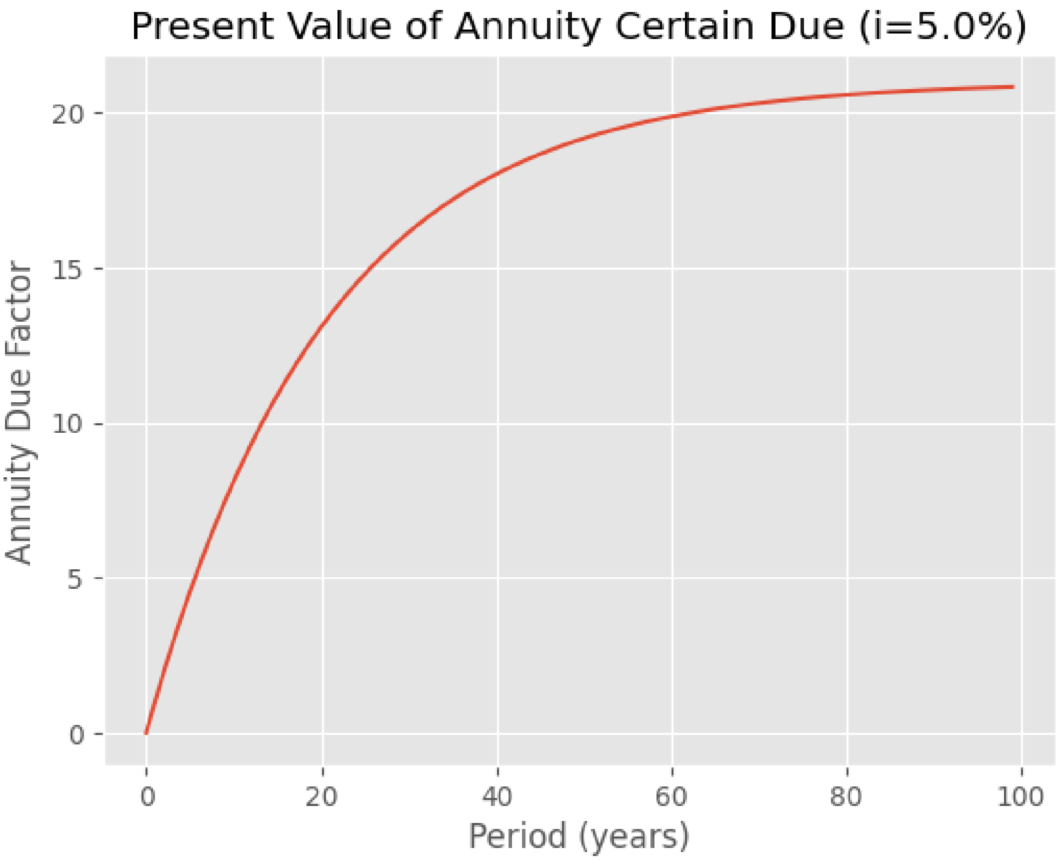
```
import matplotlib.pyplot as plt
t = range(100)
plt.plot(t, [interest.annuity(t=s, due=True) for s in t])
plt.title(f'Present Value of Annuity Certain Due (i={interest.i * 100}%)')
plt.xlabel('Period (years)')
plt.ylabel('Annuity Due Factor')
print("Annuity Factors By Payment Frequency (perpetual, i=5%):")
print('1. Immediate:                 ', interest.annuity(t=Interest.WHOLE, due=False))
print('2. Immediate (quarterly pay): ', interest.annuity(t=Interest.WHOLE, m=4,␣
 ↪due=False))
print('3. Continuous:                ', interest.annuity(t=Interest.WHOLE, m=0))
print('4. Due (quarterly pay):       ', interest.annuity(t=Interest.WHOLE, m=4,␣
 ↪due=True))
print('5. Due:                       ', interest.annuity(t=Interest.WHOLE, due=True))
```

```
Annuity Factors By Payment Frequency (perpetual, i=5%):
1. Immediate:                  19.999999999999982
```

(continues on next page)

(continued from previous page)

```
2. Immediate (quarterly pay):  20.371188429095998
3. Continuous:                 20.49593431428785
4. Due (quarterly pay):        20.621188429096076
5. Due:                        20.999999999999975
```



The `double_force` static method takes an interest rate in any form, and converts to the same form after doubling the force of interest.

```
print("Example: double the force of interest i=0.05")
i = 0.05
d = Interest(i=i).d                # convert interest rate to discount rate
print('i:', i, 'd:', d)
i2 = Interest.double_force(i=i)    # interest rate after doubling force
d2 = Interest.double_force(d=d)    # discount rate after doubling force
print('i:', round(i2, 6), round(Interest(d=d2).i, 6))
print('d:', round(d2, 6), round(Interest(i=i2).d, 6))
```

```
Example: double the force of interest i=0.05
i: 0.05 d: 0.047619047619047616
i: 0.1025 0.1025
d: 0.092971 0.092971
```

SOA Question 3.10:

A group of 100 people start a Scissor Usage Support Group. The rate at which members enter and leave the group is dependent on whether they are right-handed or left-handed. You are given the following:

- The initial membership is made up of 75% left-handed members (L) and 25% right-handed members (R)
- After the group initially forms, 35 new (L) and 15 new (R) join the group at the start of each subsequent year
- Members leave the group only at the end of each year
- q^L = 0.25 for all years
- q^R = 0.50 for all years Calculate the proportion of the Scissor Usage Support Group's expected membership that is left-handed at the start of the group's6 th year, before any new members join for that year.

```
print("SOA Question 3.10:  (C) 0.86")
interest = Interest(v=0.75)
L = 35 * interest.annuity(t=4, due=False) + 75 * interest.v_t(t=5)
interest = Interest(v=0.5)
R = 15 * interest.annuity(t=4, due=False) + 25 * interest.v_t(t=5)
print(L / (L + R))
```

```
SOA Question 3.10:  (C) 0.86
0.8578442833761983
```

CHAPTER

THREE

LIFE CONTINGENT RISKS

Life contingencies use probability models as well as interest rate functions. By modeling future lifetimes as random variables, probabilities of survival or death can be calculated. This section reviews essential probability relationships and the moments of common distributions.

3.1 Probability

The cumulative distribution function of a r.v. X is the probability $F(x) = Pr(X \leq x)$. For a continuous r.v., $f(x) = \frac{d}{dx}F(x)$ is the probability density function. For a discrete r.v. $p(x) = Pr(X = x)$ is the probability mass function.

$E(x) = \int_{-\infty}^{\infty} x f(x) dx$ when continuous, or $\sum_x x f(x)$ when discrete

- is the mean, or first moment of X.

$Var(x) = E[(X - E[x])^2]$

- is the variance, or second central moment of X, often denoted σ^2

Linearity of expectations

$E(a + bX + cY) = a + bE(X) + cE(Y)$

- the mean of the sum of random variables is the sum of the means

Variances

$Var(aX + bY) = a^2 \, Var(X) + b^2 \, Var(Y) + 2 \, a \, b \, Cov(X, Y)$

- is the formula to compute the variance of the weighted sum of two random variables

$Cov(X, Y) = E[XY] - E[X] \cdot E[Y]$

- is the formula to compute the covariance between two random variables

Bernoulli:

The Bernoulli is a discrete distribution that takes one of two values $Y \in \{a, \ b\}$ with probabilities $(p, \ 1 - p)$

$E[Y] = a \, p + b \, (1 - p)$

- mean of Bernoulli distribution

$Var[Y] = (a - b)^2 \, p \, (1 - p)$

- Bernoulli shortcut formula for the variance

Binomial:

A binomial r.v. measures the total number of successes from N independent 0-1 Bernoulli r.v., where each has a probability of success p.

$E[Y] = N\ p$

- mean of Binomial distribution

$Var[Y] = N\ p\ (1-p)$

- variance of Binomial distribution

Mixture

A mixture distribution is a random variable Y whose distribution function can be expressed as a weighted average of the distribution functions of n random variables $Y_1, \cdots, Y_n$. A common case is when Y is a mixture of two Binomial distributions $(p',\ N)$, where $p' \in (p_1,\ p_2)$ with respective probabilities $(p,\ 1-p)$: a value is first drawn from a Bernoulli random variable w.p. p, then depending on its value, a final value is then drawn from the selected one of the two component Binomial distributions.

$E[Y] = p\ N\ p_1 + (1-p)\ N\ p_2$

- mean of mixture of two Binomial distributions

$Var[Y] = E[Y^2] - E[Y]^2 = E[Var(Y \mid p') + E(Y \mid p')^2] - E[Y]^2$

- variance of mixture of two Binomial distributions

Conditional Variance

$Var[Y] = Var(E[Y \mid p']) + E[Var(Y \mid p')]$

- alternative calculation of the variance of a mixture by using conditional variance formula, where the first term can be easily calculated using the Bernoulli shortcut, since $E[Y \mid p']$ is a Bernoulli random variable that takes a value of either $E[Y \mid p_1]$ or $E[Y \mid p_2]$ with respective probabilities $(p, 1-p)$

3.2 Portfolio Percentile

Normal Approximation

A portfolio, Y, which is the sum of N iid random variables each with mean μ and variance σ^2, has a normal distribution with

- mean $E[Y] = N\ \mu$ and
- variance $Var[Y] = N\ \sigma^2$

Percentiles

Percentiles are essentially an inverse function of the cumulative probability distribution. If $F(y)$ is the cdf for Y, then the p'th quantile (or $100p$'th percentile) is a number y_p such that $F(y_p) = Pr(Y \le y_p) = p$.

$Y_p = E[Y] + z_p\ \sqrt{Var[Y]}$

- value of Y at percentile p when Y is normally distributed, where z_p is the inverse standard normal function $z_p = \Phi^{-1}(p)$

3.3 Methods

The `Life` class implements methods for computing moments and probability distributions of random variables.

```
import math
from actuarialmath import Life
import describe
describe.methods(Life)
```

```
class Life - Compute moments and probabilities

    Methods:
    --------

    variance(a, b, var_a, var_b, cov_ab):
      Variance of weighted sum of two r.v.

    covariance(a, b, ab):
      Covariance of two r.v.

    bernoulli(p, a, b, variance):
      Mean or variance of bernoulli r.v. with values {a, b}

    binomial(p, N, variance):
      Mean or variance of binomial r.v.

    mixture(p, p1, p2, N, variance):
      Mean or variance of binomial mixture

    conditional_variance(p, p1, p2, N):
      Conditional variance formula

    portfolio_percentile(mean, variance, prob, N):
      Probability percentile of the sum of N iid r.v.'s

    set_interest(interest):
      Set interest rate, which can be given in any form

    quantiles_frame(quantiles):
      Display selected quantile values from Normal distribution table
```

3.4 Examples

The `variance` method computes the variance of $aX + bY$ by plugging in the required inputs into the formula $a^2\ Var(X) + b^2\ Var(Y) + 2\ a\ b\ Cov(X, Y)$. Similarly, the `covariance` method computes the covariance of X and Y by pluggin the required inputs into the formula $E[XY] - E[X] \cdot E[Y]$

```
Life.variance(a=2, b=3, var_a=4, var_b=5, cov_ab=6)
```

```
133
```

```
Life.covariance(a=2, b=3, ab=8)
```

```
2
```

The `bernoulli` and `binomial` methods compute the mean or variance (by setting the argument `variance=True`) of the respective probability distributions

```
print(Life.bernoulli(p=0.5, a=0, b=1), Life.bernoulli(p=0.5, a=0, b=1, variance=True))
```

```
0.5 0.25
```

```
print(Life.binomial(p=0.5, N=100), Life.binomial(p=0.5, N=100, variance=True))
```

```
50.0 25.0
```

For a mixture of two binomial distribution, the `mixture` method computes the mean or variance, while the `conditional_variance` method computes the variance using the conditional variance formula.

```
print(Life.mixture(p=0.5, p1=0.3, p2=0.4, N=100),
      Life.mixture(p=0.5, p1=0.3, p2=0.4, N=100, variance=True))
```

```
35.0 47.5
```

The sum of N iid r.v. with identical mean and variance can be approximated by a normal distribution that is computed by the `portfolio_cdf` method, or whose $100p$'th percentile is computed by the `portfolio_percentile` method.

```
Life.portfolio_cdf(mean=0, variance=1, N=50, value=5)
```

```
0.7602499389065233
```

```
Life.portfolio_percentile(mean=0, variance=1, N=50, prob=0.76)
```

```
4.994313317536634
```

The `set_interest` method is called to initialize and store an `Interest` object with the assumed interest rate, expressed in any annual or m-thly form, which is used in all subsequent life contingency computations.

```
life = Life().set_interest(i=0.05)
life.interest.delta
```

```
0.04879016416943205
```

SOA Question 2.2:

Scientists are searching for a vaccine for a disease. You are given:

- 100,000 lives age x are exposed to the disease
- Future lifetimes are independent, except that the vaccine, if available, will be given to all at the end of year 1

- The probability that the vaccine will be available is 0.2
- For each life during year 1, $q_x = 0.02$
- For each life during year 2, $q_{x+1} = 0.01$ if the vaccine has been given and $q_{x+1} = 0.02$ if it has not been given

Calculate the standard deviation of the number of survivors at the end of year 2.

```
print("SOA Question 2.2: (D) 400")
p1 = (1. - 0.02) * (1. - 0.01)  # 2_p_x if vaccine given
p2 = (1. - 0.02) * (1. - 0.02)  # 2_p_x if vaccine not given
print(math.sqrt(Life.conditional_variance(p=.2, p1=p1, p2=p2, N=100000)))
print(math.sqrt(Life.mixture(p=.2, p1=p1, p2=p2, N=100000, variance=True)))
```

```
SOA Question 2.2: (D) 400
396.5914603215815
396.5914603237804
```

Normal distribution table:

The `quantiles_frame` method displays an extract of normal distribution table for common quantile values, that is made available during SOA FAM-L exam.

```
print("Values of z for selected values of Pr(Z<=z)")
print("-------------------------------------------")
print(Life.quantiles_frame().to_string(float_format=lambda x: f"{x:.3f}"))
```

```
Values of z for selected values of Pr(Z<=z)
-------------------------------------------
z          0.842  1.036  1.282  1.645  1.960  2.326  2.576
Pr(Z<=z)   0.800  0.850  0.900  0.950  0.975  0.990  0.995
```

CHAPTER

FOUR

SURVIVAL MODELS

The future lifetime of an individual is represented as a random variable. Under this framework, probabilities of death or survival, as well as an important quantity known as the force of mortality, can be calculated. It is shown how these quantitites are related, along with some actuarial notation.

4.1 Lifetime distribution

Let (x) denotes a life aged x, where $x \geq 0$, and T_x is time-to-death, or future lifetime, of (x). This means that $x + T_x$ represents the age-at-death random variable for (x).

$F_x(t) = Pr[T_x \leq t] = \int_0^t f_x(t) ds$

- probability that (x) does not survive beyond age $x + t$.

Lifetime density function

$f_x(t) = \frac{\partial}{\partial t} F_x(t) = \dfrac{f_0(x+t)}{S_0(x)}$

- probability density function for the random variable T_x

4.2 Survival function

In life insurance problems we may be interested in the probability of survival rather than death

$S_x(t) \equiv {}_tp_x = Pr[T_x > t] = 1 - F_x(t)$

- the probability that (x) survives for at least t years

$S_x(t) = \dfrac{S_0(x+t)}{S_0(x)}$

- by assumption that $Pr[T_x \leq t] = Pr[T_0 \leq x + t | T_0 > x]$

$S_x(t) = \int_t^\infty f_x(s) ds$

- since $\int_0^t f_x(s)) ds + \int_t^\infty f_x(s) ds = 1$

$S_x(t) = \dfrac{l_{x+t}}{l_x}$

- a life table provides a concrete way to visualize the survival function: from a certain number of lives at a starting age, we apply survival probabilities to compute the expected number of lives at any other age.

$S_x(0) = 1, \; S_x(\infty) = 0, \; S'_x(t) \leq 0$

- conditions on S_x to be a valid survival function

Percentiles:

A $100p$-th percentile of survival time is the time t such that there is a probability p that survival time ie less than t.

4.3 Force of mortality

What actuaries call the force of mortality, denoted μ_x, is known as the hazard rate in survival analysis and the failure rate in reliability theory.

$$\mu_{x+t} = \frac{f_x(t)}{S_x(t)} = \frac{-\frac{\partial}{\partial t} l_{x+t}}{l_{x+t}} = \frac{-\frac{\partial}{\partial t}\, {}_t p_x}{{}_t p_x} = \frac{\partial}{\partial t} \ln\ {}_t p_x$$

- the force of mortality actually completely determines the survival function

$$f_x(t) = \ {}_t p_x\ \mu_{x+t}$$

- can be related to the lifetime density function for (x)

$$S_x(t) = e^{-\int_x^{x+t} \mu_s ds} = e^{-\int_0^t \mu_{x+s} ds}$$

- the survival probability, conditional on survival to the age represented by the lower bound, can be computed by integrating the force of mortality up to the age represented by the upper bound.

$$\mu_x dx \approx Pr[T_0 \le x + dx | T_0 > x]$$

- intuitively, multiplying the force of mortality by a small number dx can be interpreted as the probability that (x) dies before attaining age $x + dx$

$$\int_0^\infty \mu_{x+s} ds = \infty$$

- since $S_x(\infty) = 0$

4.4 Actuarial notation

The actuarial profession uses shorthand notation to record mathematical formulas:

Survival probability

$${}_t p_x = Pr[T_x > t] \equiv S_x(t)$$

- probability that (x) survives to at least age $x + t$

Expected number of survivors

$$l_x = l_{x_0}\ {}_{x-x_0} p_{x_0}$$

- is the expected number of surviving lives at age x from l_{x_0} independent individuals aged x_0.

Mortality rate

$${}_t q_x = 1 - \ {}_t p_x \equiv F_x(t)$$

- probability that (x) dies before age $x + t$.

Deferred mortality rate

$${}_{u|t} q_x = Pr[u < T_x \le u + t] = \int_u^{u+t} {}_s p_x \mu_{x+s} ds$$

- probability that (x) survives u years, and then dies within the subsequent t years.

$${}_{u|t}q_x = \frac{l_{x+u} - l_{x+u+t}}{l_x}$$

- can be related to the number of lives in the life table: total number of deaths of ages between $x + u$ and $x + u + t$, divided by number of lives aged x

$${}_{u|t}q_x = {}_u p_x t q_{x+u} = {}_{u+t}q_x - {}_u q_x = {}_u p_x - {}_{u+t}p_x$$

- there are three useful formulas for computing deferred mortality: (1) deferred mortality (2) limited mortality, or (3) limited survival probabilities.

4.5 Methods

The `Survival` class implements methods to compute and apply relationships between the various equivalent forms of survival and mortality functions. The force of mortality function fully describes the lifetime distribution, just as the survival function does.

```python
import math
from actuarialmath import Survival
import describe
describe.methods(Survival)
```

```
class Survival - Set and derive basic survival and mortality functions

    Methods:
    --------

    set_survival(S, f, l, mu, minage, maxage):
      Construct the basic survival and mortality functions given any one form

    l_x(x, s):
      Number of lives at integer age [x]+s: l_[x]+s

    d_x(x, s):
      Number of deaths at integer age [x]+s: d_[x]+s

    p_x(x, s, t):
      Probability that [x]+s lives another t years: : t_p_[x]+s

    q_x(x, s, t, u):
      Probability that [x]+s lives for u, but not t+u years: u|t_q_[x]+s

    f_x(x, s, t):
      Lifetime density function of [x]+s after t years: f_[x]+s(t)

    mu_x(x, s, t):
      Force of mortality of [x] at  s+t years: mu_[x](s+t)
```

4.6 Examples

The `set_survival` class method is called to specify any survival model by either its survival probability (with the argument `S`), force of mortality (`mu`), number of lives (`l`), or lifetime density (`f`) functions. Then the various survival and mortality probabilities and function values can be retrieved using methods with names similar to their respective actuarial notations: `p_x, q_x, f_x, mu_x, d_x, l_x`. In later sections, special mortality laws like the uniform and exponential distributions, along with their available shortcut formulas, will be described and implemented.

```
# specify survival function by linearly declining number of lives
life = Survival().set_survival(l=lambda x,s: 100-(x+s), minage=0, maxage=100).set_
↪interest(i=0.05)
print(life.l_x(x=0), life.l_x(x=100), life.l_x(x=50), life.d_x(x=0), life.d_x(x=50))
for x in [10, 50, 90]:
  print(life.p_x(x=x), life.q_x(x=x, u=10, t=10), life.f_x(x=x), life.mu_x(x=x))
```

```
100 0 50 1 1
0.9888888888888889 0.11111111111111105 1.0 0.011111111111111112
0.98 0.20000000000000007 1.0 0.02
0.9 1.0 1.0 0.1
```

SOA Question 2.3:

You are given that mortality follows Gompertz Law with B = 0.00027 and c = 1.1. Calculate $f_{50}(10)$.

```
print("SOA Question 2.3: (A) 0.0483")
B, c = 0.00027, 1.1
def S(x,s,t): return (math.exp(-B * c**(x+s) * (c**t - 1)/math.log(c)))
life = Survival().set_survival(S=S)
print(life.f_x(x=50, t=10))
```

```
SOA Question 2.3: (A) 0.0483
0.048327399045049846
```

SOA Question 2.6

You are given the survival function:

$$S_0(x) = \left(1 - \frac{x}{60}\right)^{\frac{1}{3}}, \quad 0 \le x \le 60$$

Calculate $1000\mu_{35}$.

```
print("# SOA Question 2.6: (C) 13.3")
life = Survival().set_survival(l=lambda x,s: (1 - (x+s) / 60)**(1 / 3))
print(1000*life.mu_x(35))
```

```
# SOA Question 2.6: (C) 13.3
13.337287043994648
```

SOA Question 2.7

You are given the following survival function of a newborn:

$$\begin{aligned} S_0(x) &= 1 - \frac{x}{250}, \quad 0 \leq x < 40 \\ &= 1 - (\frac{x}{100})^2, \quad 40 \leq x \leq 100 \end{aligned}$$

Calculate the probability that (30) dies within the next 20 years

```
print("SOA Question 2.7: (B) 0.1477")
def S(x,s):
    return 1 - ((x+s) / 250) if (x+s) < 40 else 1 - ((x+s) / 100)**2
print(Survival().set_survival(l=S).q_x(30, t=20))
```

```
SOA Question 2.7: (B) 0.1477
0.1477272727272727
```

SOA Question 2.8

In a population initially consisting of 75% females and 25% males, you are given:

- For a female, the force of mortality is constant and equals μ
- For a male, the force of mortality is constant and equals 1.5μ
- At the end of 20 years, the population is expected to consist of 85% females and 15% males.

Calculate the probability that a female survives one year.

```
print("SOA Question 2.8: (C) 0.938")
def fun(mu):  # Solve first for mu, given ratio of start and end proportions
    male = Survival().set_survival(mu=lambda x,s: 1.5 * mu)
    female = Survival().set_survival(mu=lambda x,s: mu)
    return (75 * female.p_x(0, t=20)) / (25 * male.p_x(0, t=20))
mu = Survival.solve(fun, target=85/15, grid=0.5)
p = Survival().set_survival(mu=lambda x,s: mu).p_x(0, t=1)
print(p)
```

```
SOA Question 2.8: (C) 0.938
0.9383813306903799
```

CAS41-F99:12

You are given the following survival function:

$S(x) = 100(k - \frac{x}{2})^{\frac{2}{3}}$

Find k, given that $\mu_{50} = \frac{1}{48}$

```
print("CAS41-F99:12: k = 41")
def fun(k):
    return Survival().set_survival(l=lambda x,s: 100*(k - (x+s)/2)**(2/3))\
                     .mu_x(50)
print(Survival.solve(fun, target=1/48, grid=50))
```

```
CAS41-F99:12: k = 41
41.00115767959991
```

CHAPTER

FIVE

EXPECTED FUTURE LIFETIMES

In many insurance applications we are interested in the first and second moments not only of the future lifetime of an individual, but also of the individual's *curtate* future lifetime, defined as the integer part of future lifetime. For some lifetime distributions (as described in later sections), we are able to obtain the mean and variances of future lifetimes directly without using numerical integrationn techniques.

5.1 Complete expectation of life

$\mathring{e}_x = E[T_x] = \int_0^\infty t\ {}_tp_x\ \mu_{x+t} dt = \int_0^\infty\ {}_tp_x\ dt$

- is the complete expectation of life, or the mean future lifetime: it can be computed either as (1) the weighted sum of lifetimes lived or (2) the sum over all years of the probabilities of living at least those years.

$E[T_x^2] = \int_0^\infty t^2\ {}_tp_x\ \mu_{x+t}\ ds = \int_0^\infty\ 2\ t\ {}_tp_x\ dt$

- second moment of future lifetime

$Var[T_x] = E[T_x^2] - (\mathring{e}_x)^2$

- variance of future lifetime

5.2 Curtate expectation of life

$K_x = \lfloor T_x \rfloor$

- is the curtate future lifetime random variable, representing the number of completed whole future years by (x) prior to death

$e_x = E[K_x] = \sum_{k=0}^\infty k\ {}_{k|}q_x\ = \sum_{k=1}^\infty\ {}_kp_x\ dt$

- Is the curtate expectation of life, or the mean curtate lifetime: it can be computed two ways intuitively similar to the approach of complete expectations.

$E[K_x^2] = \sum_{k=0}^\infty k^2\ {}_{k|}q_x\ = \sum_{k=1}^\infty (2k-1)\ {}_kq_x\ dt$

- second moment of curtate future lifetime

$Var[K_x] = E[K_x^2] - (e_x)^2$

- variance of curtate future lifetime

5.3 Temporary expectation of life

We are sometimes interested in the future lifetime random variable subject to a cap of n years, which is represented by the random variable $min(T_x, n)$.

$\overset{\circ}{e}_{x:\overline{n|}} = \int_0^n t\ {}_tp_x\ \mu_{x+t}\ ds + n\ {}_np_x = \int_0^n\ {}_tp_x\ dt$

- term complete expectation of life

$e_{x:\overline{n|}} = \sum_{k=0}^{n-1} k\ {}_{k|}q_x\ + n\ {}_np_x = \sum_{k=1}^{n}\ {}_kp_x$

- temporary curtate expectation of life, limited at n years

5.4 Methods

The `Lifetime` class implements methods to compute curtate and complete expectations and second moments of future lifetime

```
from actuarialmath import Lifetime
import describe
describe.methods(Lifetime)
```

```
class Lifetime - Computes expected moments of future lifetime

    Methods:
    --------

    e_x(x, s, t, curtate, moment):
      Compute curtate or complete expectations and moments of life
```

5.5 Examples

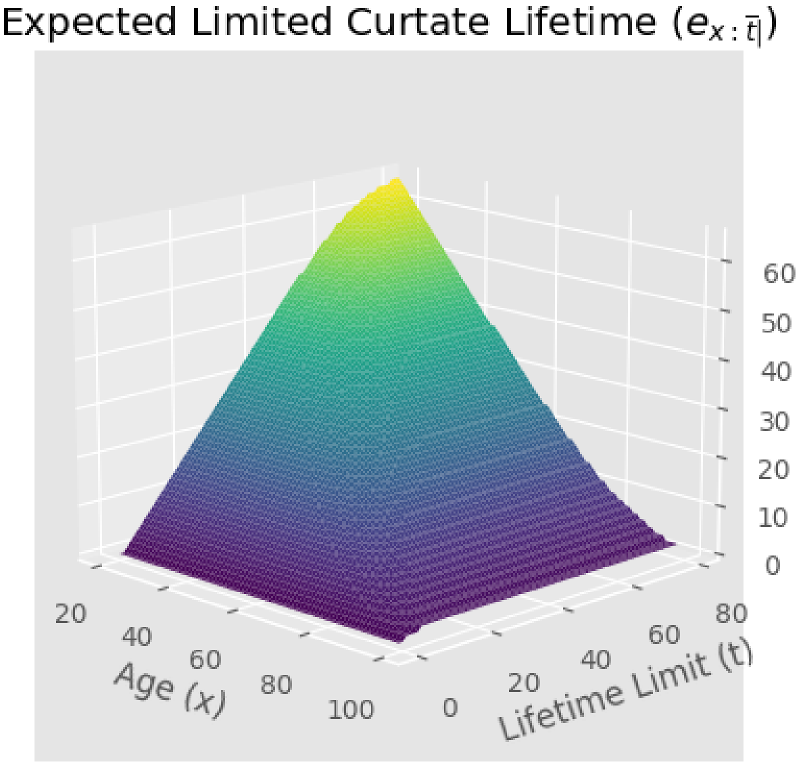
The `e_x` class method computes the mean, variance or second moment (by setting the value of the argument `moment` to be 1, VARIANCE, or 2 respectively) of either the complete or curtate future lifetime (by setting the value of the argument `curtate` to be False or True respectively).

```
life = Lifetime().set_survival(mu=lambda *args: 0.1)
print(life.e_x(x=50, curtate=False), life.e_x(x=50, moment=life.VARIANCE,␣
 ↪curtate=False))
print(life.e_x(x=50, curtate=True), life.e_x(x=50, moment=life.VARIANCE,␣
 ↪curtate=True))
```

```
9.999999999999998 99.99999999999997
9.508331944775032 99.91670831680462
```

Limited future lifetimes are specified by the argument `t`. The example below plots the expected limited curtate lifetimes by varying age x and lifetime limit t.

```
import math
import numpy as np
import matplotlib.pyplot as plt

def S(x, s, t):  # define a survival function
    return math.exp(-.00022*t - (.0000027*1.124**(x+s)*(1.124**t-1))/math.log(1.124))
life = Lifetime().set_survival(S=S, minage=20, maxage=130)

X, Y = np.meshgrid(np.arange(20, 100, 1), np.arange(0, 80, 1))
Z = np.zeros_like(X)
for i in range(Z.shape[0]):
    for j in range(Z.shape[1]):
        Z[i][j] = life.e_x(x=X[i][j], t=Y[i][j])

ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='viridis', edgecolor=None)
ax.set_xlabel('Age (x)')
ax.set_ylabel('Lifetime Limit (t)')
ax.set_title('Expected Limited Curtate Lifetime ($e_{x:\overline{t}|}$)')
ax.view_init(15, -45)
```



```
Y
```

```
array([[ 0,  0,  0, ...,  0,  0,  0],
       [ 1,  1,  1, ...,  1,  1,  1],
       [ 2,  2,  2, ...,  2,  2,  2],
       ...,
       [57, 57, 57, ..., 57, 57, 57],
```

(continues on next page)

(continued from previous page)

```
        [58, 58, 58, ..., 58, 58, 58],
        [59, 59, 59, ..., 59, 59, 59]])
```

AMLCR Exericse 2.1

Let $F_0(t) = 1 - (1 - t/105)^{1/5}$ for $0 \leq t \leq 105$. Calculate

(a) the probability that a newborn life dies before age 60 [0.1559]

(b) the probability that a life aged 30 survives to at least age 70 [0/8586]

(c) the probability that a life aged 20 dies between ages 90 and 100 [0.1394]

(d) the force of mortality at age 50 [0.0036]

(e) the median future lifetime at age 50 [53.28]

(f) the complete expectation of life at age 50 [45.83]

(g) the curtate expectation of life at age 50 [45.18]

```
def F(t): return (1 - (1 - t/105)**(1/5) if t <= 105 else 1)
life = Lifetime().set_survival(S=lambda x,s,t: (1-F(x+s+t)) / (1-F(x+s)))
print("(a)", life.q_x(x=0, t=60))
print("(b)", life.p_x(x=30, t=70-30))
print("(c)", life.q_x(x=20, u=90-20, t=100-90))
print("(d)", life.mu_x(x=50))
print("(e)", life.solve(fun=lambda t: life.p_x(x=50, t=t), target=0.5, grid=105/2))
print("(f)", life.e_x(x=50, curtate=False))
print("(g)", life.e_x(x=50))
```

```
(a) 0.1558791201558899
(b) 0.8586207034844889
(c) 0.13943444007155992
(d) 0.003636764417087519
(e) 53.28125000000001
(f) 45.833333325511866
(g) 45.17675143564247
```

SOA Question 2.1

You are given:

- $S0(t) = (1 - \frac{t}{\omega})^{\frac{1}{4}}, \quad 0 \leq t \leq \omega$
- $\mu_{65} = \frac{1}{180}$

Calculate e_{106}, the curtate expectation of life at age 106.

```
print("SOA Question 2.1: (B) 2.5")
def fun(omega):  # Solve first for omega, given mu_65 = 1/180
    return Lifetime().set_survival(l=lambda x,s: (1 - (x+s)/omega)**0.25,
                                   maxage=omega)\
                     .mu_x(65)
omega = int(Lifetime.solve(fun, target=1/180, grid=100))  # solve for omega
e = Lifetime().set_survival(l=lambda x,s: (1 - (x+s)/omega)**0.25,
                            maxage=omega).e_x(106)
print(e)
```

```
SOA Question 2.1: (B) 2.5
2.4786080555423604
```

SOA Question 2.4

You are given ${}_tq_0 = \frac{t^2}{10{,}000} \quad 0 < t < 100$. Calculate $\overset{\circ}{e}_{75:\overline{10|}}$.

```
print("SOA Question 2.4: (E) 8.2")
def q(t) : return (t**2)/10000. if t < 100 else 1.
e = Lifetime().set_survival(l=lambda x,s: 1 - q(x+s))\
              .e_x(75, t=10, curtate=False)
print(e)
```

```
SOA Question 2.4: (E) 8.2
8.20952380952381
```

CHAPTER
SIX

FRACTIONAL AGES

Given values of l_x at integer ages only, we need to make some assumption about the probability distribution for the future lifetime random variable between integer ages, in order to calculate survival function values for non-integer ages or durations. Such fractional age assumptions may be specified in terms of the force of mortality function (e.g. constant) or the survival or mortality probabilities (e.g. uniform distribution of deaths).

6.1 Uniform distribution of deaths

This is the simplest assumption, which linearly interpolates the number of lives between integer ages.

$T_x = K_x + R_x$

- The UDD assumptions define can be modeled by a new random variable $R_x \sim U(0, 1)$ which is independent of curtate lifetime K_x.

${}_r q_x = r\ q_x$, for integer x and $0 \leq r < 1$

- is an equivalent way of formulating the UDD assumption

$l_{x+r} = (1 - r)\ l_x + r\ l_{x+1} = l_x - r\ d_x$

- linear interpolation of lives between integer ages

$${}_r q_{x+s} = \frac{r\ q_x}{1 - s\ q_x}, \quad \text{for } 0 \leq s + r < 0$$

- mortality rate at a fractional age over a fractional duration, under UDD

$$\mu_{x+r} = \frac{1}{1 - r\ q_x}$$

- applying the UDD approximation over successive ages implies a discontinuous function for the force of mortality, with discontinuities occurring at integer ages.

$f_x(r) = {}_r p_x \mu_{x+r} = q_x$

- lifetime density is constant between integer ages, which also follows from the UDD assumption for R_x.

$$\mathring{e}_{x:\overline{1}|} = 1 - q_x \frac{1}{2} = q_x \frac{1}{2} + p_x$$

- 1-year limited complete expectation

$$\mathring{e}_{x:\overline{r}|} = {}_r q_x \frac{r}{2} + {}_r p_x r$$

- fractional year limited complete expectation

$$\mathring{e}_x = q_x \frac{1}{2} + p_x (1 + \mathring{e}_{x+1}) = \mathring{e}_{x:\overline{1}|} + p_x \mathring{e}_{x+1}$$

- recursive for complete expectation of life: UDD-specific and general formulas

$$\overset{\circ}{e}_x = e_x + 0.5$$

- since $E[R_x] = 0.5$

$$Var(T_x) = Var(K_x) + \frac{1}{12}$$

- since $Var[R_x] = \frac{1}{12}$

$$\overset{\circ}{e}_{x:\overline{n|}} = e_{x:\overline{n|}} + 0.5\ {}_nq_x$$

- These exact results under UDD are often used generally as approximations between complete and curtate expectations.

6.2 Constant force of mortality

Under this assumption, the force of mortality is set to be constant between integer ages

$$\mu_{x+r} = \ \mu_x = \ -\ln\ p_x, \quad \text{for } 0 \le r < 1$$

- which leads to a step function for the force of mortality over successive years of age

$$l_{x+r} = (l_x)^{1-r} \cdot (l_{x+1})^r$$

- constant force of mortality is equivalent to exponential interpolation of lives between integer ages

$${}_rp_x = e^{-\mu_x r} = (p_x)^r$$

- since $p_x = e^{-\int_0^1 \mu_{x+u} du} = e^{-\mu_x}$

$${}_rp_{x+s} = e^{-\int_0^r \mu_{x+s+u} du} = (p_x)^r, \quad \text{for } 0 \le r + s < 1$$

- the probability of surviving for period of r from age $x + s$ is independent of s under constant force of mortality

$$f_x(r) = \ {}_rp_x\ \mu_{x+r} = e^{-\mu_x r} \cdot \mu_x, \quad \text{for } 0 \le r < 1$$

- lifetime density function at fractional age using constant force of mortality assumption

$$\overset{\circ}{e}_{x:\overline{r|}} = \frac{1 - e^{-\mu_x r}}{\mu}$$

- fractional year limited complete expectation

6.3 Methods

The `Fractional` class implements methods to compute survival and mortality functions between integer ages, assuming either uniform distribution of deaths or constant force of mortality

```
from actuarialmath import Fractional
import describe
describe.methods(Fractional)
```

```
class Fractional - Compute survival functions at fractional ages and durations

    Args:
      udd : select UDD (True, default) or CFM (False) between integer ages

    Methods:
    --------

    l_r(x, s, r):
      Number of lives at fractional age: l_[x]+s+r

    p_r(x, s, r, t):
      Probability of survival from and through fractional age: t_p_[x]+s+r

    q_r(x, s, r, t, u):
      Deferred mortality rate within fractional ages: u|t_q_[x]+s+r

    mu_r(x, s, r):
      Force of mortality at fractional age: mu_[x]+s+r

    f_r(x, s, r, t):
      mortality function at fractional age: f_[x]+s+r (t)

    E_r(x, s, r, t):
      Pure endowment at fractional age: t_E_[x]+s+r

    e_r(x, s, t):
      Temporary expected future lifetime at fractional age: e_[x]+s:t

    e_approximate(e_complete, e_curtate, variance):
      Convert between curtate and complete expectations assuming UDD shortcut
```

6.4 Examples

The `Fractional` class can be initialized to apply the uniform distribution or constant force of mortality assumptions at fractional ages (by setting the `udd` argument to be either True or False respectively), while using any other survival model for integer ages. Then survival function values at fractional ages can be computed by the methods `l_r`, `p_r`, `q_r`, `mu_r`, `f_r` and `E_r` (for the number of lives, survival probability, mortality rate, force of mortality, lifetime density and pure endowment functions respectively).

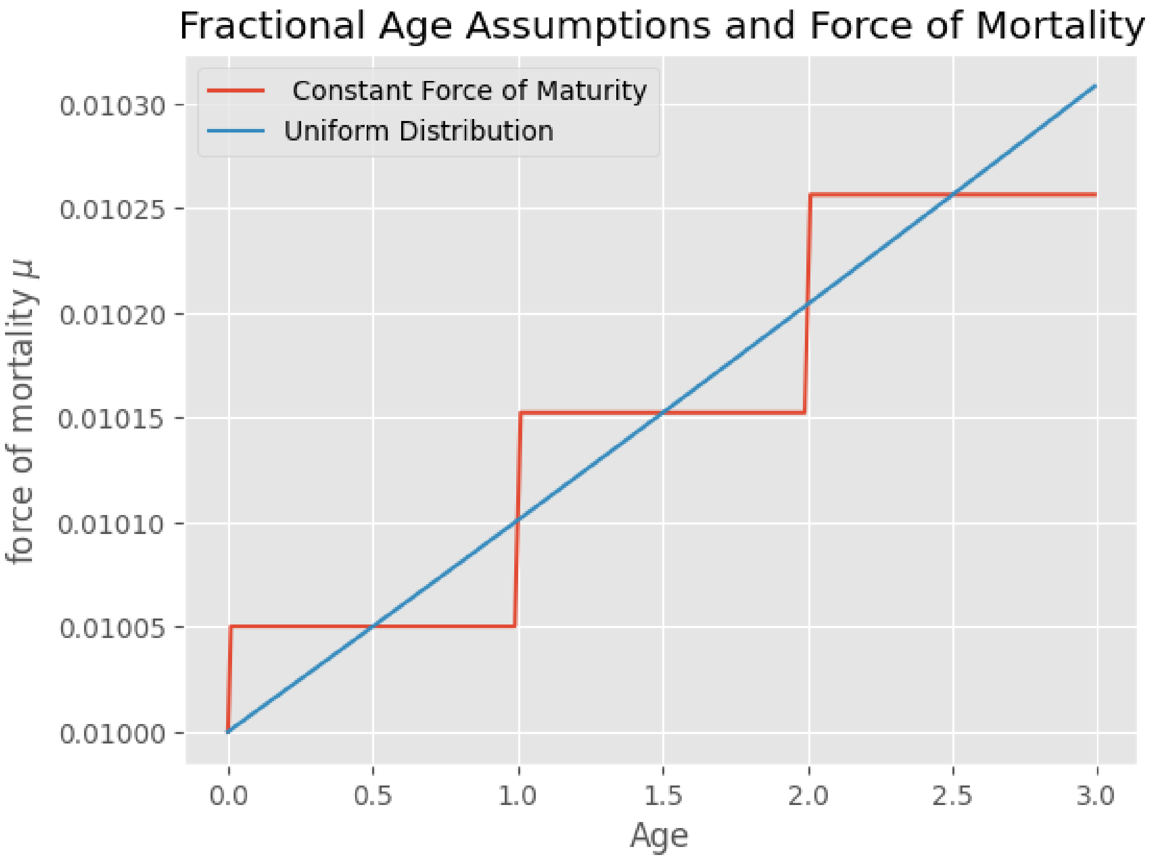
```python
cfm = Fractional(udd=False).set_survival(l=lambda x,t: 50-x-t)
udd = Fractional(udd=True).set_survival(l=lambda x,t: 50-x-t)
print('Compare:   Constant Force of Mortality    vs    UDD  assumptions')
print('           ---------------------------         -----')
print('mortality rate    ', cfm.q_r(40, t=0.5), udd.q_r(40, t=0.5))
print('force of mortality', cfm.mu_r(40, r=0.5), udd.mu_r(40, r=0.5))
print('lifetime density  ', cfm.f_r(40, r=0.5), udd.f_r(40, r=0.5))
```

```
Compare:   Constant Force of Mortality    vs    UDD  assumptions
           ---------------------------         -----
mortality rate     0.05131670194948623 0.04999999999999999
force of mortality 0.10536051565782628 0.10526315789473682
lifetime density   0.10536051565782628 0.09999999999999998
```

When using constant force of maturity as the fractional age assumption, the force of maturity may show discontinuous jumps at integer ages

```
import numpy as np
import matplotlib.pyplot as plt
cfm = Fractional(udd=False).set_survival(l=lambda x,s: 100 - (x+s))
udd = Fractional(udd=True).set_survival(l=lambda x,s: 100 - (x+s))
x = np.arange(0, 3, 0.01)
plt.plot(x, [cfm.mu_r(x=0, r=r) for r in x])
plt.plot(x, [udd.mu_r(x=0, r=r) for r in x])
plt.legend([' Constant Force of Maturity', 'Uniform Distribution'])
plt.ylabel('force of mortality $\mu$')
plt.xlabel('Age')
plt.title('Fractional Age Assumptions and Force of Mortality')
```

```
Text(0.5, 1.0, 'Fractional Age Assumptions and Force of Mortality')
```



The uniform distribution assumption for fractional ages linearly interpolates the number of lives between integer ages, whereas the constant force of mortality asssumption uses exponential interpolation.

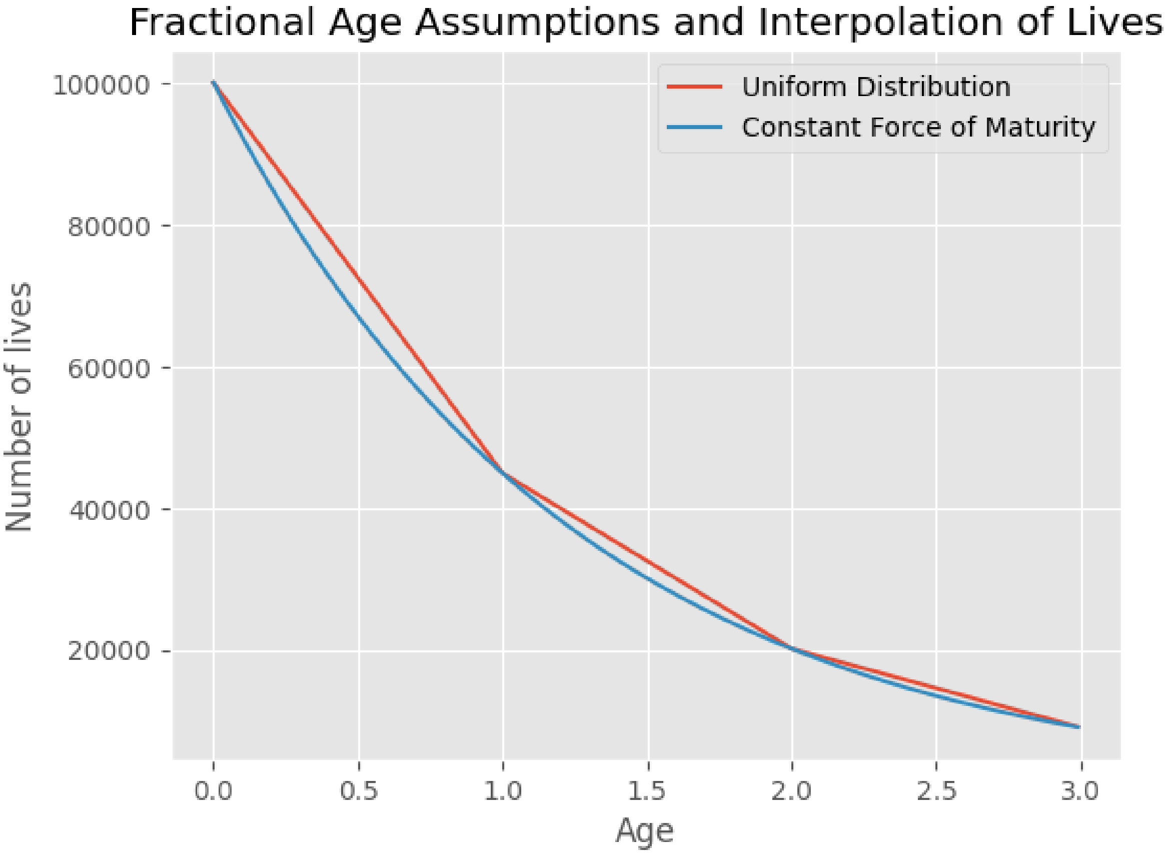
```
udd = Fractional(udd=True).set_survival(mu=lambda x,s: 0.8)
cfm = Fractional(udd=False).set_survival(mu=lambda x,s: 0.8)
plt.plot(x, [udd.l_r(x=0, r=r) for r in x])
plt.plot(x, [cfm.l_r(x=0, r=r) for r in x])
plt.legend(['Uniform Distribution', 'Constant Force of Maturity'])
plt.ylabel('Number of lives')
```

(continues on next page)

(continued from previous page)

```
plt.xlabel('Age')
plt.title('Fractional Age Assumptions and Interpolation of Lives')
```

```
Text(0.5, 1.0, 'Fractional Age Assumptions and Interpolation of Lives')
```



The complete future lifetime for fractional limited durations starting from integer ages is computed by the `e_r` class method. Under the hood, when the duration is one year or less, then shortcuts are available for both the uniform distribution and constant force of maturity cases; when the limited duration is more than one year, then the general one-year recursion formula is repeatedly applied until the last period of length less than or equal to one year is reached.

```
import numpy as np
x = [0.5, 1., 1.5, 2.]
udd = Fractional(udd=False).set_survival(l=lambda x,s: 50 - (x+s), maxage=50)
cfm = Fractional(udd=True).set_survival(mu=lambda x,s: 0.03)
print('UDD: ', [round(udd.e_r(x=0, t=t), 4) for t in x])
print('CFM: ', [round(cfm.e_r(x=0, t=t), 4) for t in x])
```

```
UDD:  [0.4975, 0.99, 1.4774, 1.9599]
CFM:  [0.4963, 0.9852, 1.4669, 1.9413]
```

The exact results under the assumption of uniform distribution of deaths are often used to approximate between curtate and complete expectations, and can be computed by the `e_approximate` static method

```
print(Fractional.e_approximate(e_complete=15),
      Fractional.e_approximate(e_complete=200, variance=True))
print(Fractional.e_approximate(e_curtate=15),
      Fractional.e_approximate(e_curtate=200, variance=True))
```

```
14.5 199.91666666666666
15.5 200.08333333333334
```

CHAPTER

SEVEN

INSURANCE

For a life insurance policy, the time at which the benefit will be paid is unknown until the policyholder actually dies and the policy becomes a claim. In this section, formulas are developed for the valuation of traditional insurance benefits (such as whole life, term and endowment insurance) and their second moments, as well as varying insurances.

7.1 Present value of life insurance r.v. Z

Valuation functions for the present value of insurance benefits, denoted by Z, are based on the continuous future lifetime random variable, T_x, or the curtate future lifetime random variable, K_x. The interest rate is generally assumed to be fixed and constant. Given a survival model and an interest rate we can derive the distribution of the present value random variable for a life contingent benefit, and can therefore compute quantities such as the mean and variance of the present value. Valuation functions are developed per dollar of sum assured, which can be multiplied by the actual sum insured for different benefit amounts. The expected present value of insurance benefit, denoted and solved by $EPV(Z)$, is sometimes referred to as the actuarial value, actuarial present valuen (APV), or single net premium. Later sections will review special mortality law assumptions where EPV's can be computed without the need for integration.

7.2 Whole life insurance

Whole life insurance pays a benefit whenver (x) dies.

$Z = v^{T_x}$

- continuous insurance, where the benefit of \$1 is payable immediately on death

$\overline{A}_x = E[v^{T_x}] = \int_{t=0}^{\infty}~v^t~{}_tp_x~\mu_{x+t}~dt$

- EPV continuous whole life insurance that pays benefit of \$1 immediately on death

$Z = v^{K_x+1}$

- annual insurance, where benefit of \$1 is payable at end of year of death

$A_x = E[v^{K_x+1}] = \sum_{k=0}^{\infty}~v^{k+1}~{}_{k|}q_x$

- EPV of annual whole life insurance that pays benefit of \$1 at end of year on death

7.3 Term insurance

A term life insurace for t years pays the benefit if death occurs within t years, 0 otherwise.

$Z = 0$ when $T_x > t$, else v^{T_x}

- death benefit is payable immediately only if the policy-holder dies within t years

$\overline{A}^{1}_{x:\overline{t|}} = \int_{s=0}^{t} ~ v^s ~ {}_sp_x ~ \mu_{x+s} ~ ds = \overline{A}_x - ~{}_tE_x ~ \overline{A}_{x+t}$

- continous term insurance as the difference of continous whole life and deferred continuous whole life

$Z = 0$ when $K_x \geq t$, else v^{K_x+1}

- death benefit is payable at the end of the year of death provided this occurs within t years

$A^{1}_{x:\overline{t|}} = \sum_{k=0}^{t-1} ~ v^{k+1} ~ {}_{k|}q_x = A_x - ~{}_tE_x ~ A_{x+t}$

- annual term insurance as the difference of annual whole life and deferred annual whole life

7.4 Deferred insurance

Does not begin to offer death benefit cover until the end of a deferred period

$Z = 0$ when $T_x < u$, else v^{T_x}

- benefit is payable immediately on the death of (x) provided that (x) dies after the age of $x + u$

${}_{u|}\overline{A}_{x:\overline{t|}} = ~{}_uE_x ~ \overline{A}_{x+u:\overline{t|}}$

- continuous deferred insurance as the EPV of whole life insurance starting at the end of deferred period.

$Z = 0$ when $K_x < u$, else v^{K_x+1}

- annual deferred insurance where the death benefit is payable at the end of the year of death

${}_{u|}A_{x:\overline{t|}} = ~{}_uE_x ~ A_{x+u:\overline{t|}}$

- annual deferred insurance as the EPV of whole life insurance starting at the end of deferred period.

7.5 Endowment insurance

An endowment insurance provides a combination of a term insurance and a pure endowment.

$Z = v^t$ when $T_x \geq t$, else v^{T_x}

- benefit is payable on the death of (x) should (x) die within t years, but if (x) survives for t years, the sum insured is payable at the end of the t-th year.

$\overline{A}_{x:\overline{t|}} = \overline{A}^{1}_{x:\overline{t|}} + ~{}_tE_x$

- continuous endowment insurance as continuous term insurance plus a pure endowment

$Z = v^t$ when $K_x \geq t$, else v^{K_x+1}

- annual endowment insurance where the death benefit is payable at the end of the year of death

$A_{x:\overline{t|}} = A^{1}_{x:\overline{t|}} + ~{}_tE_x$

- annual endowment insurance as annual term insurance plus a pure endowment

7.6 Pure endowment

$Z = 0$ when $T_x < t$, else $Z = v^t$

- benefit payable in t years if (x) is still alive at that time, but pays nothing if (x) dies before then.

$${}_nE_x = A_{x:\overline{n|}}^{\;\;1} = v^n \; {}_np_x$$

- because the pure endowment will be paid only at time t, assuming the life survives, there is only the discrete time version

7.7 Variances

The variance of life insurance benefits is computed as the difference of the second moment $E[Z^2]$ and square of the first moment $E[Z]^2$. By the rule of momenets: $E[Z^n]$ at force of interest δ equals $E[Z]$ at force of interest $n\delta$ with the benefit amount also raised to the n-th power.

$$E[Z^n] = \int_{t=0}^{\infty} \; b_t^n \; v_t^n \; {}_tp_x \; \mu_{x+t} \; dt$$

- when calculating with moments, the benefit and discount factor are raised to the n-th power.

$${}^2\overline{A}_x = \int_{t=0}^{\infty} \; v^{2t} \; {}_tp_x \; \mu_{x+t} \; dt$$

- second moment of continuous insurance equal to $\overline{A}_x$ at double the force of interest.

$${}^2A_x = \sum_{k=0}^{\infty} \; v^{2(k+1)} \; {}_{k|}q_x$$

- second moment of annual insurance is equal to A_x at double the force of interest.

Whole Life insurance:

$$Var(\overline{A}_x) = {}^2\overline{A}_x - (\overline{A}_x)^2$$

- variance of continuous whole life insurance

$$Var(A_x) = {}^2A_x - \; (A_x)^2$$

- variance of annual whole life insurance

Term Life insurance:

$$Var(\overline{A}^{1}_{x:\overline{t|}}) = {}^2\overline{A}^{1}_{x:\overline{t|}} - (\overline{A}^{1}_{x:\overline{t|}})^2$$

- variance of continuous term life insurance

$$Var(A^{1}_{x:\overline{t|}}) = {}^2A^{1}_{x:\overline{t|}} - \; (A^{1}_{x:\overline{t|}})^2$$

- variance of annual term life insurance

Deferred Life insurance:

$$Var({}_{u|}\overline{A}_{x:\overline{t|}}) = {}^2_{u|}\overline{A}_{x:\overline{t|}} - ({}_{u|}\overline{A}_{x:\overline{t|}})^2$$

- variance of continuous deferred life insurance

$$Var({}_{u|}A_{x:\overline{t|}}) = {}^2_{u|}A_{x:\overline{t|}} - \; ({}_{u|}A_{x:\overline{t|}})^2$$

- variance of annual deferred life insurance

Endowment insurance:

$$Var(\overline{A}_{x:\overline{t|}}) = {}^2\overline{A}_{x:\overline{t|}} - (\overline{A}_{x:\overline{t|}})^2 = Var(\overline{A}^{1}_{x:\overline{t|}}) + Var({}_tE_x) - Cov(\overline{A}^{1}_{x:\overline{t|}}, \; {}_tE_x)$$

- variance of continous endowment insurance may also be expressed on the sum of a term insurance and pure endowment, which are mutually exclusive hence $Cov(\overline{A}^{1}_{x:\overline{t|}},\ {}_tE_x) = E[\overline{A}^{1}_{x:\overline{t|}}]E[{}_tE_x]$

$$Var(A_{x:\overline{t|}}) = {}^2A_{x:\overline{t|}} -\ (A_{x:\overline{t|}})^2 = Var(A^1_{x:\overline{t|}}) + Var({}_tE_x) - Cov(A^1_{x:\overline{t|}},\ {}_tE_x)$$

- variance of annual endowment insurance

Pure Endowment:

$${}^2_tE_x = v^{2t}\ {}_tp_x = v^t\ {}_tE_x$$

- second moment of pure endowment by discounting ${}_tE_x$

$$Var(\ {}_tE_x) = v^{2t}\ {}_tp_x\ {}_tq_x = v^{2t}\ {}_tp_x - (v^t\ {}_tp_x)^2$$

- variance of pure endowment is the variance of a Bernoulli random variable

7.8 Varying insurance

Increasing insurance:

Amount of death benefit increases arithmetically at a rate of $1 per year.

$$(\overline{IA})_x = \int_{t=0}^{\infty}\ t\ v^t\ {}_tp_x\ \mu_{x+t}\ dt$$

- increasing continuous whole life insurance

$$(IA)_x = \sum_{k=0}^{\infty}\ (k+1)\ v^{k+1}\ {}_{k|}q_x$$

- increasing annual whole life insurance

$$(\overline{IA})^1_{x:\overline{t|}} = \int_{s=0}^{t}\ s\ v^s\ {}_sp_x\ \mu_{x+s}\ ds$$

- increasing continuous term insurance

$$(IA)^1_{x:\overline{t|}} = \sum_{k=0}^{t-1}\ (k+1)\ v^{k+1}\ {}_{k|}q_x$$

- increasing annual term insurance

Decreasing insurance:

Amount of death benefit increasing arithmetically at a rate of $1 per year.

$$(\overline{DA})^1_{x:\overline{t|}} = \int_{s=0}^{t}\ (t-s)\ v^s\ {}_sp_x\ \mu_{x+s}\ ds$$

- decreasing continuous term insurance (not defined for whole life)

$$(DA)^1_{x:\overline{t|}} = \sum_{k=0}^{t-1}\ (t-k)\ v^{k+1}\ {}_{k|}q_x$$

- decreasing annual term insurance (not defined for whole life)

Identity relationship:

$$(\overline{DA})^1_{x:\overline{t|}} + (\overline{IA})^1_{x:\overline{t|}} = t\ \overline{A}^{1}_{x:\overline{t|}}$$

- relates continuous increasing and decreasing insurances to level insurance

$$(DA)^1_{x:\overline{t|}} + (IA)^1_{x:\overline{t|}} = (t+1)\ A^1_{x:\overline{t|}}$$

- relates annual increasing and decreasing insurances to level insurance

7.9 Probabilities and percentiles

The larger T, the lower the present value of the payment and the random variable Z. For a level benefit whole life insurance, the 100π-th percentile of the actuarial present value is determined by the $100(1-\pi)$-th percentile of the lifetime random variable.

Percentile of Z:

$t_\pi = S^{-1}(\pi)$

- the $100(1-p)$-th percentile of the lifetime variable for WL insurance can be obtained by looking up the inverse of the survival function at probability p; e.g. if ${}_tp_x = e^{-\mu t}$, then $t_\pi = \dfrac{-\ln \pi}{\mu}$

$Z_\pi = b_\pi v^{t_\pi}$

- the PV of the insurance r.v. that pays benefit b_{t_π} at time t_π

Probability of Z:

To determine the probability that the present value of an annuity is greater than some percentile value $Pr(Z \geq z_\pi)$, it is necessary to plot Z as a function of t, then identify those time periods where $Z \geq z_\pi$ and compute the cumulative probability of T being in those time periods.

7.10 Methods

The `Insurance` class implements methods to compute the expected present value of life insurance

```
import math
import numpy as np
import matplotlib.pyplot as plt
from actuarialmath import Insurance
import describe
describe.methods(Insurance)
```

```
class Insurance - Compute expected present values of life insurance

    Methods:
    --------

    E_x(x, s, t, endowment, moment):
      Pure endowment: t_E_x

    A_x(x, s, t, u, benefit, endowment, moment, discrete):
      Numerically compute EPV of insurance from basic survival functions

    insurance_variance(A2, A1, b):
      Compute variance of insurance given moments and benefit

    whole_life_insurance(x, s, moment, b, discrete):
      Whole life insurance: A_x

    term_insurance(x, s, t, b, moment, discrete):
      Term life insurance: A_x:t^1

    deferred_insurance(x, s, u, t, b, moment, discrete):
```

(continues on next page)

(continued from previous page)

```
      Deferred insurance n|_A_x:t^1 = discounted term or whole life

    endowment_insurance(x, s, t, b, endowment, moment, discrete):
      Endowment insurance: A_x^1:t = term insurance + pure endowment

    increasing_insurance(x, s, t, b, discrete):
      Increasing life insurance: (IA)_x

    decreasing_insurance(x, s, t, b, discrete):
      Decreasing life insurance: (DA)_x

    Z_x(x, s, t, discrete):
      EPV of year t insurance death benefit for life aged [x]+s: b_x[s]+s(t)

    Z_t(x, prob, discrete):
      T_x, given the prob of the PV of life insurance, i.e. r.v. Z(t)

    Z_from_t(t, discrete):
      PV of annual or continuous insurance payment Z(t) at t=T_x

    Z_from_prob(x, prob, discrete):
      Percentile of annual or continuous WL insurance PV r.v. Z given probability

    Z_to_t(Z):
      T_x s.t. PV of continuous WL insurance payment is Z

    Z_to_prob(x, Z):
      Probability that continuous WL insurance PV r.v. is no more than Z

    Z_plot(x, s, stop, benefit, T, discrete, ax, dual, title, color, alpha):
      Plot of PV of insurance r.v. Z vs t
```

7.11 Examples

Given a survival model, the `A_x` class method computes the EPV for arbitrary life insurance benefits, endowments, discount functions, terms and moments using numerical integration or summation.

```
life = Insurance().set_interest(delta=0.06)\
                  .set_survival(mu=lambda *x: 0.04)
benefit = lambda x,t: math.exp(0.02 * t)
A1 = life.A_x(0, benefit=benefit, discrete=False)
A2 = life.A_x(0, moment=2, benefit=benefit, discrete=False)
var = A2 - A1**2   # compute variance of this life insurance
print(var)  # 0.0833
```

```
0.08333333333333331
```

But typically, the common life insurance EPV's are computed by calling the class methods `whole_life_insurance`, `term_insurance`, `deferred_insurance`, `endowment_insurance`, `increasing_insurance`, `decreasing insurance`, or `E_x` (i.e. pure endowment). It is observed in the example below that term insurance can be equivalently computed as the difference of endowment insurance and a pure endowment, or as the difference of whole life and deferred insurance.

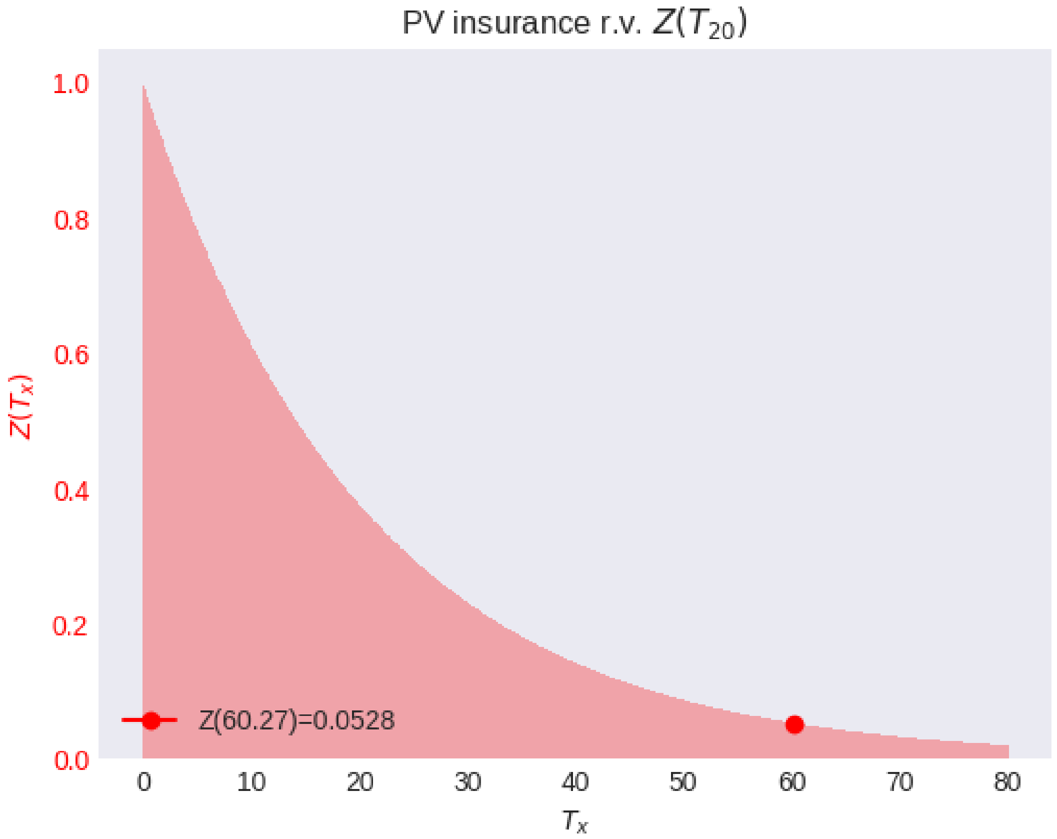
```
def S(x, s, t):  # define a survival function
    return math.exp(-.00022*t - (.0000027*1.124**(x+s)*(1.124**t-1))/math.log(1.124))
life = Insurance().set_survival(S=S, minage=20, maxage=130)\
                  .set_interest(i=0.05)
x, t = 50, 10
print(f"{t}-year endowment insurance for ({x}): ", life.endowment_insurance(x, t=t))
print(f"{t}-year pure endowment for ({x}):      ", life.E_x(x, t=t))
print(f"{t}-year term insurance for ({x}):      ", life.term_insurance(x, t=t))
print(f"{t}-year deferred insurance for ({x}):  ", life.deferred_insurance(x, u=t))
print(f"whole life insurance for ({x}):        ", life.whole_life_insurance(x))
print(f"{t}-year increasing insurance for ({x}):", life.increasing_insurance(x, t=t))
print(f"{t}-year decreasing insurance for ({x}):", life.decreasing_insurance(x, t=t))
```

```
10-year endowment insurance for (50):  0.6164284147269636
10-year pure endowment for (50):       0.6018174267001221
10-year term insurance for (50):       0.014610988026841604
10-year deferred insurance for (50):   0.1746968722738868
whole life insurance for (50):         0.1893078603007284
10-year increasing insurance for (50): 0.08665589675401192
10-year decreasing insurance for (50): 0.07406497154124572
```

The `Z_t` class method returns the $100p$-th percentile of the survival function, then `Z_from_t` can be called to discount the benefit paid at that time. The `Z_from_prob` class method performs both steps with one call. `Z_plot` plots the insurance present value r.v. Z as a function of time t.

```
def S(x, s, t):  # define a survival function
    return math.exp(-.00022*t - (.0000027*1.124**(x+s)*(1.124**t-1))/math.log(1.124))
life = Insurance().set_survival(S=S, minage=20, maxage=130).set_interest(i=0.05)
P = 0.75  # to find the 75-th percentile of Z
T = life.Z_t(x=20, prob=P, discrete=False)  # corresponding percentile of lifetime
life.Z_plot(x=20, stop=80, T=T, discrete=False)  # plot Z vs t, and annotate at time␣
↪T
print('T:', T)  # call methods and compare results
print('p:', P, life.p_x(x=20, t=T))
print('Z:', life.Z_from_prob(x=20, prob=P, discrete=False), life.Z_from_t(t=T,␣
↪discrete=False))
```

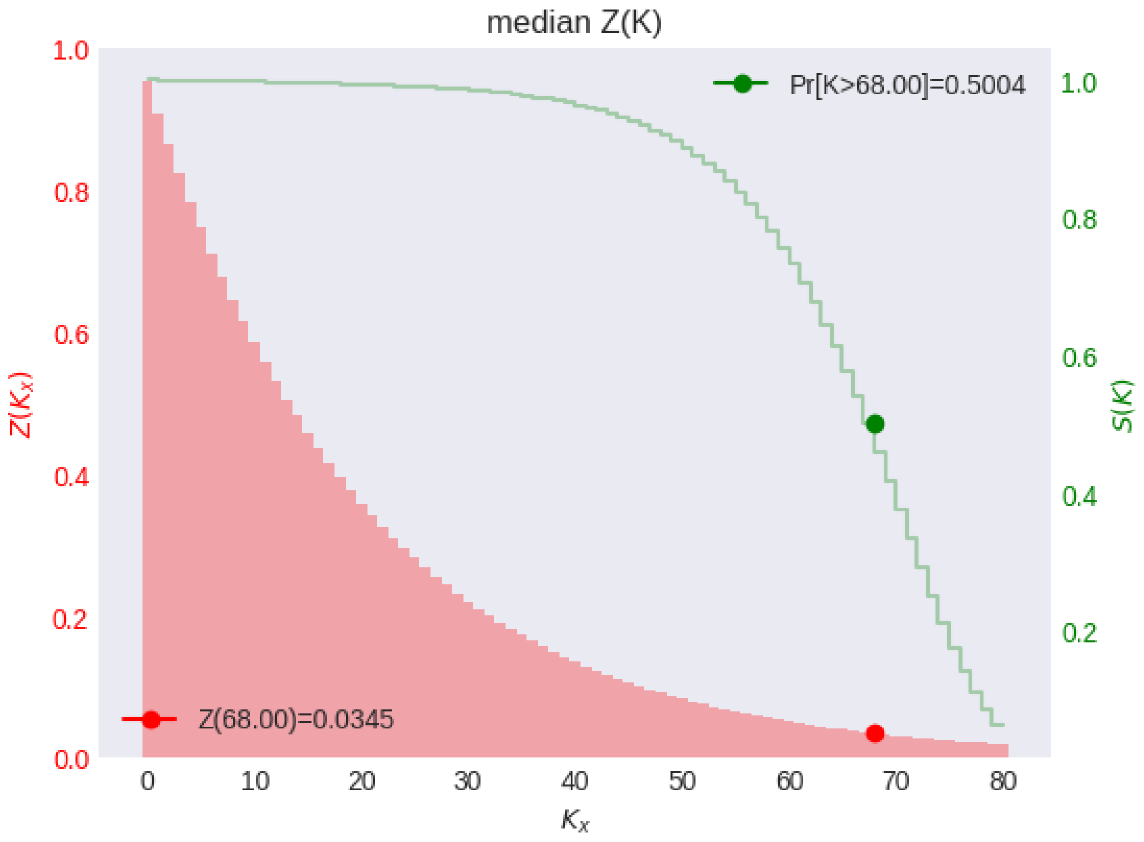
```
T: 60.274191128246535
p: 0.75 0.7500000000000304
Z: 0.05282410383956641 0.05282410383956641
```



By setting the `dual` argument to True, the survival probability of (x) can be simultaneously plotted to annotate, say, the median survival lifetime.

```python
def S(x, s, t):  # define a survival function
    return math.exp(-.00022*t - (.0000027*1.124**(x+s)*(1.124**t-1))/math.log(1.124))
life = Insurance().set_survival(S=S, minage=20, maxage=130).set_interest(i=0.05)
T = life.Z_t(x=20, prob=0.5, discrete=False)  # compute the median lifetime
K = life.Z_t(x=20, prob=0.5, discrete=True)   # plot using discrete approximation
life.Z_plot(x=20, stop=80, T=K, discrete=True, dual=True, title="median Z(K)")
print(f"S(K={K:.4f}):", life.p_x(x=20, t=K))  # survival function at integer age
print(f"S(T={T:.4f}):", life.p_r(x=20, t=T))  # survival function at fractional age
```

```
S(K=68.0000): 0.5003864946350018
S(T=68.0097): 0.49999343308443683
```



SOA Question 4.18

You are given that T, the time to first failure of an industrial robot, has a density f(t) given by

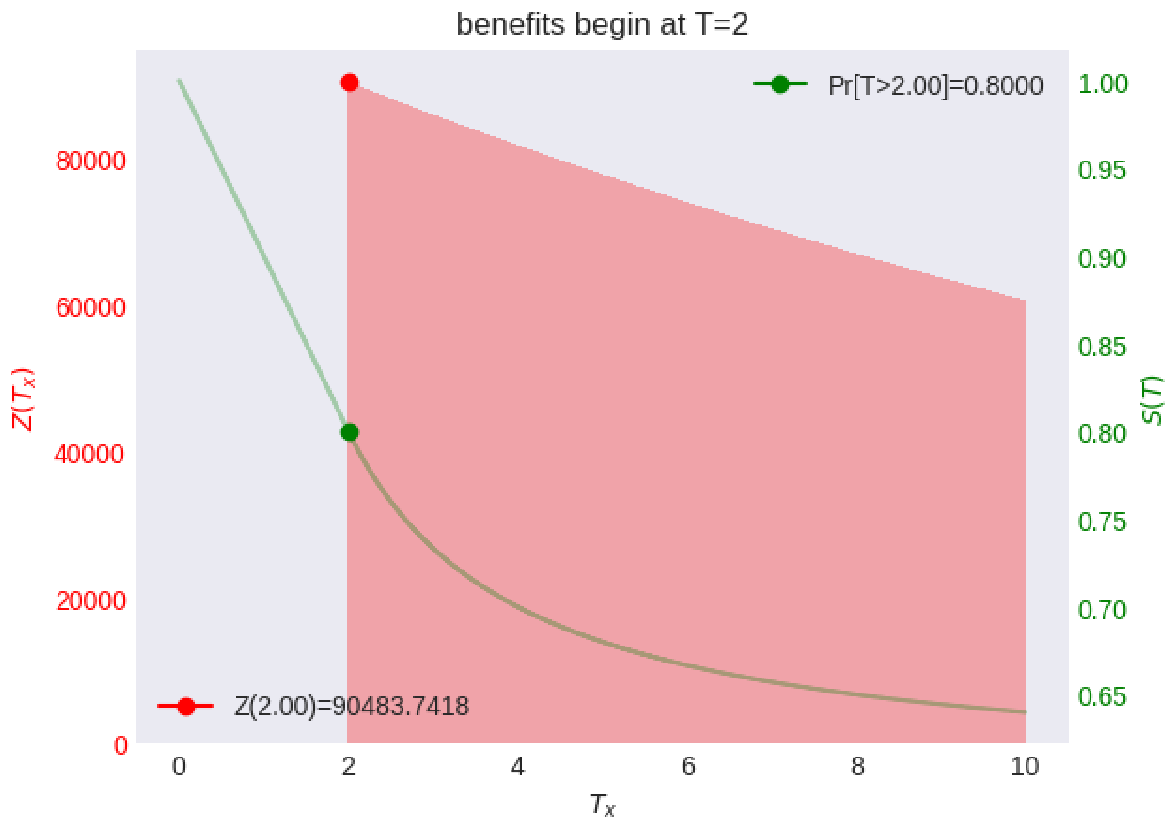
$$\begin{aligned} f(t) &= 0.1, \quad 0 \le t < 2 \\ &= 0.4t^{-2}, \quad t \le t < 10 \end{aligned}$$

with $f(t)$ undetermined on $[10, \infty)$.

Consider a supplemental warranty on this robot that pays 100,000 at the time T of its first failure if $2 \le T \le 10$, with no benefits payable otherwise. You are also given that $\delta = 5\%$. Calculate the 90th percentile of the present value of the future benefits under this warranty.

```
print("SOA Question 4.18  (A) 81873 ")
def f(x,s,t): return 0.1 if t < 2 else 0.4*t**(-2)
life = Insurance().set_interest(delta=0.05)\
                  .set_survival(f=f, maxage=10)
def benefit(x,t):
    return 0 if t < 2 else 100000
life.Z_plot(x=0, stop=10, benefit=benefit, T=2, dual=True, discrete=False,
            title='benefits begin at T=2')
prob = 0.9 - life.q_x(x=0, t=2)
T = life.Z_t(x=0, prob=prob)
Z = life.Z_from_t(T, discrete=False) * benefit(0, T)
print(T, Z)
```

```
SOA Question 4.18  (A) 81873
4 81873.07530779815
```



SOA Question 4.10

The present value random variable for an insurance policy on (x) is expressed as:

$$
\begin{aligned}
Z &= 0, \quad \text{if } T_x \leq 10 \\
&= v^T, \quad \text{if } 10 < T_x \leq 20 \\
&= 2v^T, \quad \text{if } 20 < T_x \leq 30 \\
&= 0, \quad \text{thereafter}
\end{aligned}
$$

Determine which of the following is a correct expression for $E[Z]$.

(A) ${}_{10|}\overline{A}_x + {}_{20|}\overline{A}_x - {}_{30|}\overline{A}_x$

(B) $\overline{A}_x + {}_{20}E_x\overline{A}_{x+20} - 2\ {}_{30}E_x\overline{A}_{x+30}$

(C) ${}_{10}E_x\overline{A}_x + {}_{20}E_x\overline{A}_{x+20} - 2\ {}_{30}E_x\overline{A}_{x+30}$

(D) ${}_{10}E_x\overline{A}_{x+10} + {}_{20}E_x\overline{A}_{x+20} - 2\ {}_{30}E_x\overline{A}_{x+30}$

(E) ${}_{10}E_x[\overline{A}_x + {}_{10}E_{x+10} + \overline{A}_{x+20} - {}_{10}E_{x+20} + \overline{A}_{x+30}]$

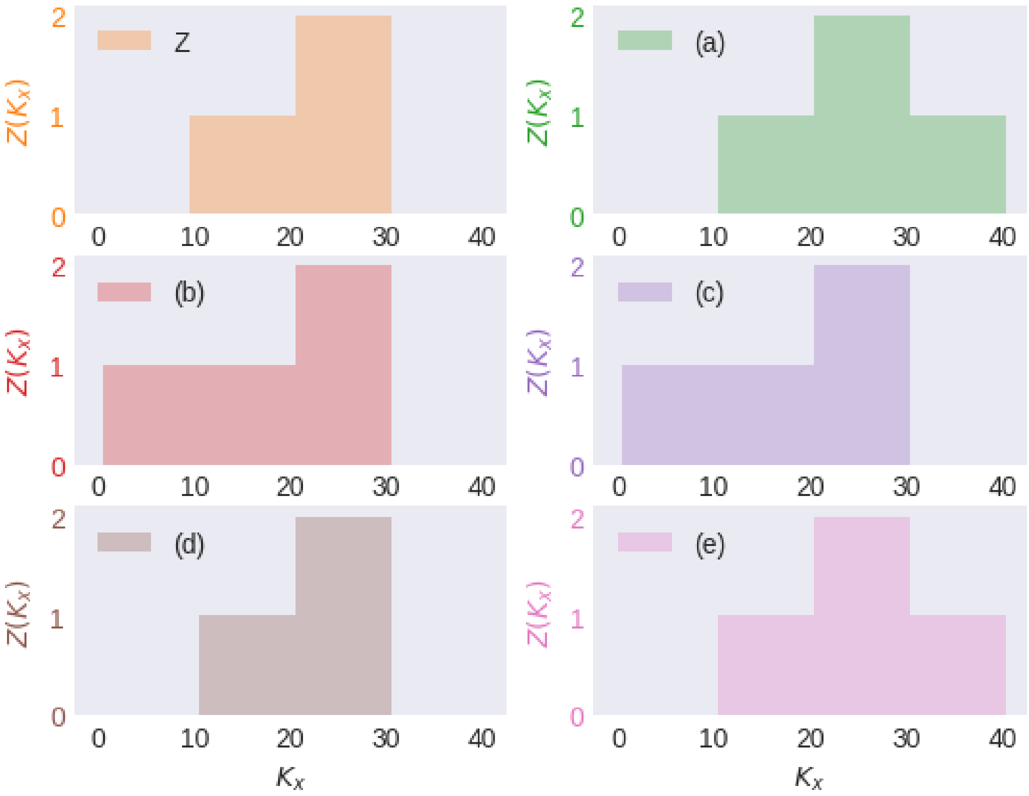
```
print("SOA Question 4.10:  (D)")
life = Insurance().set_interest(i=0)\
                  .set_survival(S=lambda x,s,t: 1, maxage=40)
def fun(x, t):  # given target
    if 10 <= t <= 20: return life.interest.v_t(t)
    elif 20 < t <= 30: return 2 * life.interest.v_t(t)
    else: return 0
def A(x, t):    # Z_x+k (t-k)
```

(continues on next page)

(continued from previous page)

```
    return life.interest.v_t(t - x) * (t > x)
x = 0
benefits=[lambda x,t: (life.E_x(x, t=10) * A(x+10, t)
                            + life.E_x(x, t=20)* A(x+20, t)
                            - life.E_x(x, t=30) * A(x+30, t)),
            lambda x,t: (A(x, t)
                            + life.E_x(x, t=20) * A(x+20, t)
                            - 2 * life.E_x(x, t=30) * A(x+30, t)),
            lambda x,t: (life.E_x(x, t=10) * A(x, t)
                            + life.E_x(x, t=20) * A(x+20, t)
                            - 2 * life.E_x(x, t=30) * A(x+30, t)),
            lambda x,t: (life.E_x(x, t=10) * A(x+10, t)
                            + life.E_x(x, t=20) * A(x+20, t)
                            - 2 * life.E_x(x, t=30) * A(x+30, t)),
            lambda x,t: (life.E_x(x, t=10)
                            * (A(x+10, t)
                            + life.E_x(x+10, t=10) * A(x+20, t)
                            - life.E_x(x+20, t=10) * A(x+30, t)))]

fig, ax = plt.subplots(3, 2)   # plot shapes of the target and the five functions
ax = ax.ravel()
for i, b in enumerate([fun] + benefits):
    life.Z_plot(0, benefit=b, ax=ax[i], color=f"C{i+1}", title='')
    ax[i].legend(["(" + "abcde"[i-1] + ")" if i else "Z"])

z = [sum(abs(b(0, t) - fun(0, t)) for t in range(40)) for b in benefits]
print("ABCDE"[np.argmin(z)])    # output letter of the function most similar to target
```

```
SOA Question 4.10:  (D)
D
```



SOA Question 4.12

For three fully discrete insurance products on the same (x), you are given:

- Z_1 is the present value random variable for a 20-year term insurance of 50
- Z_2 is the present value random variable for a 20-year deferred whole life insurance of 100
- Z_3 is the present value random variable for a whole life insurance of 100.
- $E[Z_1] = 1.65$ and $E[Z_2] = 10.75$
- $Var(Z_1) = 46.75$ and $Var(Z_2) = 50.78$

Calculate $Var(Z_3)$.

```
print("SOA Question 4.12:  (C) 167")
cov = Insurance.covariance(a=1.65, b=10.75, ab=0) # Z1 and Z2 nonoverlapping
var = Insurance.variance(a=2, b=1, var_a=46.75, var_b=50.78, cov_ab=cov)
print(var)
```

```
SOA Question 4.12:  (C) 167
166.82999999999998
```

SOA Question 4.11

You are given:

- Z_1 is the present value random variable for an n-year term insurance of 1000 issued to (x)
- Z_2 is the present value random variable for an n-year endowment insurance of 1000 issued to (x)
- For both Z_1 and Z_2 the death benefit is payable at the end of the year of death

- $E[Z_1] = 528$
- $Var(Z_2) = 15,000$
- $A_{\overset{1}{x:n|}} = 0.209$
- ${}^2A_{\overset{\ \ 1}{x:n|}} = 0.136$

Calculate $Var(Z_1)$.

```
print("SOA Question 4.11:  (A) 143385")
A1 = 528/1000   # E[Z1]  term insurance
C1 = 0.209      # E[pure_endowment]
C2 = 0.136      # E[pure_endowment^2]
def fun(A2):
    B1 = A1 + C1   # endowment = term + pure_endowment
    B2 = A2 + C2   # double force of interest
    return Insurance.insurance_variance(A2=B2, A1=B1)
A2 = Insurance.solve(fun, target=15000/(1000*1000), grid=[143400, 279300])
var = Insurance.insurance_variance(A2=A2, A1=A1, b=1000)
print(var)
```

```
SOA Question 4.11:  (A) 143385
143384.99999999997
```

SOA Question 4.15

For a special whole life insurance on (x), you are given:

- Death benefits are payable at the moment of death
- The death benefit at time t is $b_t = e^{0.02t}$, for $t \geq 0$
- $\mu_{x+t} = 0.04$, for $t \geq 0$
- $\delta = 0.06$
- Z is the present value at issue random variable for this insurance.

Calculate $Var(Z)$.

```
print("SOA Question 4.15  (E) 0.0833 ")
life = Insurance().set_survival(mu=lambda *x: 0.04)\
                  .set_interest(delta=0.06)
benefit = lambda x,t: math.exp(0.02*t)
A1 = life.A_x(0, benefit=benefit, discrete=False)
A2 = life.A_x(0, moment=2, benefit=benefit, discrete=False)
var = A2 - A1**2
print(var)
```

```
SOA Question 4.15  (E) 0.0833
0.08333333333333331
```

SOA Question 4.4

For a special increasing whole life insurance on (40), payable at the moment of death, you are given:

- The death benefit at time t is $b_t = 1 + 0.2t, \quad t \geq 0$
- The interest discount factor at time t is $v(t) = (1 + 0.2t) - 2, \quad t \geq 0$

- ${}_tp_{40}\mu_{40+t} = 0.025$ if $0 \leq t < 40$, otherwise 0
- Z is the present value random variable for this insurance Calculate Var(Z).

```
print("SOA Question 4.4  (A) 0.036")
x = 40
life = Insurance().set_survival(f=lambda *x: 0.025, maxage=x+40)\
                  .set_interest(v_t=lambda t: (1 + .2*t)**(-2))
benefit = lambda x,t: 1 + .2 * t
A1 = life.A_x(x, benefit=benefit, discrete=False)
A2 = life.A_x(x, moment=2, benefit=benefit, discrete=False)
var = A2 - A1**2
print(var)
```

```
SOA Question 4.4  (A) 0.036
0.03567680106032681
```

CHAPTER
EIGHT

ANNUITIES

A life annuity is a regular sequence of payments as long as the annuitant is alive on the payment date.

8.1 Present value of life annuity r.v. Y

Valuation functions for the present value of annuity benefits, denoted by Y, are based on the continuous future lifetime random variable, T_x, or the curtate future lifetime random variable, K_x. The expected present value of annuity benefits is denoted and solved by EPV(Y). A useful formula from interest theory is that for a T-year continuous annuity-certain and annual annuity-due – $\overline{a}_{\overline{T|}} = \dfrac{1-v^T}{\delta}$ and $\ddot{a}_{\overline{T|}} = \dfrac{1-v^T}{d}$ – which are used to calculate the actuarial present value of life annuities.

8.2 Whole life annuity

A whole life annuity pays a level benefit until the death of the annuitant

$Y = \overline{a}_{\overline{T_x|}} = \dfrac{1-v^{T_x}}{\delta}$

- present value random variable for continuous life annuity that pays at a rate of 1 per year until the moment of death T_x

$\overline{a}_x = EPV[\overline{a}_{\overline{T_x|}}] = \int_{t=0}^{\infty} v^t \ {}_tp_x \ dt$

- expected present value of continuous whole life annuity

$\overline{a}_x = \overset{\circ}{e}_x$

- special case when interest rate is 0

$Y = \ddot{a}_{\overline{K_x+1|}} = \dfrac{1-v^{K_x+1}}{d}$

- present value random variable for annual life annuity due that pays 1 up to the beginning of the year of death

$\ddot{a}_x = EPV[\ddot{a}_{\overline{K_x+1|}}] = \sum_{k=0}^{\infty} v^k \ {}_kp_x$

- expected present value of annual whole life annuity due

$\ddot{a}_x = 1 + e_x$

- special case when interest rate is 0

8.3 Temporary annuity

A t-year temporary life year annuity pays a level benefit until the earlier of death and year t

$Y = \bar{a}_{\overline{t|}}$ when or $T_x > t$, else $Y = \bar{a}_{\overline{T_x|}}$

- present value random variable for continuous temporary life annuity that pays at a rate of 1 until the moment of death, or t if earlier

$\bar{a}_{x:\overline{t|}} = \int_{s=0}^{t} v^s \, {}_sp_x \, ds = \bar{a}_x - \, {}_tE_x \, \bar{a}_{x+t}$

- expected present value of continuous temporary life annuity

$\bar{a}_{x:\overline{t|}} = \mathring{e}_{x:\overline{t|}}$

- when interest rate is 0

$Y = \ddot{a}_{\overline{t|}}$ when $K_x \geq t$, else $Y = \ddot{a}_{\overline{K_x+1|}}$

- present value random variable for annual life annuity that pays 1 at the beginning of the year until the moment of death, or t if earlier.

$\ddot{a}_{x:\overline{t|}} = \sum_{k=0}^{t-1} v^k \, {}_{k|}p_x = \ddot{a}_x - \, {}_tE_x \, \ddot{a}_{x+t}$

- expected present value of annual temporary life annuity due

$\ddot{a}_{x:\overline{t|}} = 1 + e_{x:\overline{t|}} - \, {}_tp_x$

- when interest rate is 0

8.4 Deferred whole life annuity

A n-year deferred life annuity pays a level benefit starting at time n and continuing until death.

${}_{u|}\bar{a}_x = \bar{a}_x - \bar{a}_{x+u}$

- continuous deferred life annuity as the difference of whole life annuities

${}_{u|}\ddot{a}_x = \ddot{a}_x - \ddot{a}_{x+u}$

- annual deferred annuity due as the difference of annual whole life annuities due

8.5 Certain and life annuity

A common feature of pension benefits is that the pension annuity is guaranteed to be paid for some period even if the annuitant dies before the end of the period.

$Y = \bar{a}_{\overline{n|}}$ when $T_x \leq n$, else $\bar{a}_{\overline{T_x|}}$

- present value random variable for continuous certain and life annuity

$\bar{a}_{\overline{x:\overline{n|}}} = \bar{a}_{\overline{n|}} + \, {}_{n|}\bar{a}_x$

- can be computed as a continuous temporary certain annuity plus a deferred continuous whole life annuity

$Y = \ddot{a}_{\overline{n|}}$ when $K_x < n$, else $\ddot{a}_{\overline{K_x+1|}}$

- present value random variable for annual certain and life annuity due

$\ddot{a}_{\overline{x:\overline{n|}}} = \ddot{a}_{\overline{n|}} + \, {}_{n|}\ddot{a}_x$

- can be computed as an annual temporary certain annuity due, plus a deferred annual life annuity due

8.6 Life insurance twin

Whole and *Temporary Life Annuities* (and *Whole Life* and *Endowment Insurance*) **ONLY**:

$$\bar{a}_x = \frac{1 - \overline{A}_x}{\delta}$$

$$\overline{A}_x = 1 - \delta \, \bar{a}_x$$

- continuous whole life insurance twin for continuous whole life annuity

$$\ddot{a}_x = \frac{1 - A_x}{d}$$

$$A_x = 1 - d \, \ddot{a}_x$$

- annual whole life insurance twin for annual life annuity due

$$\bar{a}_{x:\overline{t}|} = \frac{1 - \overline{A}_{x:\overline{t}|}}{\delta}$$

$$\overline{A}_{x:\overline{t}|} = 1 - \delta \, \bar{a}_{x:\overline{t}|}$$

- continuous endowment insurance twin for continuous temporary life annuity

$$\ddot{a}_{x:\overline{t}|} = \frac{1 - A_{x:\overline{t}|}}{d}$$

$$A_{x:\overline{t}|} = 1 - d \, \ddot{a}_{x:\overline{t}|}$$

- annual endowment insurance twin for annual temporary life annuity due

8.7 Variances

The second moment of a life annuity is not easy to compute directly. However, for whole life and temporary life annuities, the variance can be calculated directly using the insurance twin equations.

Whole life annuity

$${}^2\bar{a}_x = \frac{1 - {}^2\overline{A}_x}{2\delta}$$

$${}^2\overline{A}_x = 1 - (2\delta) \, {}^2\bar{a}_x$$

$$Var(\bar{a}_x) = \frac{{}^2\overline{A}_x - (\overline{A}_x)^2}{\delta^2}$$

- from doubling the force of interest of continuous whole life insurance

$${}^2\ddot{a}_x = \frac{1 - {}^2A_x}{2d - d^2}$$

$${}^2A_x = 1 - (2d - d^2) \, {}^2\ddot{a}_x$$

$$Var(\ddot{a}_x) = \frac{{}^2\ddot{A}_x - \, (A_x)^2}{d^2}$$

- from doubling the force of interest of annual whole life insurance

Temporary life annuity

$^2\overline{a}_{x:\overline{t|}} = \dfrac{1 - {}^2\overline{A}_{x:\overline{t|}}}{2\delta}$

$^2\overline{A}_{x:\overline{t|}} = 1 - (2\delta)\ {}^2\overline{a}_{x:\overline{t|}}$

$Var(\overline{a}_{x:\overline{t|}}) = \dfrac{{}^2\overline{A}_{x:\overline{t|}} - (\overline{A}_{x:\overline{t|}})^2}{\delta^2}$

- from doubling the force of interest of continuous endowment insurance

$^2\ddot{a}_{x:\overline{t|}} = \dfrac{1 - {}^2A_{x:\overline{t|}}}{2d - d^2}$

$^2A_{x:\overline{t|}} = 1 - (2d - d^2)\ {}^2\ddot{a}_{x:\overline{t|}}$

$Var(\ddot{a}_{x:\overline{t|}}) = \dfrac{{}^2A_{x:\overline{t|}} -\ (A_{x:\overline{t|}})^2}{d^2}$

- from doubling the force of interest of annual endowment insurance

8.8 Immediate life annuity

Benefit is paid at the *end of the year*, as long as the annuitant is alive, and can be related to the value of an annual life annuity due:

$a_x = \ddot{a}_x - 1$

- whole life annuities

$a_{x:\overline{t|}} = \ddot{a}_{x:\overline{t|}} - 1 +\ {}_tE_x$

- temporary life annuities

8.9 Varying life annuities

Increasing annuity:

The amount of the annuity payment increases arithmetically with time.

$(\overline{I}\overline{a})_x = \int_{t=0}^{\infty}\ t\ v^t\ {}_tp_x\ dt$

- EPV of continuously increasing whole life annuity

$(I\ddot{a})_x = \sum_{k=0}^{\infty}\ (k+1)\ v^{k+1}\ {}_kp_x$

- EPV of annual increasing whole life annuity due

$(\overline{I}\overline{a})_{x:\overline{t|}} = \int_{s=0}^{t}\ s\ v^s\ {}_sp_x\ ds$

- EPV of continuously increasing temporary life annuity

$(I\ddot{a})_{x:\overline{t|}} = \sum_{k=0}^{t-1}\ (k+1)\ v^{k+1}\ {}_kp_x$

- EPV of annual increasing temporary life annuity due

Decreasing annuity:

The amount of the annuity payment decreases arithmetically with time. There are no decreasing whole life annuities.

$(\overline{Da})_{x:\overline{t|}} = \int_{s=0}^{t} ~(t-s)~ v^s ~_sp_x~ ds$

- EPV of continuously decreasing temporary life annuity (not defined for whole life)

$(D\ddot{a})_{x:\overline{t|}} = \sum_{k=0}^{t-1} ~(t-k)~ v^{k+1} ~_kp_x$

- EPV of annual decreasing temporary life annuity due (not defined for whole life)

Identity relationships:

$(\overline{Da})_{x:\overline{t|}} + (\overline{Ia})_{x:\overline{t|}} = t~\overline{a}_{x:\overline{t|}}$

- the sum of continuous t-year decreasing and increasing life annuities is t times a t-year temporary annuity

$(D\ddot{a})_{x:\overline{t|}} + (I\ddot{a})_{x:\overline{t|}} = (t+1)~\ddot{a}_{x:\overline{t|}}$

- the sum of t-year annual decreasing and increasing life annuities due is $t+1$ times a t-year temporary annuity

8.10 Probabilities and percentiles

The value of annuity random variables always increases as a function of survival time. Hence calculating a percentile of a life annuity simply requires calculating the percentile of T, then calculating the value of a temporary annuity.

Percentile of Y:

$t_{\pi} = S^{-1}(1 - \pi)$

- the $100p$-th percentile of the lifetime variable for whole life annuity can be obtained from looking up the inverse of the survival function at probability p.

$Y_{\pi} = b~\overline{a}_{\overline{t_{\pi}|}} = b~\dfrac{1 - v^{t_{\pi}}}{\delta}$

- the PV of the annuity r.v. that pays level benefit b up to time t_{π}

Probability of Y:

Since annuity values monotonically increases with survaval time, to calculate he probability that the present value of an annuity is greater than some value $Pr(Y \ge Y_{\pi})$, calculate the time t_{π} for which $\overline{a}_{\overline{t_{\pi}|}}$ has the required value, then calculate the probability that time of death T_x is in that range.

8.11 Methods

The `Annuity` class implements methods to compute the expected present value of life annuities

```python
from actuarialmath import Annuity
import math
import describe
describe.methods(Annuity)
```

```
class Annuity - Compute present values and relationships of life annuities

    Methods:
    --------

    a_x(x, s, t, u, benefit, discrete):
      Compute EPV of life annuity from survival function
```

(continues on next page)

(continued from previous page)

```
    immediate_annuity(x, s, t, b, variance):
      Compute EPV of immediate life annuity

    annuity_twin(A, discrete):
      Returns annuity from its WL or Endowment Insurance twin"

    insurance_twin(a, moment, discrete):
      Returns WL or Endowment Insurance twin from annuity

    annuity_variance(A2, A1, b, discrete):
      Compute variance from WL or endowment insurance twin

    whole_life_annuity(x, s, b, variance, discrete):
      Whole life annuity: a_x

    temporary_annuity(x, s, t, b, variance, discrete):
      Temporary life annuity: a_x:t

    deferred_annuity(x, s, u, t, b, discrete):
      Deferred life annuity n|t_a_x =  n+t_a_x - n_a_x

    certain_life_annuity(x, s, u, t, b, discrete):
      Certain and life annuity = certain + deferred

    increasing_annuity(x, s, t, b, discrete):
      Increasing annuity

    decreasing_annuity(x, s, t, b, discrete):
      Identity (Da)_x:n + (Ia)_x:n = (n+1) a_x:n temporary annuity

    Y_x(x, s, t, discrete):
      EPV of year t annuity benefit for life aged [x]+s: b_x[s]+s(t)

    Y_t(x, prob, discrete):
      T_x given percentile of the r.v. Y = PV of WL or Temporary Annuity

    Y_from_t(t, discrete):
      PV of insurance payment Y(t), given T_x

    Y_from_prob(x, prob, discrete):
      Percentile of annual or continuous WL annuity PV r.v. Y, given probability

    Y_plot(x, s, stop, benefit, T, discrete, ax, dual, title, color, alpha):
      Plot PV of annuity r.v. Y vs T
```

8.12 Examples

Given a survival model, the `a_x` class method computes the EPV for arbitrary life annuity benefits, discount functions, and terms using numerical integration or summation. Common life annuity EPV's are computed by calling the class methods `whole_life_annuity`, `temporary_annuity`, `deferred_annuity`, `certain_life_annuity`, `increasing_insurance`, `decreasing insurance`, or `immediate_annuity`. The examples below confirm the numerical relationships of

- whole life immediate annuity with annuity-due
- whole life with temporary and deferred annuities
- certain-and-life annuity with deferred life annuity and annuity-certain
- temporary life annuity with increasing and decreasing life annuities

```
def S(x, s, t):  # define a survival function
    return math.exp(-.00022*t - (.0000027*1.124**(x+s)*(1.124**t-1))/math.log(1.124))
life = Annuity().set_survival(S=S, minage=20, maxage=130)\
                .set_interest(i=0.05)
x, t = 50, 10
print(f"whole life annuity due for ({x}):           ", life.whole_life_annuity(x))
print(f"immediate whole life annuity for ({x}):     ", life.immediate_annuity(x))
print(f"{t}-year certain-and-life annuity for ({x}):", life.certain_life_annuity(x,␣
↪u=t))
print(f"{t}-year deferred life annuity for ({x}):    ", life.deferred_annuity(x, u=t))
print(f"{t}-year annuity-certain for ({x}):           ", life.interest.annuity(t=t))
print(f"{t}-year increasing insurance for ({x}):     ", life.increasing_annuity(x,␣
↪t=t))
print(f"{t}-year decreasing insurance for ({x}):     ", life.decreasing_annuity(x,␣
↪t=t))
print(f"{t}-year temporary annuity for ({x}):         ", life.temporary_annuity(x, t=t))
print(f"{t//2}-year temporary annuity for ({x}):         ", life.temporary_annuity(x,␣
↪t=t//2))
print(f"{t//2}-year deferred {t//2}-year temporary for ({x}):",
      life.a_x(x, u=t//2, t=t//2))
```

```
whole life annuity for (50):             17.024534933684702
immediate whole life annuity for (50):   16.024534933684702
10-year certain-and-life annuity for (50): 17.077353318594994
10-year deferred annuity for (50):       8.96953164295094
10-year annuity-certain for (50):        8.107821675644052
10-year increasing insurance for (50):   40.9536356665489
10-year decreasing insurance for (50):   47.65140053152248
10-year temporary annuity for (50):      8.055003290733762
5-year temporary annuity for (50):       4.534416095954249
5-year deferred 5-year temporary for (50): 3.520587194779506
```

The `Y_t` class method returns the 100p-th percentile of the survival function, then `Y_from_t` can be called to discount the benefits up to that time. The `Y_from_prob` class method performs both steps with one call. `Y_plot plots the annuity present value r.v. as a function of time.

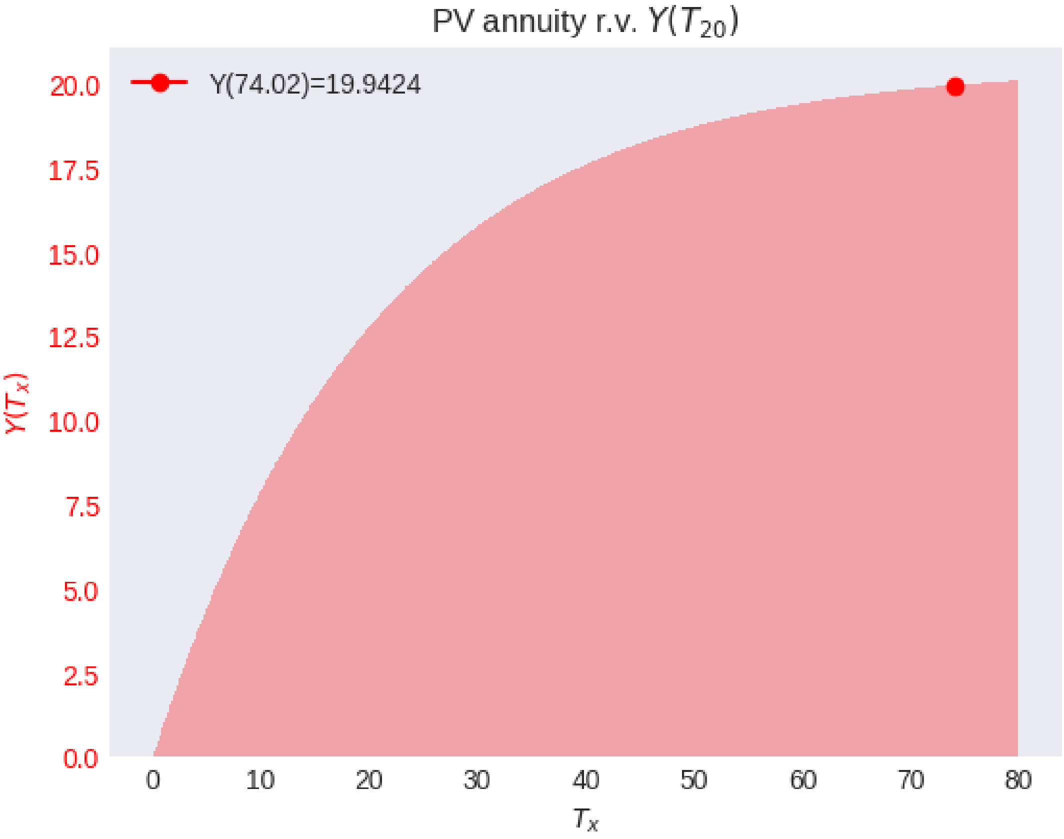
```
def S(x, s, t):  # define a survival function
    return math.exp(-.00022*t - (.0000027*1.124**(x+s)*(1.124**t-1))/math.log(1.124))
life = Annuity().set_survival(S=S, minage=20, maxage=130)\
                .set_interest(i=0.05)
```

(continues on next page)

(continued from previous page)

```
P = 0.75  # to find the 75-th percentile of Y
T = life.Y_t(x=20, prob=P, discrete=False)  # corresponding percentile of lifetime
life.Y_plot(x=20, stop=80, T=T, discrete=False)  # plot Y vs t, and annotate at time␣
↪T
print('T:', T)   # call methods and compare results
print('p:', P, life.q_x(x=20, t=T))
print('Y:', life.Y_from_prob(x=20, prob=P, discrete=False), life.Y_from_t(t=T,␣
↪discrete=False))
```

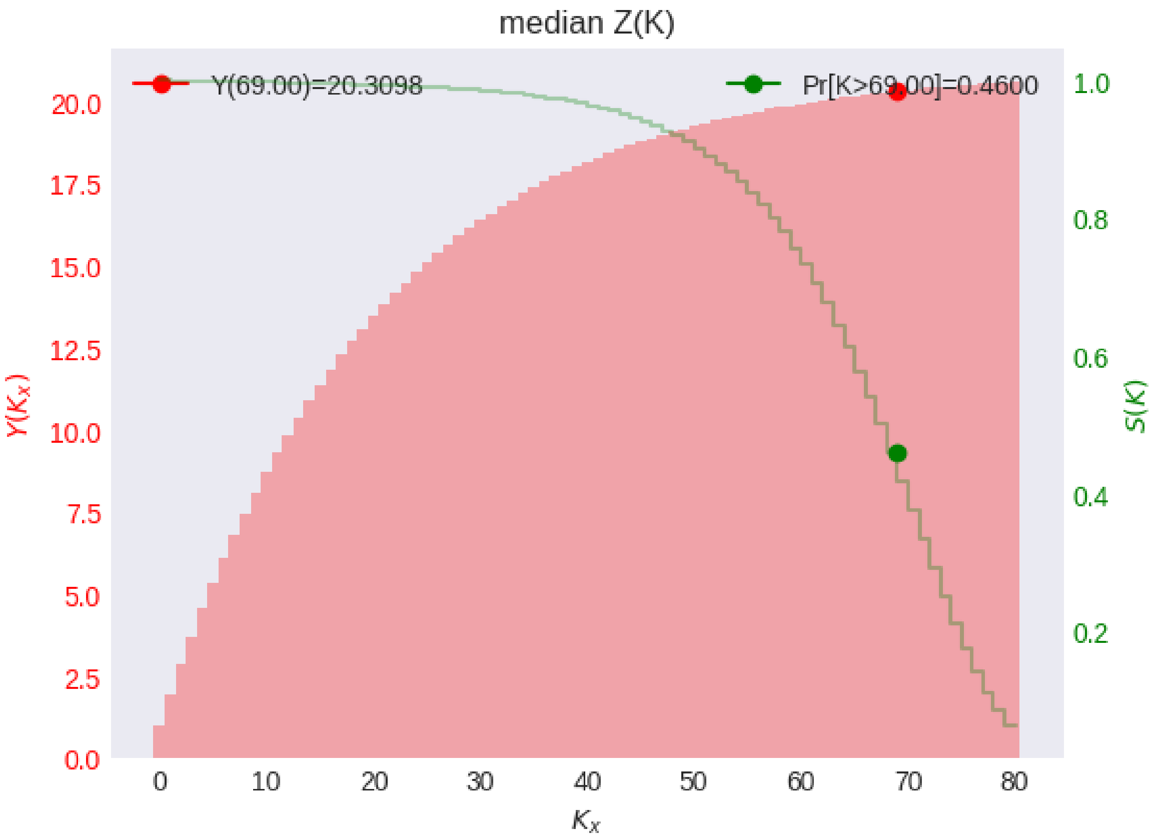
```
T: 74.02351166727799
p: 0.75 0.7499999999997666
Y: 19.94237852945247 19.94237852945247
```



By setting the `dual`` argument to True, the survival probability of (x) can be simultaneously plotted to annotate, say, the median survival lifetime.

```
def S(x, s, t):  # define a survival function
    return math.exp(-.00022*t - (.0000027*1.124**(x+s)*(1.124**t-1))/math.log(1.124))
life = Annuity().set_survival(S=S, minage=20, maxage=130)\
                .set_interest(i=0.05)
T = life.Y_t(x=20, prob=0.5, discrete=False)  # compute the median lifetime
K = life.Y_t(x=20, prob=0.5, discrete=True)   # plot using discrete approximation
life.Y_plot(x=20, stop=80, T=K, discrete=True, dual=True, title="median Z(K)")
print(f"S(K={K:.4f}):", life.p_x(x=20, t=K))  # survival function at integer age
print(f"S(T={T:.4f}):", life.p_r(x=20, t=T))  # survival function at fractional age
```

```
S(K=69.0000): 0.45995640909219604
S(T=68.0097): 0.49999343308443683
```



SOA Question 5.6

For a group of 100 lives age x with independent future lifetimes, you are given:

- Each life is to be paid 1 at the beginning of each year, if alive
- $A_x = 0.45$
- ${}^2A_x = 0.22$
- $i = 0.05$
- Y is the present value random variable of the aggregate payments.

Using the normal approximation to Y, calculate the initial size of the fund needed to be 95% certain of being able to make the payments for these life annuities.

```
print("SOA Question 5.6:  (D) 1200")
life = Annuity().set_interest(i=0.05)
var = life.annuity_variance(A2=0.22, A1=0.45)
mean = life.annuity_twin(A=0.45)
print(life.portfolio_percentile(mean=mean, variance=var, prob=.95, N=100))
```

```
SOA Question 5.6:  (D) 1200
1200.6946732201702
```

CHAPTER

NINE

PREMIUMS

When insurance company agrees to pay some life contingent benefits, the policyholder agrees to pay premiums to the insurance company to secure these benefits. The premiums will also need to reimburse the insurance company for the expenses associated with the policy.

The calculation of the premium may not explicitly allow for the insurance company's expenses, which we refer to as a net premium (also called a risk premium or benefit premium). If the calculation does explicitly allow for expenses, the premium is called a gross premium (also called expense-loaded premium).

Premiums for life insurance are payable in advance, with the first premium payable when the policy is purchased.

- *Fully continuous insurance*: both benefits and premiums are payable continuously
- *Fully discrete insurance*: benefits are paid at the end of the year, premiums are paid at the beginning of the year
- *Semi-continuous insurance*: benefits are paid at moment of death, premiums are paid at the beginning of the year

9.1 Present value of loss at issue r.v. ${}_0L$

The loss at issue random variable is defined as the present value of the amount the insurance loses.

${}_0L$ = PV of future benefits - PV of future premiums

- PV of loss at issue

Fully continuous whole life insurance:

$${}_0L = v^{T_x} - P\overline{a}_{\overline{T_x|}} = v^{T_x} - P(\frac{1 - v^{T_x}}{\delta})$$

- loss at issue is the present value of the insurance payment less the present value of the level premimum annuity of P per year

$${}_0L = \overline{A}_x - P\overline{a}_x = \overline{A}_x(1 + \frac{P}{\delta}) - \frac{\pi}{\delta}$$

- expected loss at issue by applying expected values to the loss at issue equation

9.2 Equivalence principle

Under this principle, premiums are set such that the actuarial present value of the benefit premiums equals the actuarial present value of the benefits, hence expected loss at issue is zero:

$$E[{}_0L] = EPV_0(\text{future benefits}) - EPV_0(\text{future premiums}) = 0$$

9.3 Net premium

For net premiums, we take into consideration outgoing benefit payments only: expenses are not a part of the calculation. The benefit may be a death benefit or a survival benefit or a combination. Under the equivalence principle, the net premium is set such that the expected value of the future loss is zero at the start of the contract, $E[L_0] = 0$.

Whole life insurance:

$$P_x = \frac{A_x}{\ddot{a}_x}$$

- fully discrete whole life insurance net premium

$$\overline{P}_x = \frac{\overline{A}_x}{\overline{a}_x}$$

- fully continuous whole life insurance net premium

Term life insurance:

$$P^1_{x:\overline{t|}} = \frac{A^1_{x:\overline{t|}}}{\ddot{a}_{x:\overline{t|}}}$$

- fully discrete term life net premium

$$\overline{P}^1_{x:\overline{t|}} = \frac{\overline{A}^1_{x:\overline{t|}}}{\overline{a}_{x:\overline{t|}}}$$

- fully continuous term life net premium

Pure endowment:

$$P_{x:\overline{t|}}^{\;\;\;1} = \frac{{}_tE_x}{\ddot{a}_{x:\overline{t|}}}$$

- fully discrete pure endowment net premium

$$\overline{P}_{x:\overline{t|}}^{\;\;\;1} = \frac{{}_tE_x}{\overline{a}_{x:\overline{t|}}}$$

- fully continuous pure endowment net premium

Endowment insurance:

$$P_{x:\overline{t|}} = \frac{A_{x:\overline{t|}}}{\ddot{a}_{x:\overline{t|}}}$$

- fully discrete endowment insurance net premium

$$\overline{P}_{x:\overline{t|}} = \frac{\overline{A}_{x:\overline{t|}}}{\overline{a}_{x:\overline{t|}}}$$

- fully continuous endowment insurance net premium

Shortcuts for whole life and endowment insurance only:

For whole life and endowment insurance only, by plugging in the insurance or annuity twin, the following shortcuts are available for calculating net premiums from only the life insurance or annuity factor.

$$P_x = b\,(\frac{1}{\ddot{a}_x} - d) = b\,(\frac{dA_x}{1 - A_x})$$

- fully discrete whole life shortcut

$$\overline{P}_x = b\,(\frac{1}{\overline{a}_x} - \delta) = b\,(\frac{d\overline{A}_x}{1 - \overline{A}_x})$$

- fully continuous whole life shortcut

$$P_{x:\overline{t|}} = b\,(\frac{1}{\ddot{a}_{x:\overline{n|}}} - d) = b\,(\frac{dA_{x:\overline{n|}}}{1 - A_{x:\overline{n|}}})$$

- fully discrete endowment insurance shortcut

$$\overline{P}_{x:\overline{t|}} = b\,(\frac{1}{\overline{a}_{x:\overline{n|}}} - \delta) = b\,(\frac{d\overline{A}_{x:\overline{n|}}}{1 - \overline{A}_{x:\overline{n|}}})$$

- fully continuous endowment insurance shortcut

9.4 Gross premium

When we calculate a gross premium for an insurance policy or an annuity, we take account of the expenses the insurer incurs. There are three main types of expense associated with policies – initial expenses, renewal expenses and termination or claim expenses.

Expenses:

$e_i =$ initial_per_policy + initial_per_premium × gross_premium

- initial expenses at the beginning of year 1 when a policy is issued, which may be both proportional to premiums or may be 'per policy', meaning that the amount is fixed for all policies, and is not related to the size of the contract.

$e_r =$ renewal_per_policy + renewal_per_premium × gross_premium

- renewal expenses in the beginning of each year 2+, may be both per policy or percent of premium.

$E =$ settlement expense

- is paid along with death benefit (b); hence total claim cost upon death is:

  claim cost $= b + E =$ death benefit + settlement expense.

Return of premiums paid without interest upon death:

$$EPV_0(\text{return of premiums paid}) = \sum_{k=0}^{t-1}\, P(k+1)\, v^{k+1}\ {}_{k|}q_x = P\cdot(IA)^1_{x:\overline{t|}}$$

- an additional benefit in some insurance policies, whose EPV can be calculated using an increasing insurance factor

Equivalence principle:

If gross premiums are set under equivalence principle, then expected gross future loss at issue equals zero:

$$E[{}_0L^g] = EPV_0(\text{future benefits}) + EPV_0(\text{future expenses}) - EPV_0(\text{future premiums}) = 0$$

9.5 Portfolio Percentile Premium

The portfolio percentile premium principle is an alternative to the equivalence premium principle. Using the mean and variance of the future loss random variable, the portfolio percentile premium principle can be used to determine a premium. We assume a large portfolio of N identical and independent policies. The present value of the total future loss $\overline{L}$ of the portfolio can be approximated by a normal distribution over the sum of the individual losses L_i

$$\begin{aligned}\overline{L} &= L_1 + L_2 + \cdots + L_N \\ E[\overline{L}] &= N\, E[L] \\ Var[\overline{L}] &= N\, Var[L]\end{aligned}$$

Note $E[\overline{L}]$ and $Var[\overline{L}]$ are functions of the unspecified premium P. A probability percentile q (say, 95% confidence) and a threshold L^* (say, 0) are chosen, then P is solved for implicitly from the following equation, such that the probability of $\overline{L}$ not exceeding L^* is q

$$Pr[\overline{L} < L^*] = Pr\left[\frac{\overline{L} - E[\overline{L}]}{\sqrt{Var[\overline{L}]}} < \frac{L^* - E[\overline{L}]}{\sqrt{Var[\overline{L}]}}\right] = \Phi\left(\frac{L^* - E[\overline{L}]}{\sqrt{Var[\overline{L}]}}\right) = q$$

9.6 Methods

The `Premiums` class implements methods for computing net and gross premiums under the equivalence principle

```python
import numpy as np
from actuarialmath import Premiums
import describe
describe.methods(Premiums)
```

```
class Premiums - Compute et and gross premiums under equivalence principle

    Methods:
    --------

    net_premium(x, s, t, u, n, b, endowment, discrete, return_premium, annuity,␣
 ↪initial_cost):
      Net level premium for special n-pay, u-deferred t-year term insurance

    insurance_equivalence(premium, b, discrete):
      Compute whole life or endowment insurance factor, given net premium

    annuity_equivalence(premium, b, discrete):
      Compute whole life or temporary annuity factor, given net premium

    premium_equivalence(A, a, b, discrete):
      Compute premium from whole life or endowment insurance and annuity factors

    gross_premium(a, A, IA, discrete, benefit, E, endowment, settlement_policy,␣
 ↪initial_policy, initial_premium, renewal_policy, renewal_premium):
      Gross premium by equivalence principle
```

9.7 Examples

When net premiums are set by the equivalence principle, then the three class methods `insurance_equivalence`, `annuity_equivalence` or `premium_equivalence` can be called to compute the insurance, annuity or net premium respectively given any one of the other value, for whole life and endowment insurances. For other general life insurance or annuity benefits, `net_premium` computes the net premium under EPP given any term (`t`), deferral period (`u`), endowment benefit amount (`endowment`), other initial expected costs (`initial_cost`), or refund of premium without interest at death (`return_premium`).

```
life = Premiums().set_interest(delta=0.06)\
                 .set_survival(mu=lambda x,s: 0.04)
P = life.net_premium(x=0, discrete=False)
A = life.whole_life_insurance(x=0, discrete=False)
a = life.whole_life_annuity(x=0, discrete=False)
print('Insurance:', A, life.insurance_equivalence(premium=P, discrete=False))
print('Annuity:', a, life.annuity_equivalence(premium=P, discrete=False))
print('Net Premium:', P,
      life.premium_equivalence(A=A, discrete=False),
      life.premium_equivalence(a=a, discrete=False))
```

```
Insurance: 0.4 0.4
Annuity: 9.999999999999996 9.999999999999996
Net Premium: 0.040000000000000015 0.040000000000000015 0.040000000000000015
```

The `gross_premium` class method computes the level premium under the equivalence principles given expense and claim amounts (initial and renewal expenses per policy or per $1 premium, settlement expense, death benefit, and endowment) and corresponding actuarial present value factors (`A` for insurance benefits, `a` for premium annuity, `IA` for refund of premium without interest death benefit, and `E` for endowment benefit).

SOA Question 6.2

For a fully discrete 10-year term life insurance policy on (x), you are given:

- Death benefits are 100,000 plus the return of all gross premiums paid without interest
- Expenses are 50% of the first year's gross premium, 5% of renewal gross premiums and 200 per policy expenses each year
- Expenses are payable at the beginning of the year
- $A^{1}_{x:\overline{10|}} = 0.17094$
- $(IA)^{1}_{x:\overline{10|}} = 0.96728$
- $\ddot{a}_{x:\overline{10|}} = 6.8865$

Calculate the gross premium using the equivalence principle.

```
print("SOA Question 6.2: (E) 3604")
life = Premiums()
A, IA, a = 0.17094, 0.96728, 6.8865
print(life.gross_premium(a=a,
                         A=A,
                         IA=IA,
                         benefit=100000,
                         initial_premium=0.5,
                         renewal_premium=.05,
```

(continues on next page)

(continued from previous page)

```
                         renewal_policy=200,
                         initial_policy=200))
```

```
SOA Question 6.2: (E) 3604
3604.229940320728
```

SOA Question 6.16

For a fully discrete 20-year endowment insurance of 100,000 on (30), you are given:

- d = 0.05
- Expenses, payable at the beginning of each year, are:

	First Year	First Year	Renewal Years	Renewal Years
	Percent of Premium	Per Policy	Percent of Premium	Per Policy
Taxes	4%	0	4%	0
Sales Commission	35%	0	2%	0
Policy Maintenance	0%	250	0%	50

- The net premium is 2143

Calculate the gross premium using the equivalence principle.

```
print("SOA Question 6.16: (A) 2408.6")
life = Premiums().set_interest(d=0.05)
A = life.insurance_equivalence(premium=2143, b=100000)
a = life.annuity_equivalence(premium=2143, b=100000)
p = life.gross_premium(A=A,
                       a=a,
                       benefit=100000,
                       settlement_policy=0,
                       initial_policy=250,
                       initial_premium=.04+.35,
                       renewal_policy=50,
                       renewal_premium=.04+.02)
print(A, a, p)
```

```
SOA Question 6.16: (A) 2408.6
0.3000139997200056 13.999720005599887 2408.575206281868
```

SOA Question 6.20

For a special fully discrete 3-year term insurance on (75), you are given:

- The death benefit during the first two years is the sum of the net premiums paid without interest
- The death benefit in the third year is 10,000

x	p_x
75	0.90
76	0.88
77	0.85

- $i = 0.04$

Calculate the annual net premium.

```python
print("SOA Question 6.20:  (B) 459")
l = lambda x,s: dict(zip([75, 76, 77, 78],
                         np.cumprod([1, .9, .88, .85]))).get(x+s, 0)
life = Premiums().set_interest(i=0.04).set_survival(l=l)
a = life.temporary_annuity(75, t=3)
IA = life.increasing_insurance(75, t=2)
A = life.deferred_insurance(75, u=2, t=1)
print(life.solve(lambda P: P*IA + A*10000 - P*a, target=0, grid=100))
```

```
SOA Question 6.20:  (B) 459
458.83181728297285
```

SOA Question 6.29

(35) purchases a fully discrete whole life insurance policy of 100,000. You are given:

- The annual gross premium, calculated using the equivalence principle, is 1770
- The expenses in policy year 1 are 50% of premium and 200 per policy
- The expenses in policy years 2 and later are 10% of premium and 50 per policy
- All expenses are incurred at the beginning of the policy year
- $i = 0.035$

Calculate $\ddot{a}_{35}$.

```python
print("SOA Question 6.29  (B) 20.5")
life = Premiums().set_interest(i=0.035)
def fun(a):
    return life.gross_premium(A=life.insurance_twin(a=a),
                              a=a,
                              benefit=100000,
                              initial_policy=200,
                              initial_premium=.5,
                              renewal_policy=50,
                              renewal_premium=.1)
print(life.solve(fun, target=1770, grid=[20, 22]))
```

```
SOA Question 6.29  (B) 20.5
20.480268314431726
```

SOA Question 6.33

An insurance company sells 15-year pure endowments of 10,000 to 500 lives, each age x, with independent future lifetimes. The single premium for each pure endowment is determined by the equivalence principle.

You are given:

- $i = 0.03$
- $\mu_x(t) = 0.02t, \quad t \geq 0$
- ${}_0L$ is the aggregate loss at issue random variable for these pure endowments.

Using the normal approximation without continuity correction, calculate $Pr({}_0L) > 50,000)$.

```
print("SOA Question 6.33:  (B) 0.13")
life = Premiums().set_survival(mu=lambda x,t: 0.02*t).set_interest(i=0.03)
var = life.E_x(x=0, t=15, moment=life.VARIANCE, endowment=10000)
p = 1 - life.portfolio_cdf(mean=0, variance=var, value=50000, N=500)
print(p)
```

```
SOA Question 6.33:  (B) 0.13
0.12828940905648634
```

SOA Question 5.6:

For a group of 100 lives age x with independent future lifetimes, you are given:

- Each life is to be paid 1 at the beginning of each year, if alive
- $A_x = 0.45$
- ${}_2A_x = 0.22$
- $i = 0.05$
- Y is the present value random variable of the aggregate payments.

Using the normal approximation to Y, calculate the initial size of the fund needed in order to be 95% certain of being able to make the payments for these life annuities.

```
print("SOA Question 5.6:  (D) 1200")
life = Premiums().set_interest(i=0.05)
var = life.annuity_variance(A2=0.22, A1=0.45)
mean = life.annuity_twin(A=0.45)
fund = life.portfolio_percentile(mean, var, prob=.95, N=100)
print(fund)
```

```
SOA Question 5.6:  (D) 1200
1200.6946732201702
```

CHAPTER

TEN

POLICY VALUES

Policy value at time t is the expected present value, at time t, of the future loss random variable, where P is the premium for b units of insurance, not necessarily the benefit premium determined from the equivalence principle.

Net future loss at issue

For net future loss at time $t = 0$, we consider benefit payments and net premiums only.

$${}_0L = b\, v^{K_x+1} - P\ddot{a}_{\overline{K_x+1|}} = (b + \frac{P}{d})\, v^{K_x+1} - \frac{P}{d}$$

- net future loss at issue of fully discrete whole life insurance

$${}_0L = b\, v^{T_x} - P\overline{a}_{\overline{T_x|}} = (b + \frac{P}{\delta})\, v^{T_x} - \frac{P}{\delta} \text{ (continuous)}$$

- net future loss at issue of fully continuous whole life insurance

Gross future loss at issue

For gross future loss at time $t = 0$, expenses are included along with benefits payments and gross premiums.

$${}_0L = (b + E + \frac{G - e_r}{d})\, v^{K_x+1} - \frac{G - e_r}{d} + (e_i - e_r)$$

- gross future loss at issue of fully discrete whole life insurance

10.1 Net policy value

The amount needed at time t to cover the shortfall of expected future benefits greater than the EPV of future net premiums after time t is called the policy value for the policy at time t, denoted ${}_tV$,

$${}_tV = E[{}_tL] = EPV_t(\text{future benefits}) - EPV_t(\text{future net premiums})$$

- net policy value at time t is the expected net future loss of benefits less premiums after time t

$${}_0V = 0$$

- net policy value at issue is 0 because of the equivalence principle

Term insurance:

$${}_nV = 0$$

- net policy value at year n is 0 for a n-year term insurance

Endowment insurance:

$${}_nV = \text{endowment benefit}$$

- net policy value at year n is equal to the endowment benefit for a n-year endowment insurance

Shortcuts for whole life and endowment insurance:

These have a simple relationship between insurance and annuity, hence the policy value formulas can be expressed in temrs of insurances only or annuities only.

$${}_tV = b[1 - \frac{\ddot{a}_{x+t}}{\ddot{a}_x}] \text{ or } b[\frac{A_{x+t} - A_x}{1 - A_x}]$$

- net policy value at time t of fully-discrete whole life insurance

$${}_tV = b[1 - \frac{\overline{a}_{x+t}}{\overline{a}_x}] \text{ or } b[\frac{\overline{A}_{x+t} - \overline{A}_x}{1 - \overline{A}_x}]$$

- net policy value at time t of fully-continuous whole life insurance

$${}_tV = b[1 - \frac{\ddot{a}_{x+t:\overline{n-t|}}}{\ddot{a}_{x:\overline{n|}}}] \text{ or } b[\frac{A_{x+t:\overline{n-t|}} - A_{x:\overline{n|}}}{1 - A_{x:\overline{n|}}}]$$

- net policy value at time t of fully-discrete endowment insurance

$${}_tV = b[1 - \frac{\overline{a}_{x+t:\overline{n-t|}}}{\overline{a}_{x:\overline{n|}}}] \text{ or } b[\frac{\overline{A}_{x+t:\overline{n-t|}} - \overline{A}_{x:\overline{n|}}}{1 - \overline{A}_{x:\overline{n|}}}]$$

- net policy value at time t of fully-continuous endowment insurance

10.2 Gross policy value

Gross premium policy values explicitly allow for expenses and for the full gross premium.

$${}_tV^g = E[{}_tL^g] = EPV_t(\text{future benefits}) + EPV_t(\text{future expenses}) - EPV_t(\text{future premiums})$$

- gross policy value at time t is the expected net future loss of benefits and expenses less premiums after time t

10.3 Variance of future loss

These formulas apply to *whole life* and *endowment insurance* **only**. In these cases, the function of the random variable v^{T_x} is the same for both the insurance and annuity of premiums which simplifies the formulas for variance of future loss.

Net future loss

$$Var[{}_tL] = (b + \frac{P}{d})^2\ [{}^2A_{x+t} - (A_{x+t})^2]$$

- fully discrete whole life insurance

$$Var[{}_tL] = (b + \frac{P}{\delta})^2\ [{}^2\overline{A}_{x+t} - (\overline{A}_{x+t})^2]$$

- fully continuous whole life insurance

$$Var[{}_tL] = (b + \frac{P}{d})^2\ [{}^2A_{x+t:\overline{n-t|}} - (A_{x+t:\overline{n-t|}})^2]$$

- fully discrete endowment insurance

$$Var[{}_tL] = (b + \frac{P}{\delta})^2\ [{}^2\overline{A}_{x+t:\overline{n-t|}} - (\overline{A}_{x+t:\overline{n-t|}})^2]$$

- fully continuous endowment insurance

Gross future loss

$$Var[{}_tL] = (b + E + \frac{G - e_r}{d})^2\ [{}^2A_{x+t} - (A_{x+t})^2]$$

- fully discrete whole life insurance

$$Var[{}_tL] = (b + E + \frac{G - e_r}{d})^2\ [{}^2A_{x+t:\overline{n-t|}} - (A_{x+t:\overline{n-t|}})^2]$$

- fully discrete endowment insurance

Shortcuts for variance of net future loss under equivalence principle

When net premiums are set under equivalence principle, these shortcuts are available without explicitly specifying the value of net premiums (which can be stated as a function of insurance only under this assumption) for *whole life* and *endowment insurance* **only**:

$$Var[{}_tL] = b^2[\frac{{}^2A_{x+t} - (A_{x+t})^2}{(1 - A_x)^2}]$$

- variance of net future loss for fully-discrete whole life insurance under EPP

$$Var[{}_tL] = b^2[\frac{{}^2\overline{A}_{x+t} - (\overline{A}_{x+t})^2}{(1 - \overline{A}_x)^2}]$$

- variance of net future loss for fully-continuous whole life insurance under EPP

$$Var[{}_tL] = b^2[\frac{{}^2A_{x+t:\overline{n-t|}} - (A_{x+t:\overline{n-t|}})^2}{(1 - A_{x:\overline{n|}})^2}]$$

- variance of net future loss for fully-discrete endowment insurance under EPP

$$Var[{}_tL] = b^2[\frac{{}^2\overline{A}_{x+t:\overline{n-t|}} - (\overline{A}_{x+t:\overline{n-t|}})^2}{(1 - \overline{A}_{x:\overline{n|}})^2}]$$

- variance of net future loss for fully-continuous endowment insurance under EPP

10.4 Expense reserve

$$P^e = P^g - P^n$$

- expense premium (sometimes, expense loading) is defined as the difference of gross premium and net premium: if expenses are weighted to the start of the contract, as is normally the case, then P^e will be greater than the renewal expense as it must fund both the renewal and initial expenses.

$${}_tV^e = {}_tV^g - {}_tV = EPV_t(\text{future expenses}) - EPV_t(\text{future expense loadings})$$

- expense reserves, defined as the difference between gross reserves and net reserves, also equals the expected present value of future expenses less the expected present value of future expense loadings (or expense premiums)

Generally:

- ${}_tV^e < 0$
- ${}_tV > {}_tV^g > 0 > {}_tV^e$

10.5 Probabilities and Percentiles

For whole life, term and endowment insurance, the present value of benefits decreases with time, while the present value of the premium annuity increases

Percentile of ${}_0L$:

Since loss at issue is a monotonically decreasing function of lifetime, the $100p$-th percentile of the loss at issue r.v. is determined from the $100p$-th percentile of the survival function.

Probability of ${}_0L$:

To calculate the probability that the present value of loss at issue is greater than some value $Pr({}_0L \geq L_\pi)$, calculate the time t_π for which ${}_0L = b\ v^{t_\pi} - P\overline{a}_{\overline{t_\pi|}}$ has the required value, then calculate the probability that time of death T_x is in that range.

10.6 Methods

The `PolicyValues` class implements methods for computing net and gross future losses, and policy values (expected present values). The `Contract` class is used store and retrieve policy contract terms, such as benefit amounts and the various types of expenses.

```
from actuarialmath.policyvalues import PolicyValues, Contract
import math
import describe
describe.methods(PolicyValues)
```

```
class PolicyValues - Compute net and gross future losses and policy values

    Methods:
    --------

    gross_policy_value(x, s, t, n, contract):
      Gross policy values for insurance: t_V = E[L_t]

    gross_policy_variance(x, s, t, n, contract):
      Variance of gross policy value for WL and Endowment Insurance only

    net_policy_value(x, s, t, b, n, endowment, discrete):
      Net policy value assuming premiums from equivalence: E[L_t]

    net_policy_variance(x, s, t, b, n, endowment, discrete):
      Variance of net future loss for WL or Endowment Insurance only

    net_future_loss(A, A1, b):
      Shortcut for net policy value with WL or Endowment Insurance factors

    gross_future_loss(A, a, contract):
      Shortcut for gross policy value with WL or Endowment Insurance factors

    net_variance_loss(A1, A2, A, b):
      Variance of net loss with WL or Endowment Insurance factors

    gross_variance_loss(A1, A2, contract):
      Variance of gross loss with WL or endowment insurance factors
```

(continues on next page)

(continued from previous page)

```
    L_from_t(t, contract):
      PV of Loss L(t) at time of death t = T_x

    L_from_prob(x, prob, contract):
      Percentile of PV future loss r.v. L given probability

    L_to_t(L, contract):
      Time of death T_x s.t. PV future loss is no more than L

    L_to_prob(x, L, contract):
      Probability such that PV of future loss r.v. is no more than L"

    L_plot(x, s, stop, T, contract, ax, dual, title, color, alpha):
      Plot PV of future loss r.v. L vs time of death T_x
```

```
describe.methods(Contract)
```

```
class Contract - Set and retrieve policy contract terms

    Args:
      premium : level premium amount
      benefit : insurance death benefit amount
      settlement_policy : settlement expense per policy
      endowment : endowment benefit amount
      initial_policy : first year total expense per policy
      initial_premium : first year total premium per $ of gross premium
      renewal_policy : renewal expense per policy
      renewal_premium : renewal premium per $ of gross premium
      discrete : annuity due (True) or continuous (False)
      T : term of insurance
      discrete : annuity due (True) or continuous (False)

    Methods:
    --------

    set_contract(terms):
      Update any existing policy contract terms

    premium_terms():
      Dict of terms required for calculating gross premiums

    renewal_profit():
      Renewal dollar profit (premium less renewal expenses)

    initial_cost():
      Total initial cost (excludes renewal expenses)

    claims_cost():
      Total claims cost (death benefit + settlement expense)

    renewals(t):
      Returns contract object with initial terms set to renewal terms
```

10.7 Examples

The `Contract` helper class can be initialized to store expenses, benefit amounts and values of other contract features. Its `set_contract` method is called to update individual values, its `renewals` method returns a new Contract object with its initial expenses changed to be the same as renewal expenses, and other "getter" methods derive various summary values.

```
contract = Contract(benefit=1000,
                    settlement_policy=20,
                    initial_policy=10,
                    initial_premium=0.75,
                    renewal_policy=2,
                    renewal_premium=0.1)
print(contract.premium_terms)
print(contract.renewals().premium_terms)
contract.set_contract(premium=35.38618830746352)
print(contract.initial_cost)
print(contract.claims_cost)
print(contract.renewal_profit)
```

```
{'benefit': 1000, 'initial_policy': 10, 'initial_premium': 0.75, 'renewal_policy':␣
↪2, 'renewal_premium': 0.1, 'settlement_policy': 20}
{'benefit': 1000, 'initial_policy': 2, 'initial_premium': 0.1, 'renewal_policy': 2,
↪ 'renewal_premium': 0.1, 'settlement_policy': 20}
31.00102239985129
1020
29.84756947671717
```

The `gross_policy_value` method in the `PolicyValues` class compute policy values at issue or future year t for general insurances and premiums. For whole life and endowment insurances only, `gross_policy_variance` computes the variance of the future gross loss PV random variable. Internally, these use the `gross_future_loss` and `gross_variance_loss` helper methods, which are also available to be called directly if the actuarial insurance factors are already provided as inputs.

When net premiums are given by the equivalence principle, `net_policy_value` computes the net policy values at issue or a future year t. For whole life and endowment insurances only, `net_policy_variance` compute the variance of the future net loss PV random variable. Internally, these use the `net_future_loss` and `net_variance_loss` helper methods, which are also available to be called directly if the actuarial insurance factors are already provided as inputs.

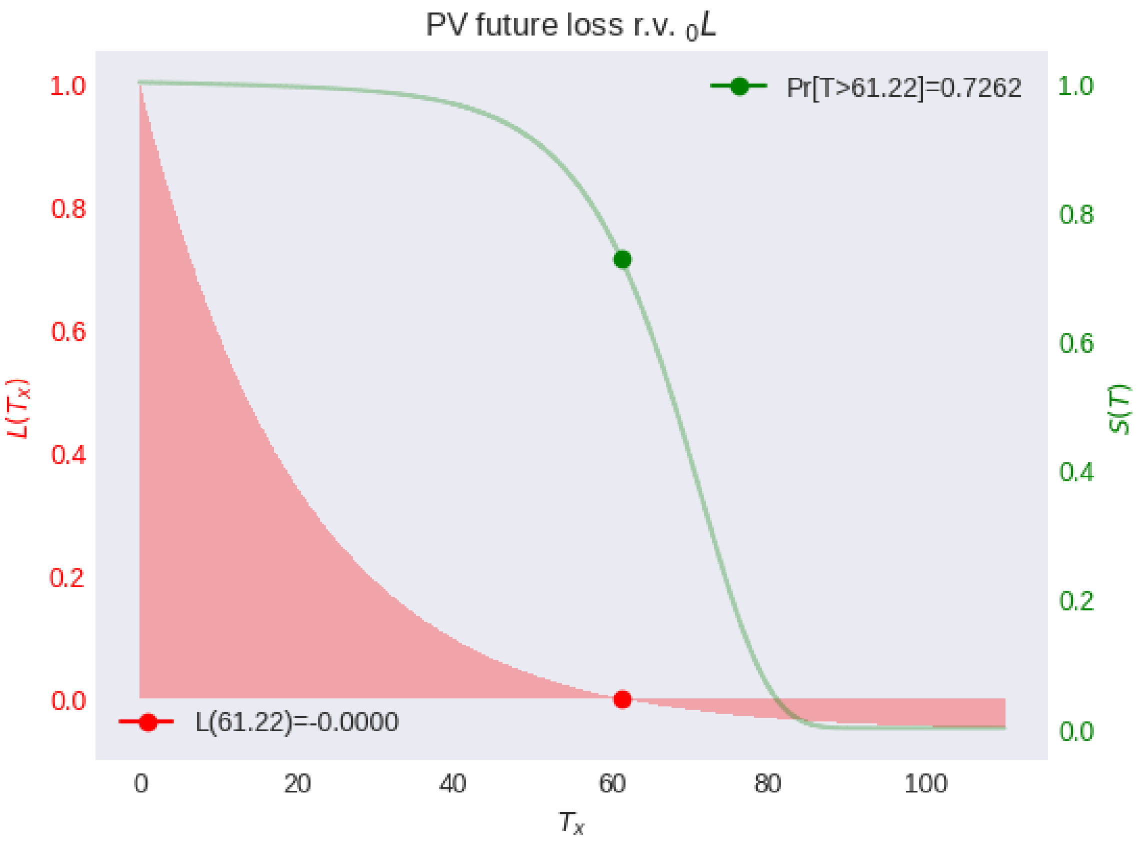
```
def S(x, s, t):  # define a survival function
    return math.exp(-.00022*t - (.0000027*1.124**(x+s)*(1.124**t-1))/math.log(1.124))
life = PolicyValues().set_survival(S=S, minage=20, maxage=130)\
                     .set_interest(i=0.05)
print(life.net_policy_value(x=20, b=contract.benefit, discrete=contract.discrete),
      life.net_policy_value(x=20, t=5, b=contract.benefit, discrete=contract.
↪discrete))
print(life.net_policy_variance(x=20, b=contract.benefit, discrete=contract.discrete),
      life.net_policy_variance(x=20, t=5, b=contract.benefit, discrete=contract.
↪discrete))

print(life.gross_policy_value(x=20, contract=contract),
      life.gross_policy_variance(x=20, contract=contract))
```

```
0.0 12.88972064810054
3734.4039865925088 4534.593306902999
-514.7435740643272 9155.101027638082
```

The `L_from_t` method computes the present value of the future loss when death occurs at time t, while `L_from_prob` returns the $100p$-th percentile value of L. The inverse computations are performed by `L_to_t` and `L_to_prob` methods respectively. `L_plot` plots the present value of future loss r.v. L as a function of time t. The example below determines the breakeven lifetime, when the present value of future loss L(t) = 0, and annotates in a plot with survival probabilities.

```
def S(x, s, t):  # define a survival function
    return math.exp(-.00022*t - (.0000027*1.124**(x+s)*(1.124**t-1))/math.log(1.124))
life = PolicyValues().set_survival(S=S, minage=20, maxage=130)\
                     .set_interest(i=0.05)
P = life.net_premium(x=20, discrete=False)
contract = Contract(premium=P, discrete=False)
T = life.L_to_t(L=0, contract=contract)         # breakeven T                        ␣
 ↪
life.L_plot(x=20, T=T, contract=contract, dual=True)
prob = life.L_to_prob(x=20, L=0, contract=contract)
print('L:', life.L_from_t(t=T, contract=contract),
      life.L_from_prob(x=20, prob=prob, contract=contract))
```

```
L: -4.905848505210031e-10 -4.905742478911179e-10
```



SOA Question 6.24

For a fully continuous whole life insurance of 1 on (x), you are given:

- L is the present value of the loss at issue random variable if the premium rate is determined by the equivalence principle
- L^* is the present value of the loss at issue random variable if the premium rate is 0.06
- $\delta = 0.07$
- $\overline{A}_x = 0.30$
- $Var(L) = 0.18$

Calculate $Var(L^*)$.

```
print("SOA Question 6.24:  (E) 0.30")
life = PolicyValues().set_interest(delta=0.07)
x, A1 = 0, 0.30   # Policy for first insurance
P = life.premium_equivalence(A=A1, discrete=False)  # Need its premium
contract = Contract(premium=P, discrete=False)
def fun(A2):  # Solve for A2, given Var(Loss)
    return life.gross_variance_loss(A1=A1, A2=A2, contract=contract)
A2 = life.solve(fun, target=0.18, grid=0.18)
contract = Contract(premium=0.06, discrete=False)     # Solve second insurance
variance = life.gross_variance_loss(A1=A1, A2=A2, contract=contract)
print(variance)
```

```
SOA Question 6.24:  (E) 0.30
0.30419999999999975
```

SOA Question 6.30

For a fully discrete whole life insurance of 100 on (x), you are given:

- The first year expense is 10% of the gross annual premium
- Expenses in subsequent years are 5% of the gross annual premium
- The gross premium calculated using the equivalence principle is 2.338
- $i = 0.04$
- $\ddot{a}_x = 16.50$
- ${}^2A_x = 0.17$

Calculate the variance of the loss at issue random variable.

```
print("SOA Question 6.30:  (A) 900")
life = PolicyValues().set_interest(i=0.04)
contract = Contract(premium=2.338, benefit=100, initial_premium=.1,
                    renewal_premium=0.05)
var = life.gross_variance_loss(A1=life.insurance_twin(16.50),
                                A2=0.17, contract=contract)
print(var)
```

```
SOA Question 6.30:  (A) 900
908.141412994607
```

SOA Question 7.32

For two fully continuous whole life insurance policies on (x), you are given:

	Death Benefit	Annual Premium Rate	Variance of the PV of Future Loss at t
Policy A	1	0.10	0.455
Policy B	2	0.16	-

- $\delta = 0.06$

Calculate the variance of the present value of future loss at t for Policy B.

```
print("SOA Question 7.32:  (B) 1.4")
life = PolicyValues().set_interest(i=0.06)
contract = Contract(benefit=1, premium=0.1)
def fun(A2):
    return life.gross_variance_loss(A1=0, A2=A2, contract=contract)
A2 = life.solve(fun, target=0.455, grid=0.455)
contract = Contract(benefit=2, premium=0.16)
var = life.gross_variance_loss(A1=0, A2=A2, contract=contract)
print(var)
```

```
SOA Question 7.32:  (B) 1.4
1.3848168384380901
```

SOA Question 6.12

For a fully discrete whole life insurance of 1000 on (x), you are given:

- The following expenses are incurred at the beginning of each year:

	Year 1	Years 2+
Percent of premium	75%	10%
Maintenance expenses	10	2

- An additional expense of 20 is paid when the death benefit is paid
- The gross premium is determined using the equivalence principle
- $i = 0.06$
- $\ddot{a}_x = 12.0$
- ${}^2A_x = 0.14$

Calculate the variance of the loss at issue random variable.

```
print("SOA Question 6.12:  (E) 88900")
life = PolicyValues().set_interest(i=0.06)
a = 12
A = life.insurance_twin(a)
contract = Contract(benefit=1000, settlement_policy=20,
                    initial_policy=10, initial_premium=0.75,
                    renewal_policy=2, renewal_premium=0.1)
contract.premium = life.gross_premium(A=A, a=a, **contract.premium_terms)
print(A, contract.premium)
L = life.gross_variance_loss(A1=A, A2=0.14, contract=contract)
print(L)
```

```
SOA Question 6.12:  (E) 88900
0.3207547169811321 35.38618830746352
88862.59592874818
```

CHAPTER

ELEVEN

RESERVES

The term reserves is sometimes used in place of policy values. AMLCR uses policy value to mean the expected value of the future loss random variable, and restricts reserve to mean the actual capital held in respect of a policy, which may be greater than or less than the policy value.

11.1 Recursion

The following recursive formulae relating ${}_tV$ to ${}_{t+1}V$ for policy values can be derived for policies with discrete cash flows.

Gross reserves

$({}_tV^g + G - e)(1+i) = q_{x+t}\,(b+E) + p_{x+t}\;{}_{t+1}V^g$

- recursion for gross reserves

Expense reserves

$({}_tV^e + P^e - e)(1+i) = q_{x+t}\,E + p_{x+t}\;{}_{t+1}V^e$

- recursion for expense reserves

Net reserves

$({}_tV + P)(1+i) = q_{x+t}\,b + p_{x+t}\;{}_{t+1}V$

- recursion for net reserves

Refund of benefit reserve

$({}_tV + P)(1+i) = q_{x+t}\,(b + {}_{t+1}V) + p_{x+t}\;{}_{t+1}V$

- if reserves are also refunded after death, then the end-of-year value reserves should be added to death benefits in the reserves recursion formula

11.2 Interim reserves

Recursive formulae for interim reserves ${}_{t+r}V$ where $0 \le r \le 1$ can be similarly obtained. These compute the reserve for fully discrete annual-premium insurances in the middle of the year.

$({}_tV + P)(1+i)^r = {}_rq_{x+t}\,b\,v^{1-r} + {}_rp_{x+t}\;{}_{t+r}V$

- forward recursion for interim net reserves

${}_{t+r}V\,(1+i)^{1-r} = {}_{1-r}q_{x+t+r}\,b + {}_{1-r}p_{x+t+r}\;{}_{t+1}V$

- backward recursion for interim net reserves

11.3 Modified reserves

Because acquisition expenses are large relative to the renewal and claims expenses, accounting with level net premiums typically results in large negative values for expense reserves (called deferred acquisition costs or DAC) particularly at issue. Modified premium reserves are computed without expenses, and modifies the net premium method to assume a lower initial net premium that allow implicitly for the DAC.

Full Preliminary Term

FPT is the most common method for modifying net premium policy value. It treats the insurance policy as one-year term insurance combined with a policy as if it were issued one year later.

$$\alpha = A^{1}_{x:\overline{1|}} = v\, q_x$$

- initial FPT premium

$$\beta = \frac{A_{x+1}}{\ddot{a}_{x+1}}$$

- renewal FPT premium

$${}_0V^{FPT} ~=~ {}_1V^{FPT} ~= 0$$

- since renewal premium set year 1 policy value to 0, while initial premium set to equal year 1 expected benefits.

$${}_tV^{FPT} \text{ for } (x) ~=~ {}_{t-1}V \text{ for } (x+1)$$

- since renewal FPT premium for (x) is net premium for (x+1) with term lengths adjusted

11.4 Methods

The `Reserves` class implements methods to solve reserves by recursion, and compute interim and modified reserves.

```python
from actuarialmath import Reserves, Contract
import describe
describe.methods(Reserves)
```

```
class Reserves - Compute recursive, interim or modified reserves

    Methods:
    --------

    set_reserves(T, endowment, V):
      Set values of the reserves table and the endowment benefit amount

    fill_reserves(x, s, reserve_benefit, contract):
      Iteratively fill in missing values in reserves table

    t_V_forward(x, s, t, premium, benefit, per_premium, per_policy, reserve_
↪benefit):
      Forward recursion (with optional reserve benefit)

    t_V_backward(x, s, t, premium, benefit, per_premium, per_policy, reserve_
↪benefit):
      Backward recursion (with optional reserve benefit)

    t_V(x, s, t, premium, benefit, reserve_benefit, per_premium, per_policy):
```

(continues on next page)

(continued from previous page)

```
      Solve year-t reserves by forward or backward recursion

    r_V_forward(x, s, r, premium, benefit):
      Forward recursion for interim reserves

    r_V_backward(x, s, r, benefit):
      Backward recursion for interim reserves

    FPT_premium(x, s, n, b, first):
      Initial or renewal Full Preliminary Term premiums

    FPT_policy_value(x, s, t, b, n, endowment, discrete):
      Compute Full Preliminary Term policy value at time t

    V_plot(ax, color, title):
      Plot values from reserves tables

    reserves_frame():
      Returns reserves table as a DataFrame
```

11.5 Examples

The `FPT_premium` and `FPT_policy_value` methods compute the Full Preliminary Term premiums and policy values for whole life, temporary and endowment insurances.

```
import math
def S(x, s, t):  # define a survival function
    return math.exp(-.00022*t - (.0000027*1.124**(x+s)*(1.124**t-1))/math.log(1.124))
life = Reserves().set_survival(S=S, minage=20, maxage=130)\
                  .set_interest(i=0.05)
P = life.net_premium(x=20, b=1000)
print('Level Net Premium:', P)
print('FPT premiums:     ',
      life.FPT_premium(x=20, b=1000, first=True),
      life.FPT_premium(x=20, b=1000, first=False))
print('FPT policy values:',
      [round(life.FPT_policy_value(x=20, b=1000, t=t), 4) for t in range(5)])
```

```
Level Net Premium: 2.465109289578718
FPT premiums:      0.2377514556176763 2.582546365777722
FPT policy values: [0, 0, 2.459, 5.0374, 7.7409]
```

The `set_reserves` method is called to load given values of reserves into an internal table, and then `fill_reserves` method can be called to iteratively fill in missing values. Internally, this calls the `t_V` method which attempts both the `t_V_forward` and `t_V_backward` methods that implement the recursion formula forwards and backwards respectively. Interim reserves recursion formulas for fractional durations are similarly implemented in the `r_V_forward` and `r_V_backward` methods. Values of reserves over time can be plotted with `V_plot` or displayed with the `reserves_frame` methods.

SOA Question 7.31

For a fully discrete 3-year endowment insurance of 1000 on (x), you are given:

- Expenses, payable at the beginning of the year, are:

Year(s)	Percent of Premium	Per Policy
1	20%	15
2 and 3	8%	5

- The expense reserve at the end of year 2 is –23.64
- The gross annual premium calculated using the equivalence principle is G = 368.
- $G = 1000P_{x:\overline{3|}} + P^e$, where P^e is the expense loading

Calculate $P_{x:\overline{3|}}$.

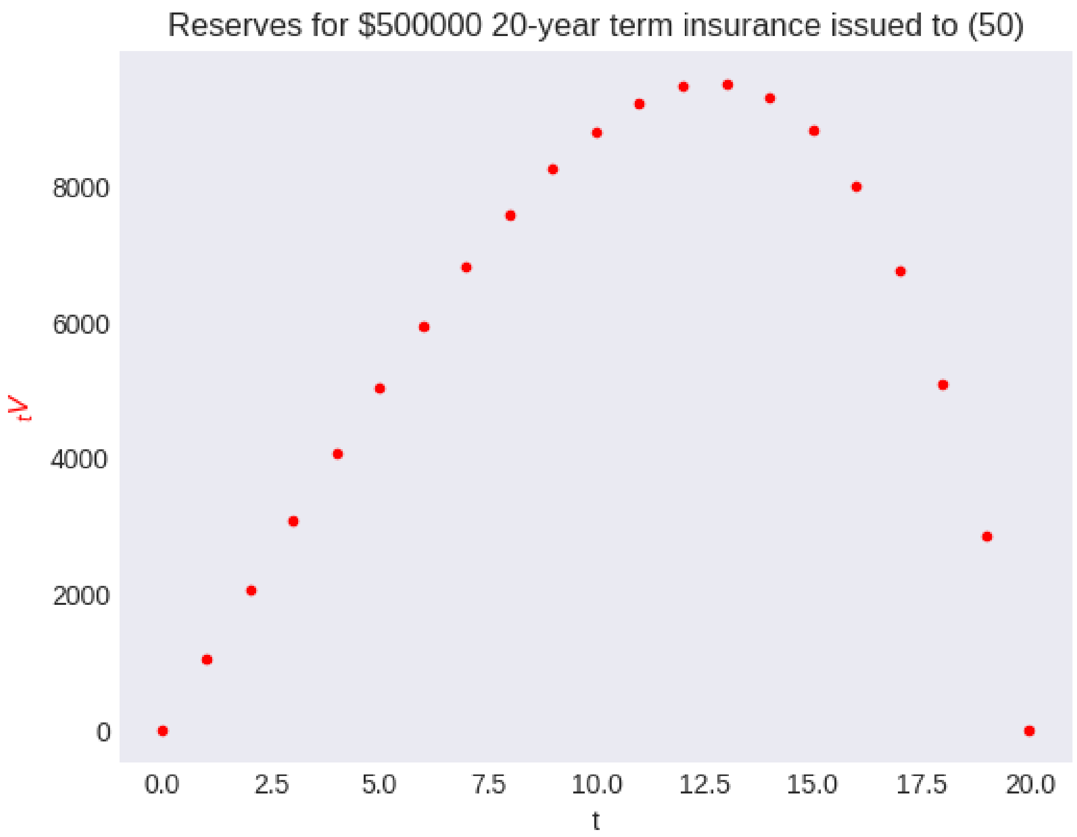
```
print("SOA Question 7.31:  (E) 0.310")
x = 0
life = Reserves().set_reserves(T=3)
G = 368.05
def fun(P):  # solve net premium from expense reserve equation
    return life.t_V(x=x, t=2, premium=G-P, benefit=lambda t: 0,
                    per_policy=5 + .08*G)
P = life.solve(fun, target=-23.64, grid=[.29, .31]) / 1000
print(P)
```

```
SOA Question 7.31:  (E) 0.310
0.309966
```

AMLCR2 Figure 7.4:

Policy values for each year of a 20-year term insurance, sum insured $500,000, issuedto(50).MortalityfollowstheStandardUltimateLi$

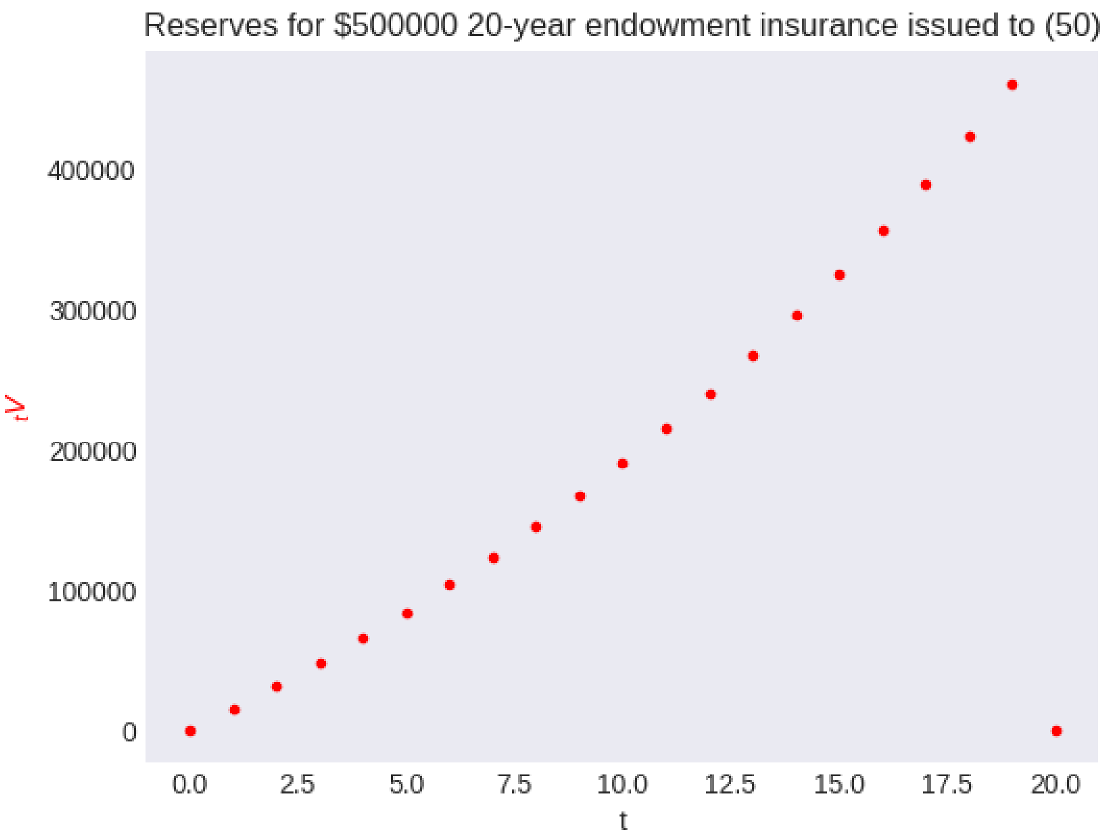
```
from actuarialmath.sult import SULT
life = SULT()
x, T, b = 50, 20, 500000    # $500K 20-year term insurance for (50)
P = life.net_premium(x=x, t=T, b=b)
life.set_reserves(T=T)\
    .fill_reserves(x=x, contract=Contract(premium=P, benefit=b))
life.V_plot(title=f"Reserves for ${b} {T}-year term insurance issued to ({x})")
```



AMLCR2 Figure 7.3:

Policy values for each year of a 20-year endowment insurance, sum insured $500,000, issuedto(50).MortalityfollowstheStandardUltimateLifeTable(noteAMLCR2usedStandardSelectTable), withint$

```
from actuarialmath.sult import SULT
life = SULT()
x, T, b = 50, 20, 500000     # $500K 20-year term insurance for (50)
P = life.net_premium(x=x, t=T, b=b, endowment=b)
life.set_reserves(T=T)\
    .fill_reserves(x=x, contract=Contract(premium=P, benefit=b, endowment=b))
life.V_plot(title=f"Reserves for ${b} {T}-year endowment insurance issued to ({x})")
```


Reserves for $500000 20-year endowment insurance issued to (50)
400000
300000
200000
100000
0
$_tV$
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
t

CHAPTER

TWELVE

RECURSION

Using annual values provided we can calculate other values at other ages and durations by applying recursion formulas and other actuarial identities.

12.1 Chain rule

${}_{t+n}p_x = {}_np_x \cdot {}_tp_{x+n}$

- survival probability chain rule

${}_{t+n}E_x = {}_nE_x \cdot {}_tE_{x+n}$

- pure endowment chain rule

12.2 Expected future lifetime

For both complete and curtate future lifetime, we can decompose the formula for the life expectancy of (x) into a temporary life expectancy, plus the probability of surviving till the end of that term times the remaining life expectancy.

$e_x = e_{x:\overline{m|}} + {}_mp_x \, e_{x+m}$

- recursion formula for curtate expectation of lifetime

$\mathring{e}_x = \mathring{e}_{x:\overline{m|}} + {}_mp_x \, \mathring{e}_{x+m}$

- recursion formula for complete expectation of lifetime

$e_{x:\overline{m+n|}} = e_{x:\overline{m|}} + {}_mp_x \, e_{x:\overline{n|}}$

- recursion formula for limited curtate expectation of lifetime

$\mathring{e}_{x:\overline{m+n|}} = \mathring{e}_{x:\overline{m|}} + {}_mp_x \, \mathring{e}_{x+m:\overline{n|}}$

- recursion formula for limited complete expectation of lifetime

$e_x = p_x(1 + e_{x+1})$

- special case of a one-year recursion formula for curtate expectaion of lifetime

$\mathring{e}_x = \mathring{e}_{x:\overline{1|}} + p_x \, \mathring{e}_{x+1}$

- special case of a one-year recursion formula for complete expectation of lifetime

$e_{x:\overline{1|}} = p_x$

- shortcut for one-year limited curtate expectaion of lifetime

12.3 Life insurance

$$A_x = v\,q_x + v\,p_x\,A_{x+1} \;\Rightarrow\; A_{x+1} = \frac{A_x - v\,q_x}{v\,p_x}$$

- whole life insurance recursion

$$A^1_{x:\overline{t|}} = v\,q_x + v\,p_x\,A^1_{x+1:\overline{t-1|}}$$

- term life insurance recursion

$$A^1_{x:\overline{1|}} = v\,q_x$$

- shortcut for one-year term life insurance

$${}^2A^1_{x:\overline{1|}} = v^2\,q_x$$

- shortcut for second moment of one-year term life insurance

$$A_{x:\overline{0|}} = {}_0E_x = 1$$

- endowment insurance at end of term is pure endowment

$$A_{x:\overline{1|}} = q_x\,v + p_x\,v = v$$

- shortcut for one-year endowment insurance

$${}^2A_{x:\overline{1|}} = v^2$$

- shortcut for second moment of one-year endowment insurance

$$IA^1_{x:\overline{t|}} = v\,q_x + v\,p_x\,(A_{x+1} + IA^1_{x+1:\overline{t-1|}})$$

- increasing insurance recursion

$$DA^1_{x:\overline{t|}} = t\,v\,q_x + v\,p_x\,(DA^1_{x+1:\overline{t-1|}})$$

- decreasing insurance recursion

12.4 Life annuities

$$\ddot{a}_x = 1 + v\,p_x\,\ddot{a}_{x+1} \;\Rightarrow\; \ddot{a}_{x+1} = \frac{\ddot{a}_x - 1}{v\,p_x}$$

- whole life annuity recursion

$$\ddot{a}_{x:\overline{t|}} = 1 + v\,p_x\,\ddot{a}_{x+1:\overline{t-1|}}$$

- temporary annuity recursion

$$\ddot{a}_{x:\overline{1|}} = 1$$

- shortcut for one-year temporary annuity

12.5 Methods

The `Recursion` class implements methods to apply recursive, shortcut and actuarial formulas, and traces the steps taken to find the solution.

Caveats:

1. Not all possible recursion rules and actuarial equations have (yet) been implemented in the present version of the package.
2. You may set the recursion depth to a larger limit than the default of 3 (with the keyword argument `depth` when initializing a Recursion class object).
3. But generally, the current implementation may be fragile if the solution is not available within a relatively shallow search.

Notes:

- If a colab or jupyter notebook is auto-detected, the steps are displayed in latex format; else as raw text.
- These display options can be changed by calling the `blog_options` static method

```
from actuarialmath import Recursion, ConstantForce, Contract
import describe
describe.methods(Recursion)
```

```
class Recursion - Solve by appling recursive, shortcut and actuarial formulas␣
↪repeatedly

    Args:
      depth : maximum depth of recursions (default is 3)
      verbose : whether to echo recursion steps (True, default)

    Notes:
      7 types of function values can be loaded for recursion computations:

      - 'q' : (deferred) probability (x) dies in t years
      - 'p' : probability (x) survives t years
      - 'e' : (temporary) expected future lifetime, or moments
      - 'A' : deferred, term, endowment or whole life insurance, or moments
      - 'IA' : decreasing life insurance of t years
      - 'DA' : increasing life insurance of t years
      - 'a' : deferred, temporary or whole life annuity of t years, or moments

    Methods:
    --------

    set_q(val, x, s, t, u):
      Set mortality rate u|t_q_[x+s] to given value

    set_p(val, x, s, t):
      Set survival probability t_p_[x+s] to given value

    set_e(val, x, s, t, curtate, moment):
      Set expected future lifetime e_[x+s]:t to given value

    set_E(val, x, s, t, endowment, moment):
      Set pure endowment t_E_[x+s] to given value
```

(continues on next page)

(continued from previous page)

```
    set_A(val, x, s, t, u, b, moment, endowment, discrete):
      Set insurance u|_A_[x+s]:t to given value

    set_IA(val, x, s, t, b, discrete):
      Set increasing insurance IA_[x+s]:t to given value

    set_DA(val, x, s, t, b, discrete):
      Set decreasing insurance DA_[x+s]:t to given value

    set_a(val, x, s, t, u, b, variance, discrete):
      Set annuity u|_a_[x+s]:t to given value

    blog_options(latex, notebook):
      Static method to change display options for tracing the recursion steps
```

12.6 Examples

The given values of functions are set by calling the respective “setter” methods `set_q`, `set_p`, `set_e`, `set_E`, `set_A`, `set_IA`, `set_DA`, or `set_a`. Then derived function values, at other ages or durations, can be computed – the successful recursion steps taken are displayed if the `verbose` flag is initiated to True (which is the default value).

AMLCR2 Exercise 2.6

Given $P_x = 0.99$, $P_{x+1} = 0.985$, ${}_3P_{x+1} = 0.95$, $q_{x+3} = 0.02$,

Calculate (a) P_{x+3}, (b) ${}_2P_x$, (c) ${}_2P_{x+1}$, (d) ${}_3P_x$, (e) ${}_{1|2}q_x$.

```python
from actuarialmath.recursion import Recursion
x = 0
life = Recursion(depth=3).set_interest(i=0.06)\
                         .set_p(0.99, x=x)\
                         .set_p(0.985, x=x+1)\
                         .set_p(0.95, x=x+1, t=3)\
                         .set_q(0.02, x=x+3)

print(life.p_x(x=x+3), 0.98)
print(life.p_x(x=x, t=2),  0.97515)
print(life.p_x(x=x+1, t=2), 0.96939)
print(life.p_x(x=x, t=3), 0.95969)
print(life.q_x(x=x, t=2, u=1),  0.03031)
```

Survival p_{x+3} :=
$p_{x+3} = 1 - q_{x+3}$ complement of mortality

```
0.98 0.98
```

$$\begin{aligned} &\texttt{Survival}~{}_{2}p_{x}~:= \\ {}_{2}p_{x} &= {}_{3}p_{x}/~p_{x+2} && \texttt{survival chain rule} \\ p_{x+2} &= {}_{2}p_{x+1}/~p_{x+1} && \texttt{survival chain rule} \\ {}_{2}p_{x+1} &= {}_{3}p_{x+1}/~p_{x+3} && \texttt{survival chain rule} \\ {}_{3}p_{x} &= {}_{4}p_{x}/~p_{x+3} && \texttt{survival chain rule} \\ {}_{4}p_{x} &= {}_{3}p_{x+1} * ~p_{x} && \texttt{survival chain rule} \\ p_{x+3} &= 1 - ~q_{x+3} && \texttt{complement of mortality} \end{aligned}$$

```
0.9751500000000001 0.97515
```

$$\begin{aligned} &\texttt{Survival}~{}_{2}p_{x+1}~:= \\ {}_{2}p_{x+1} &= {}_{3}p_{x}/~p_{x} && \texttt{survival chain rule} \\ {}_{3}p_{x} &= {}_{4}p_{x}/~p_{x+3} && \texttt{survival chain rule} \\ {}_{4}p_{x} &= {}_{3}p_{x+1} * ~p_{x} && \texttt{survival chain rule} \\ p_{x+3} &= 1 - ~q_{x+3} && \texttt{complement of mortality} \end{aligned}$$

```
0.9693877551020409 0.96939
```

$$\begin{aligned} &\texttt{Survival}~{}_{3}p_{x}~:= \\ {}_{3}p_{x} &= {}_{4}p_{x}/~p_{x+3} && \texttt{survival chain rule} \\ {}_{4}p_{x} &= {}_{3}p_{x+1} * ~p_{x} && \texttt{survival chain rule} \\ p_{x+3} &= 1 - ~q_{x+3} && \texttt{complement of mortality} \end{aligned}$$

```
0.9596938775510204 0.95969
```

$$\begin{aligned} &\texttt{Mortality}~{}_{1|2}q_{x}~:= \\ {}_{1|2}q_{x} &= ~p_{x} - ~{}_{3}p_{x} && \texttt{complement survival} \\ {}_{3}p_{x} &= {}_{4}p_{x}/~p_{x+3} && \texttt{survival chain rule} \\ {}_{4}p_{x} &= {}_{3}p_{x+1} * ~p_{x} && \texttt{survival chain rule} \\ p_{x+3} &= 1 - ~q_{x+3} && \texttt{complement of mortality} \end{aligned}$$

```
0.030306122448979567 0.03031
```

SOA Question 6.40

For a special fully discrete whole life insurance, you are given:

- The death benefit is $1000(1.03)^k$ for death in policy year k, for $k = 1, 2, 3...$
- $q_x = 0.05$
- $i = 0.06$
- $\ddot{a}_{x+1} = 7.00$
- The annual net premium for this insurance at issue age x is 110

Calculate the annual net premium for this insurance at issue age $x + 1$.

```
print("SOA Question 6.40: (C) 116 ")
x = 0
life = Recursion().set_interest(i=0.06).set_a(7, x=x+1).set_q(0.05, x=x)
a = life.whole_life_annuity(x)
A = 110 * a / 1000
print(a, A)
life = Recursion().set_interest(i=0.06).set_A(A, x=x).set_q(0.05, x=x)
```

(continues on next page)

(continued from previous page)

```
A1 = life.whole_life_insurance(x+1)
P = life.gross_premium(A=A1 / 1.03, a=7) * 1000
print(P)
```

```
SOA Question 6.40: (C) 116
```

Whole Life Annuity $\ddot{a}_x$:=
$\ddot{a}_x = 1 + E_x * \ddot{a}_{x+1}$ backward recursion
$E_x = p_x * v$ pure endowment
$p_x = 1 - q_x$ complement of mortality

```
7.2735849056603765 0.8000943396226414
```

Whole Life Insurance A_{x+1} :=
$A_{x+1} = [\,A_x/v - q_x * b]/\,p_x$ forward recursion
$p_x = 1 - q_x$ complement of mortality

```
116.51945397474269
```

SOA Question 6.10 : (D) 0.91

For a fully discrete 3-year term insurance of 1000 on (*x*), you are given:

1. $p_x = 0.975$
2. $i = 0.06$
3. The actuarial present value of the death benefit is 152.85
4. The annual net premium is 56.05

Calculate p_{x+2}.

```
print("SOA Question 6.10: (D) 0.91")
x = 0
life = Recursion(depth=5).set_interest(i=0.06)\
                         .set_p(0.975, x=x)\
                         .set_a(152.85/56.05, x=x, t=3)\
                         .set_A(152.85, x=x, t=3, b=1000)
p = life.p_x(x=x+2)
print(p)
```

```
SOA Question 6.10: (D) 0.91
```

Survival p_{x+2} :=
$p_{x+2} = E_{x+2}/v$ one-year pure endowment
$E_{x+2} = A_{x+2:\overline{1|}} - A^1_{x+2:\overline{1|}}$ endowment insurance minus term
$A^1_{x+2:\overline{1|}} = [\,A^1_{x+1:\overline{2|}}/v - q_{x+1} * b]/\,p_{x+1}$ forward recursion
$p_{x+1} = [\,\ddot{a}_{x+1:\overline{2|}} - 1]/[v * \ddot{a}_{x+2:\overline{1|}}]$ annuity recursion
$\ddot{a}_{x+1:\overline{2|}} = [\,\ddot{a}_{x:\overline{3|}} - 1]/\,E_x$ forward recursion
$A^1_{x+1:\overline{2|}} = [\,A^1_{x:\overline{3|}}/v - q_x * b]/\,p_x$ forward recursion
$E_x = p_x * v$ pure endowment

```
0.9097382950525702
```

SOA Question 6.48

For a special fully discrete 5-year deferred 3-year term insurance of 100,000 on (x) you are given:

- There are two premium payments, each equal to P . The first is paid at the beginning of the first year and the second is paid at the end of the 5-year deferral period
- $p_x = 0.95$
- $q_{x+5} = 0.02$
- $q_{x+6} = 0.03$
- $q_{x+7} = 0.04$
- $i = 0.06$

Calculate P using the equivalence principle.

```
print("SOA Question 6.48:  (A) 3195")
life = Recursion(depth=3).set_interest(i=0.06)
x = 0
life.set_p(0.95, x=x, t=5)
life.set_q(0.02, x=x+5)
life.set_q(0.03, x=x+6)
life.set_q(0.04, x=x+7)
a = 1 + life.E_x(x, t=5)
A = life.deferred_insurance(x, u=5, t=3)
P = life.gross_premium(A=A, a=a, benefit=100000)
print(P)
```

```
SOA Question 6.48:  (A) 3195
```

$$\begin{array}{rl}
\texttt{Pure Endowment } {}_5E_x := & \\
{}_5E_x = {}_5p_x * v^5 & \texttt{pure endowment}
\end{array}$$

$$\begin{array}{rl}
\texttt{Pure Endowment } {}_5E_x := & \\
{}_5E_x = {}_5p_x * v^5 & \texttt{pure endowment}
\end{array}$$

$$\begin{array}{rl}
\texttt{Term Insurance } A^1_{x+5:\overline{3|}} := & \\
A^1_{x+5:\overline{3|}} = A_{x+5:\overline{3|}} - {}_3E_{x+5} & \texttt{endowment insurance - pure} \\
{}_3E_{x+5} = E_{x+5} * {}_2E_{x+6} & \texttt{pure endowment chain rule} \\
{}_2E_{x+6} = E_{x+6} * E_{x+7} & \texttt{pure endowment chain rule} \\
E_{x+7} = p_{x+7} * v & \texttt{pure endowment} \\
E_{x+6} = p_{x+6} * v & \texttt{pure endowment} \\
E_{x+5} = p_{x+5} * v & \texttt{pure endowment} \\
p_{x+7} = 1 - q_{x+7} & \texttt{complement of mortality} \\
A^1_{x+5:\overline{3|}} = v * [\, q_{x+5} * b + p_{x+5} * A^1_{x+6:\overline{2|}}] & \texttt{backward recursion} \\
A^1_{x+6:\overline{2|}} = v * [\, q_{x+6} * b + p_{x+6} * A^1_{x+7:\overline{1|}}] & \texttt{backward recursion} \\
p_{x+6} = 1 - q_{x+6} & \texttt{complement of mortality} \\
p_{x+5} = 1 - q_{x+5} & \texttt{complement of mortality}
\end{array}$$

$$\begin{array}{rl}
\texttt{Term Insurance } A^1_{x+5:\overline{3|}} := & \\
A^1_{x+5:\overline{3|}} = A_{x+5:\overline{3|}} - {}_3E_{x+5} & \texttt{endowment insurance - pure} \\
{}_3E_{x+5} = E_{x+5} * {}_2E_{x+6} & \texttt{pure endowment chain rule} \\
{}_2E_{x+6} = E_{x+6} * E_{x+7} & \texttt{pure endowment chain rule} \\
E_{x+7} = p_{x+7} * v & \texttt{pure endowment} \\
E_{x+6} = p_{x+6} * v & \texttt{pure endowment} \\
E_{x+5} = p_{x+5} * v & \texttt{pure endowment} \\
p_{x+7} = 1 - q_{x+7} & \texttt{complement of mortality} \\
A^1_{x+5:\overline{3|}} = v * [\, q_{x+5} * b + p_{x+5} * A^1_{x+6:\overline{2|}}] & \texttt{backward recursion} \\
A^1_{x+6:\overline{2|}} = v * [\, q_{x+6} * b + p_{x+6} * A^1_{x+7:\overline{1|}}] & \texttt{backward recursion} \\
p_{x+6} = 1 - q_{x+6} & \texttt{complement of mortality} \\
p_{x+5} = 1 - q_{x+5} & \texttt{complement of mortality}
\end{array}$$

```
3195.118917658744
```

SOA Question 6.17

An insurance company sells special fully discrete two-year endowment insurance policies to smokers (S) and non-smokers (NS) age x. You are given:

- The death benefit is 100,000; the maturity benefit is 30,000
- The level annual premium for non-smoker policies is determined by the equivalence principle
- The annual premium for smoker policies is twice the non-smoker annual premium
- $\mu^{NS}_{x+t} = 0.1. \quad t > 0$
- $q^S_{x+k} = 1.5 q^{NS}_{x+k}$, for $k = 0, 1$
- $i = 0.08$

Calculate the expected present value of the loss at issue random variable on a smoker policy.

```python
print("SOA Question 6.17:  (A) -30000")
x = 0
life = ConstantForce(mu=0.1).set_interest(i=0.08)
A = life.endowment_insurance(x, t=2, b=100000, endowment=30000)
a = life.temporary_annuity(x, t=2)
P = life.gross_premium(a=a, A=A)
print(A, a, P)

life1 = Recursion().set_interest(i=0.08)\
                   .set_q(life.q_x(x, t=1) * 1.5, x=x, t=1)\
                   .set_q(life.q_x(x+1, t=1) * 1.5, x=x+1, t=1)
contract = Contract(premium=P * 2, benefit=100000, endowment=30000)
L = life1.gross_policy_value(x, t=0, n=2, contract=contract)
print(L)
```

```
SOA Question 6.17:  (A) -30000
37251.49857703495 1.8378124241073746 20269.478042694158
```

Term Insurance $A^{1}_{x:\overline{2|}}$:=
$A^{1}_{x:\overline{2|}} = A_{x:\overline{2|}} - {}_{2}E_{x}$ endowment insurance - pure
${}_{2}E_{x} = {}_{2}p_{x} * v^{2}$ pure endowment
${}_{2}p_{x} = p_{x+1} * p_{x}$ survival chain rule
$A^{1}_{x:\overline{2|}} = v * [\, q_{x} * b + p_{x} * A^{1}_{x+1:\overline{1|}}]$ backward recursion
$p_{x+1} = 1 - q_{x+1}$ complement of mortality
$E_{x} = p_{x} * v$ pure endowment
$p_{x} = 1 - q_{x}$ complement of mortality

Temporary Annuity $\ddot{a}_{x:\overline{2|}}$:=
$\ddot{a}_{x:\overline{2|}} = 1 + E_{x} * \ddot{a}_{x+1:\overline{1|}}$ backward recursion
$E_{x} = p_{x} * v$ pure endowment
$p_{x} = 1 - q_{x}$ complement of mortality

Pure Endowment ${}_{2}E_{x}$:=
${}_{2}E_{x} = {}_{2}p_{x} * v^{2}$ pure endowment
${}_{2}p_{x} = p_{x+1} * p_{x}$ survival chain rule
$p_{x} = 1 - q_{x}$ complement of mortality
$p_{x+1} = 1 - q_{x+1}$ complement of mortality

```
-30107.42633581115
```

SOA Question 2.5 : (B) 37.1

You are given the following:

1. $e_{40:20} = 18$
2. $e_{60} = 25$
3. ${}_{20}q_{40} = 0.2$
4. $q_{40} = 0.003$

Calculate e_{41}.

hints:

- solve for e_{40} from limited lifetime formula
- compute e_{41} using backward recursion

```
print("SOA Question 2.5:  (B) 37.1")
life = Recursion(verbose=True).set_e(25, x=60, curtate=True)\
                              .set_q(0.2, x=40, t=20)\
                              .set_q(0.003, x=40)\
                              .set_e(18, x=40, t=20, curtate=True)
e = life.e_x(41, curtate=True)
print(e)
```

```
SOA Question 2.5:  (B) 37.1
```

Lifetime e_{x+41} :=
$e_{x+41} = [\, e_{x+40} - e_{x+40:\overline{1|}}] / p_{x+40}$ forward recursion
$e_{x+40} = e_{x+40:\overline{20|}} + {}_{20}p_{x+40} * e_{x+60}$ backward recursion
${}_{20}p_{x+40} = 1 - {}_{20}q_{x+40}$ complement of mortality
$e_{x+40:\overline{1|}} = p_{x+40}$ 1-year curtate shortcut
$p_{x+40} = 1 - q_{x+40}$ complement of mortality

```
37.11434302908726
```

CHAPTER

THIRTEEN

LIFE TABLE

A life table, from some initial age x_0 to a maximum age ω, represents a survival model with probabilities ${}_tp_x$. A life table is typically tabulated for the number of lives l_x at integer ages x only – fractional age assumptions would be needed to calculate survival probabilities for non-integer ages and durations.

$l_{x_0+t} = l_{x_0}\ {}_tp_{x_0}$.

- l_{x_0} is an arbitrary positive number of lives at starting age x_0 called the radix

$d_x = l_x - l_{x+1}$

- it is usual for a life table to also show the values of d_x, the expected of deaths in the year of age x to $x+1$.

$q_x = \dfrac{d_x}{l_x}$

- the mortality rate can then be derived, which is the probability that a life aged x dies within one year.

Recursion formulas from earlier sections allow the recursive construction of a table of other life contingent values, such as life expectancy, insurance, and annuities, at all ages.

13.1 Methods

The `LifeTable` class specifies a given life table to be the survival model. It inherits all the general methods for computing life contingency risks, and overrides those methods where values can be looked up or calculated from life table entries.

```python
from actuarialmath import LifeTable
import describe
describe.methods(LifeTable)
```

```
class LifeTable - Calculate life table, and iteratively fill in missing values

    Args:
      udd : assume UDD or constant force of mortality for fractional ages
      verbose : whether to echo update steps

    Notes:
      4 types of columns can be loaded and calculated in the life table:

      - 'q' : probability (x) dies in one year
      - 'l' : number of lives aged x
      - 'd' : number of deaths of age x
      - 'p' : probability (x) survives at least one year
```

(continues on next page)

(continued from previous page)

```
    Methods:
    --------

    set_table(fill, minage, maxage, l, d, p, q):
      Update life table

    fill_table(radix):
      Iteratively fill in missing table cells (does not check consistency)

    frame():
      Return life table columns and values in a DataFrame

    __getitem__(col):
      Returns a column of the life table
```

13.2 Examples

The `set_table` method is called to load a life table, by age, with given values of number of lives, number of deaths, mortality rate, and/or survival probability: if the `fill` flag is set to True (by default), `fill_table` is automatically called to fill in any missing values using recursion and identify formulas. All other computational methods can then be called in the usual manner, which will use the survival model provided by the life table.

AMLCR2 Exercise 3.2

You are given the following life table extract.

Age,x	l_x
52	89948
53	89089
54	88176
55	87208
56	86181
57	85093
58	83940
59	82719
60	81429

Calculate

- ${}_{0.2}q_{52.4}$ assuming UDD (fractional age assumption),
- ${}_{0.2}q_{52.4}$ assuming constant force of mortality (fractional age assumption),
- ${}_{5.7}p_{52.4}$ assuming UDD,
- ${}_{5.7}p_{52.4}$ assuming constant force of mortality,
- ${}_{3.2|2.5}q_{52.4}$ assuming UDD, and
- ${}_{3.2|2.5}q_{52.4}$ assuming constant force of mortality.

```
table = {x:l for x,l in zip(range(52, 61),
                            [89948, 89089, 88176, 87208, 86181,
                             85093, 83940, 82719, 81429])}
life1 = LifeTable(udd=True).set_table(l=table)
life2 = LifeTable(udd=False).set_table(l=table)
print([round(r, 6) for r in [life1.q_r(x=52, r=0.4, t=0.2),    # 0.001917        ␣
 ↪
                             life2.q_r(x=52, r=0.4, t=0.2),    # 0.001917        ␣
 ↪
                             life1.p_r(x=52, r=0.4, t=5.7),    # 0.935422        ␣
 ↪
                             life2.p_r(x=52, r=0.4, t=5.7),    # 0.935423        ␣
 ↪
                             life1.q_r(x=52, r=0.4, u=3.2, t=2.5),  # 0.030957   ␣
 ↪
                             life2.q_r(x=52, r=0.4, u=3.2, t=2.5)]])  # 0.030950
```

```
[0.001917, 0.001917, 0.935422, 0.935423, 0.030957, 0.03095]
```

SOA Question 6.53

A warranty pays 2000 at the end of the year of the first failure if a washing machine fails within three years of purchase. The warranty is purchased with a single premium, G, paid at the time of purchase of the washing machine. You are given:

- 10% of the washing machines that are working at the start of each year fail by the end of that year
- i = 0.08
- The sales commission is 35% of G
- G is calculated using the equivalence principle

Calculate G.

```
print("SOA Question 6.53:  (D) 720")
x = 0
life = LifeTable().set_interest(i=0.08)\
                  .set_table(q={x: 0.1, x+1: 0.1, x+2: 0.1})
A = life.term_insurance(x, t=3)
G = life.gross_premium(a=1, A=A, benefit=2000, initial_premium=0.35)
print(A, G)
print(life.frame())
```

```
SOA Question 6.53:  (D) 720
0.23405349794238678 720.1646090534978
          l        d    q    p
0  100000.0  10000.0  0.1  0.9
1   90000.0   9000.0  0.1  0.9
2   81000.0   8100.0  0.1  0.9
3   72900.0      NaN  NaN  NaN
```

SOA Question 6.41

For a special fully discrete 2-year term insurance on (x), you are given:

- $q_x = 0.01$
- $q_{x+1} = 0.02$

- $i = 0.05$
- The death benefit in the first year is 100,000
- Both the benefits and premiums increase by 1% in the second year

Calculate the annual net premium in the first year.

```
print("SOA Question 6.41:  (B) 1417")
x = 0
life = LifeTable().set_interest(i=0.05)\
                  .set_table(q={x:.01, x+1:.02})
P = 1416.93
a = 1 + life.E_x(x, t=1) * 1.01
A = (life.deferred_insurance(x, u=0, t=1)
     + 1.01 * life.deferred_insurance(x, u=1, t=1))
print(a, A)
P = 100000 * A / a
print(P)
print(life.frame())
```

```
SOA Question 6.41:  (B) 1417
1.9522857142857144 0.027662585034013608
1416.9332301924137
          l       d     q     p
0  100000.0  1000.0  0.01  0.99
1   99000.0  1980.0  0.02  0.98
2   97020.0     NaN   NaN   NaN
```

SOA Question 3.11

For the country of Bienna, you are given:

- Bienna publishes mortality rates in biennial form, that is, mortality rates are of the form: ${}_2q_{2x}$, for $x = 0, 1, 2, ...$
- Deaths are assumed to be uniformly distributed between ages $2x$ and $2x + 2$, for $x = 0, 1, 2, ...$
- ${}_2q_{50} = 0.02$
- ${}_2q_{52} = 0.04$ Calculate the probability that (50) dies during the next 2.5 years.

```
print("SOA Question 3.11:  (B) 0.03")
life = LifeTable(udd=True).set_table(q={50//2: .02, 52//2: .04})
print(life.q_r(50//2, t=2.5/2))
print(life.frame())
```

```
SOA Question 3.11:  (B) 0.03
0.0298
           l       d     q     p
25  100000.0  2000.0  0.02  0.98
26   98000.0  3920.0  0.04  0.96
27   94080.0     NaN   NaN   NaN
```

SOA Question 3.5

You are given:

x	60	61	62	63	64	65	66	67
l_x	99,999	88,888	77,777	66,666	55,555	44,444	33,333	22,222

$a = {}_{3.4|2.5}q_{60}$ assuming a uniform distribution of deaths over each year of age

$b = {}_{3.4|2.5}q_{60}$ assuming a constant force of mortality over each year of age

Calculate $100,000(a - b)$

```
print("SOA Question 3.5:  (E) 106")
l = {60+x: n*11111 for x,n in enumerate([9, 8, 7, 6, 5, 4, 3, 2])}
a, b = (LifeTable(udd=udd).set_table(l=l).q_r(60, u=3.4, t=2.5)
        for udd in [True, False])
print(100000 * (a - b))
```

```
SOA Question 3.5:  (E) 106
106.16575827938624
```

SOA Question 3.14

You are given the following information from a life table:

x	l_x	d_x	p_x	q_x
95	–	–	–	0.40
96	–	–	0.20	–
97	–	72	–	1.00

You are also given:

- $l_{90} = 1000$ and $l_{93} = 825$
- Deaths are uniformly distributed over each year of age.

Calculate the probability that (90) dies between ages 93 and 95.5.

```
print("SOA Question 3.14:  (C) 0.345")
life = LifeTable(udd=True).set_table(l={90: 1000, 93: 825},
                                     d={97: 72},
                                     p={96: .2},
                                     q={95: .4, 97: 1})
print(life.q_r(90, u=93-90, t=95.5-93))
print(life.frame())
```

```
SOA Question 3.14:  (C) 0.345
0.345
         l      d    q    p
90  1000.0    NaN  NaN  NaN
93   825.0    NaN  NaN  NaN
95   600.0  240.0  0.4  0.6
96   360.0  288.0  0.8  0.2
97    72.0   72.0  1.0  0.0
98     0.0    NaN  NaN  NaN
```

CHAPTER

FOURTEEN

SULT

14.1 Standard ultimate life table

This tabulates single net premiums and basic functions (whole life and endowment insurances, whole life and temporary annuities and pure endowments) for several time periods at integer ages between 20 and 100 years. According to the SOA's "Excel Workbook for FAM-L Tables", this table was developed from the following assumptions:

- constant interest rate $i = 0.05$
- radix of 100000 initial lives aged 20
- incorporates Makeham's Law as its survival model with $A = 0.00022,\ B = 0.0000027,\ c = 1.124$

14.2 Pure endowment

Pure endowment functions can be calculated from numbers of lives survived and compounded interest rates.

$$_tE_x = v^t\ \frac{l_{x+t}}{l_x}$$

$$^2_tE_x = v^{2t}\ \frac{l_{x+t}}{l_x} = v^t\ _tE_x$$

14.3 Term life insurance

Term life insurance functions can be calculated from whole life insurance and pure endowment table columns.

$$A^1_{x:\overline{t|}} = A_x -\ _tE_x\ A_{x+t} = A_{x:\overline{t|}} -\ _tE_x$$

$$^2A^1_{x:\overline{t|}} = {}^2A_x -\ ^2_tE_x\ {}^2A_{x+t} = {}^2A_x -\ v^t\ _tE_x\ {}^2A_{x+t}$$

14.4 Methods

The `SULT` class implements an instance of a LifeTable, called the standard ultimate life table, which is based on Makeham's Law with parameters specified in SOA's "Excel Workbook for FAM-L Tables"

```
import math
from actuarialmath import SULT
import describe
describe.methods(SULT)
```

```
class SULT - Generates and uses a standard ultimate life table

    Args:
      i : interest rate
      radix : initial number of lives
      minage : minimum age
      maxage : maximum age
      S : survival function, default is Makeham with SOA FAM-L parameters

    Examples:
      >>> sult = SULT()
      >>> a = sult.temporary_annuity(70, t=10)
      >>> A = sult.deferred_annuity(70, u=10)
      >>> P = sult.gross_premium(a=a, A=A, benefit=100000, initial_premium=0.75,
      >>>                        renewal_premium=0.05)

    Methods:
    --------

    frame(minage, maxage):
      Derive FAM-L exam table columns of SULT as a DataFrame

    __getitem__(col):
      Returns a column of the sult table
```

14.5 Examples

SOA Question 6.52

For a fully discrete 10-payment whole life insurance of H on (45), you are given:

- Expenses payable at the beginning of each year are as follows:

Expense Type	First Year	Years 2-10	Years 11+
Per policy	100	20	10
% of Premium	105%	5%	0%

- Mortality follows the Standard Ultimate Life Table
- i = 0.05
- The gross annual premium, calculated using the equivalence principle, is of the form $G = gH + f$, where g is the premium rate per 1 of insurance and f is the per policy fee

Calculate f.

```
print("SOA Question 6.52:  (D) 50.80")
sult = SULT()
a = sult.temporary_annuity(45, t=10)
other_cost = 10 * sult.deferred_annuity(45, u=10)
P = sult.gross_premium(a=a,
                       A=0,
                       benefit=0,
                       initial_premium=1.05,
                       renewal_premium=0.05,
                       initial_policy=100 + other_cost,
                       renewal_policy=20)
print(a, P)
```

```
SOA Question 6.52:  (D) 50.80
8.0750937741422 50.80135534704229
```

SOA Question 6.47

For a 10-year deferred whole life annuity-due with payments of 100,000 per year on (70), you are given:

- Annual gross premiums of G are payable for 10 years
- First year expenses are 75% of premium
- Renewal expenses for years 2 and later are 5% of premium during the premium paying period
- Mortality follows the Standard Ultimate Life Table
- i = 0.05

Calculate G using the equivalence principle.

```
print("SOA Question 6.47:  (D) 66400")
sult = SULT()
a = sult.temporary_annuity(70, t=10)
A = sult.deferred_annuity(70, u=10)
P = sult.gross_premium(a=a, A=A, benefit=100000, initial_premium=0.75,
                        renewal_premium=0.05)
print(P)
```

```
SOA Question 6.47:  (D) 66400
66384.13293704337
```

SOA Question 6.43

For a fully discrete, 5-payment 10-year term insurance of 200,000 on (30), you are given:

- Mortality follows the Standard Ultimate Life Table
- The following expenses are incurred at the beginning of each respective year:

	Percent of Premium	Per Policy	Percent of Premium	Per Policy
	Year 1	Year 1	Years 2 - 10	Years 2 - 10
Taxes	5%	0	5%	0
Commissions	30%	0	10%	0
Maintenance	0%	8	0%	4

- i = 0.05
- $\ddot{a}_{30:\overline{5}|} = 4.5431$

Calculate the annual gross premium using the equivalence principle.

```
print("SOA Question 6.43:  (C) 170")
sult = SULT()
a = sult.temporary_annuity(30, t=5)
A = sult.term_insurance(30, t=10)
other_expenses = 4 * sult.deferred_annuity(30, u=5, t=5)
P = sult.gross_premium(a=a, A=A, benefit=200000, initial_premium=0.35,
                        initial_policy=8 + other_expenses, renewal_policy=4,
                        renewal_premium=0.15)
print(P)
```

```
SOA Question 6.43:  (C) 170
171.22371939459944
```

SOA Question 6.39

XYZ Insurance writes 10,000 fully discrete whole life insurance policies of 1000 on lives age 40 and an additional 10,000 fully discrete whole life policies of 1000 on lives age 80.

XYZ used the following assumptions to determine the net premiums for these policies:

- Mortality follows the Standard Ultimate Life Table
- i = 0.05

During the first ten years, mortality did follow the Standard Ultimate Life Table.

Calculate the average net premium per policy in force received at the beginning of the eleventh year.

```
print("SOA Question 6.39:  (A) 29")
sult = SULT()
P40 = sult.premium_equivalence(sult.whole_life_insurance(40), b=1000)
P80 = sult.premium_equivalence(sult.whole_life_insurance(80), b=1000)
p40 = sult.p_x(40, t=10)
p80 = sult.p_x(80, t=10)
P = (P40 * p40 + P80 * p80) / (p80 + p40)
print(P)
```

```
SOA Question 6.39:  (A) 29
29.033866427845496
```

SOA Question 6.37

For a fully discrete whole life insurance policy of 50,000 on (35), with premiums payable for a maximum of 10 years, you are given:

- Expenses of 100 are payable at the end of each year including the year of death
- Mortality follows the Standard Ultimate Life Table
- i = 0.05

Calculate the annual gross premium using the equivalence principle.

```
print("SOA Question 6.37:  (D) 820")
sult = SULT()
benefits = sult.whole_life_insurance(35, b=50000 + 100)
expenses = sult.immediate_annuity(35, b=100)
a = sult.temporary_annuity(35, t=10)
print(benefits, expenses, a)
print((benefits + expenses) / a)
```

```
SOA Question 6.37:  (D) 820
4836.382819496279 1797.2773668474615 8.092602358383987
819.7190338249138
```

SOA Question 6.35

For a fully discrete whole life insurance policy of 100,000 on (35), you are given:

- First year commissions are 19% of the annual gross premium
- Renewal year commissions are 4% of the annual gross premium
- Mortality follows the Standard Ultimate Life Table
- i = 0.05

Calculate the annual gross premium for this policy using the equivalence principle.

```
print("SOA Question 6.35:  (D) 530")
sult = SULT()
A = sult.whole_life_insurance(35, b=100000)
a = sult.whole_life_annuity(35)
print(sult.gross_premium(a=a, A=A, initial_premium=.19, renewal_premium=.04))
```

```
SOA Question 6.35:  (D) 530
534.4072234303344
```

SOA Question 5.8

For an annual whole life annuity-due of 1 with a 5-year certain period on (55), you are given:

- Mortality follows the Standard Ultimate Life Table
- i = 0.05

Calculate the probability that the sum of the undiscounted payments actually made under this annuity will exceed the expected present value, at issue, of the annuity.

```
print("SOA Question 5.8: (C) 0.92118")
sult = SULT()
a = sult.certain_life_annuity(55, u=5)
print(sult.p_x(55, t=math.floor(a)))
```

```
SOA Question 5.8: (C) 0.92118
0.9211799771029529
```

SOA Question 5.3

You are given:

- Mortality follows the Standard Ultimate Life Table
- Deaths are uniformly distributed over each year of age
- i = 0.05

Calculate $\frac{d}{dt}(\overline{I}\overline{a})_{40:\overline{t}|}$ at $t = 10.5$.

```
print("SOA Question 5.3:  (C) 6.239")
sult = SULT()
t = 10.5
print(t * sult.E_r(40, t=t))
```

```
SOA Question 5.3:  (C) 6.239
6.23871918627528
```

SOA Question 4.17

For a special whole life policy on (48), you are given:

- The policy pays 5000 if the insured's death is before the median curtate future lifetime at issue and 10,000 if death is after the median curtate future lifetime at issue
- Mortality follows the Standard Ultimate Life Table
- Death benefits are paid at the end of the year of death
- i = 0.05

Calculate the actuarial present value of benefits for this policy.

```
print("SOA Question 4.17:  (A) 1126.7")
sult = SULT()
median = sult.Z_t(48, prob=0.5, discrete=False)
benefit = lambda x,t: 5000 if t < median else 10000
print(sult.A_x(48, benefit=benefit))
```

```
SOA Question 4.17:  (A) 1126.7
1126.774772894844
```

SOA Question 4.14

A fund is established for the benefit of 400 workers all age 60 with independent future lifetimes. When they reach age 85, the fund will be dissolved and distributed to the survivors.

The fund will earn interest at a rate of 5% per year.

The initial fund balance, F, is determined so that the probability that the fund will pay at least 5000 to each survivor is 86%, using the normal approximation.

Mortality follows the Standard Ultimate Life Table.

Calculate F.

```
print("SOA Question 4.14:  (E) 390000    ")
sult = SULT()
p = sult.p_x(60, t=85-60)
mean = sult.bernoulli(p)
var = sult.bernoulli(p, variance=True)
F = sult.portfolio_percentile(mean=mean, variance=var, prob=.86, N=400)
print(F * 5000 * sult.interest.v_t(85-60))
```

```
SOA Question 4.14:  (E) 390000
389322.86778416135
```

SOA Question 4.5

For a 30-year term life insurance of 100,000 on (45), you are given:

- The death benefit is payable at the moment of death
- Mortality follows the Standard Ultimate Life Table
- $\delta = 0.05$
- Deaths are uniformly distributed over each year of age

Calculate the 95th percentile of the present value of benefits random variable for this insurance

```
print("SOA Question 4.5:  (C) 35200")
sult = SULT(udd=True).set_interest(delta=0.05)
Z = 100000 * sult.Z_from_prob(45, prob=0.95, discrete=False)
print(Z)
```

```
SOA Question 4.5:  (C) 35200
35187.952037196534
```

SOA Question 3.9

A father-son club has 4000 members, 2000 of which are age 20 and the other 2000 are age 45. In 25 years, the members of the club intend to hold a reunion. You are given:

- All lives have independent future lifetimes.
- Mortality follows the Standard Ultimate Life Table.

Using the normal approximation, without the continuity correction, calculate the 99th percentile of the number of surviving members at the time of the reunion.

```
print("SOA Question 3.9:  (E) 3850")
sult = SULT()
p1 = sult.p_x(20, t=25)
p2 = sult.p_x(45, t=25)
mean = sult.bernoulli(p1) * 2000 + sult.bernoulli(p2) * 2000
var = (sult.bernoulli(p1, variance=True) * 2000
       + sult.bernoulli(p2, variance=True) * 2000)
print(sult.portfolio_percentile(mean=mean, variance=var, prob=.99))
```

```
SOA Question 3.9:  (E) 3850
3850.144345130047
```

SOA Question 3.8

A club is established with 2000 members, 1000 of exact age 35 and 1000 of exact age 45. You are given:

- Mortality follows the Standard Ultimate Life Table
- Future lifetimes are independent
- N is the random variable for the number of members still alive 40 years after the club is established

Using the normal approximation, without the continuity correction, calculate the smallest n such that $Pr(N \geq n) \leq 0.05$.

```
print("SOA Question 3.8:  (B) 1505")
sult = SULT()
p1 = sult.p_x(35, t=40)
p2 = sult.p_x(45, t=40)
mean = sult.bernoulli(p1) * 1000 + sult.bernoulli(p2) * 1000
var = (sult.bernoulli(p1, variance=True) * 1000
        + sult.bernoulli(p2, variance=True) * 1000)
print(sult.portfolio_percentile(mean=mean, variance=var, prob=.95))
```

```
SOA Question 3.8:  (B) 1505
1504.8328375406456
```

SOA Question 3.4

The SULT Club has 4000 members all age 25 with independent future lifetimes. The mortality for each member follows the Standard Ultimate Life Table.

Calculate the largest integer N, using the normal approximation, such that the probability that there are at least N survivors at age 95 is at least 90%.

```
print("SOA Question 3.4:  (B) 815")
sult = SULT()
mean = sult.p_x(25, t=95-25)
var = sult.bernoulli(mean, variance=True)
print(sult.portfolio_percentile(N=4000, mean=mean, variance=var, prob=.1))
```

```
SOA Question 3.4:  (B) 815
815.0943255167722
```

Generate SULT Table:

```
import pandas as pd
print("Standard Ultimate Life Table at i=0.05")
pd.set_option('display.max_rows', None)
sult.frame()
```

```
Standard Ultimate Life Table at i=0.05
```

```
          l_x       q_x      a_x      A_x      2A_x  a_x:10   A_x:10   a_x:20
20   100000.0  0.000250  19.9664  0.04922  0.00580  8.0991  0.61433  13.0559  \
21    99975.0  0.000253  19.9197  0.05144  0.00614  8.0990  0.61433  13.0551
22    99949.7  0.000257  19.8707  0.05378  0.00652  8.0988  0.61434  13.0541
23    99924.0  0.000262  19.8193  0.05622  0.00694  8.0986  0.61435  13.0531
24    99897.8  0.000267  19.7655  0.05879  0.00739  8.0983  0.61437  13.0519
```

(continues on next page)

(continued from previous page)

```
25    99871.1  0.000273  19.7090  0.06147  0.00788  8.0981  0.61438  13.0506
26    99843.8  0.000280  19.6499  0.06429  0.00841  8.0978  0.61439  13.0491
27    99815.9  0.000287  19.5878  0.06725  0.00900  8.0974  0.61441  13.0474
28    99787.2  0.000296  19.5228  0.07034  0.00964  8.0970  0.61443  13.0455
29    99757.7  0.000305  19.4547  0.07359  0.01033  8.0966  0.61445  13.0434
30    99727.3  0.000315  19.3834  0.07698  0.01109  8.0961  0.61447  13.0410
31    99695.8  0.000327  19.3086  0.08054  0.01192  8.0956  0.61450  13.0384
32    99663.2  0.000341  19.2303  0.08427  0.01281  8.0949  0.61453  13.0354
33    99629.3  0.000356  19.1484  0.08817  0.01379  8.0943  0.61456  13.0320
34    99593.8  0.000372  19.0626  0.09226  0.01486  8.0935  0.61460  13.0282
35    99556.7  0.000391  18.9728  0.09653  0.01601  8.0926  0.61464  13.0240
36    99517.8  0.000412  18.8788  0.10101  0.01727  8.0916  0.61468  13.0192
37    99476.7  0.000436  18.7805  0.10569  0.01863  8.0905  0.61474  13.0138
38    99433.3  0.000463  18.6777  0.11059  0.02012  8.0893  0.61480  13.0078
39    99387.3  0.000493  18.5701  0.11571  0.02173  8.0879  0.61486  13.0011
40    99338.3  0.000527  18.4578  0.12106  0.02347  8.0863  0.61494  12.9935
41    99285.9  0.000565  18.3403  0.12665  0.02536  8.0846  0.61502  12.9850
42    99229.8  0.000608  18.2176  0.13249  0.02741  8.0826  0.61511  12.9754
43    99169.4  0.000656  18.0895  0.13859  0.02963  8.0804  0.61522  12.9647
44    99104.3  0.000710  17.9558  0.14496  0.03203  8.0779  0.61534  12.9526
45    99033.9  0.000771  17.8162  0.15161  0.03463  8.0751  0.61547  12.9391
46    98957.6  0.000839  17.6706  0.15854  0.03744  8.0720  0.61562  12.9240
47    98874.5  0.000916  17.5189  0.16577  0.04047  8.0684  0.61579  12.9070
48    98783.9  0.001003  17.3607  0.17330  0.04374  8.0645  0.61598  12.8880
49    98684.9  0.001100  17.1960  0.18114  0.04727  8.0600  0.61619  12.8667
50    98576.4  0.001209  17.0245  0.18931  0.05108  8.0550  0.61643  12.8428
51    98457.2  0.001331  16.8461  0.19780  0.05517  8.0494  0.61670  12.8161
52    98326.2  0.001469  16.6606  0.20664  0.05957  8.0431  0.61700  12.7862
53    98181.8  0.001623  16.4678  0.21582  0.06430  8.0360  0.61733  12.7527
54    98022.4  0.001797  16.2676  0.22535  0.06938  8.0281  0.61771  12.7154
55    97846.2  0.001993  16.0599  0.23524  0.07483  8.0192  0.61813  12.6737
56    97651.2  0.002212  15.8444  0.24550  0.08067  8.0092  0.61861  12.6271
57    97435.2  0.002459  15.6212  0.25613  0.08692  7.9980  0.61914  12.5752
58    97195.6  0.002736  15.3901  0.26714  0.09360  7.9854  0.61974  12.5174
59    96929.6  0.003048  15.1511  0.27852  0.10073  7.9713  0.62041  12.4531
60    96634.1  0.003398  14.9041  0.29028  0.10834  7.9555  0.62116  12.3816
61    96305.8  0.003792  14.6491  0.30243  0.11644  7.9379  0.62201  12.3024
62    95940.6  0.004234  14.3861  0.31495  0.12506  7.9181  0.62295  12.2145
63    95534.4  0.004730  14.1151  0.32785  0.13421  7.8960  0.62400  12.1174
64    95082.5  0.005288  13.8363  0.34113  0.14392  7.8712  0.62518  12.0101
65    94579.7  0.005915  13.5498  0.35477  0.15420  7.8435  0.62650  11.8920
66    94020.3  0.006619  13.2557  0.36878  0.16507  7.8126  0.62797  11.7622
67    93398.1  0.007409  12.9542  0.38313  0.17654  7.7781  0.62961  11.6199
68    92706.1  0.008297  12.6456  0.39783  0.18862  7.7396  0.63145  11.4643
69    91936.9  0.009294  12.3302  0.41285  0.20133  7.6968  0.63349  11.2949
70    91082.4  0.010413  12.0083  0.42818  0.21467  7.6491  0.63576  11.1109
71    90134.0  0.011670  11.6803  0.44379  0.22864  7.5961  0.63828  10.9118
72    89082.1  0.013081  11.3468  0.45968  0.24324  7.5373  0.64108  10.6974
73    87916.8  0.014664  11.0081  0.47580  0.25847  7.4721  0.64419  10.4675
74    86627.6  0.016440  10.6649  0.49215  0.27433  7.3999  0.64762  10.2221
75    85203.5  0.018433  10.3178  0.50868  0.29079  7.3203  0.65142   9.9616
76    83632.9  0.020668   9.9674  0.52536  0.30783  7.2325  0.65560   9.6866
77    81904.3  0.023175   9.6145  0.54217  0.32544  7.1360  0.66019   9.3980
78    80006.2  0.025984   9.2598  0.55906  0.34359  7.0302  0.66523   9.0970
79    77927.4  0.029132   8.9042  0.57599  0.36224  6.9146  0.67074   8.7850
80    75657.2  0.032658   8.5484  0.59293  0.38134  6.7885  0.67674   8.4639
```

(continues on next page)

(continued from previous page)

```
81    73186.3  0.036607   8.1934  0.60984  0.40086  6.6517  0.68325   8.1354
82    70507.2  0.041025   7.8401  0.62666  0.42075  6.5037  0.69030   7.8018
83    67614.6  0.045968   7.4893  0.64336  0.44094  6.3443  0.69789   7.4651
84    64506.5  0.051493   7.1421  0.65990  0.46137  6.1735  0.70602   7.1275
85    61184.9  0.057665   6.7993  0.67622  0.48199  5.9915  0.71469   6.7910
86    57656.7  0.064554   6.4619  0.69229  0.50272  5.7986  0.72388   6.4574
87    53934.7  0.072237   6.1308  0.70806  0.52349  5.5954  0.73355   6.1285
88    50038.6  0.080798   5.8068  0.72349  0.54422  5.3828  0.74368   5.8057
89    45995.6  0.090326   5.4908  0.73853  0.56484  5.1620  0.75419   5.4903
90    41841.1  0.100917   5.1835  0.75317  0.58528  4.9346  0.76502   5.1833
91    37618.6  0.112675   4.8858  0.76735  0.60545  4.7021  0.77609   4.8857
92    33379.9  0.125708   4.5981  0.78104  0.62529  4.4665  0.78731   4.5981
93    29183.8  0.140128   4.3213  0.79423  0.64472  4.2299  0.79858   4.3213
94    25094.3  0.156052   4.0556  0.80688  0.66368  3.9945  0.80979   4.0556
95    21178.3  0.173599   3.8017  0.81897  0.68209  3.7624  0.82084   3.8017
96    17501.8  0.192887   3.5597  0.83049  0.69991  3.5356  0.83164   3.5597
97    14125.9  0.214030   3.3300  0.84143  0.71708  3.3159  0.84210   3.3300
98    11102.5  0.237134   3.1127  0.85177  0.73356  3.1050  0.85214   3.1127
99     8469.7  0.262294   2.9079  0.86153  0.74930  2.9039  0.86172   2.9079
100    6248.2  0.289584   2.7156  0.87068  0.76427  2.7137  0.87078   2.7156

      A_x:20    5_E_x   10_E_x   20_E_x
20   0.37829  0.78252  0.61224  0.37440
21   0.37833  0.78250  0.61220  0.37429
22   0.37837  0.78248  0.61215  0.37417
23   0.37842  0.78245  0.61210  0.37404
24   0.37848  0.78243  0.61205  0.37390
25   0.37854  0.78240  0.61198  0.37373
26   0.37862  0.78236  0.61191  0.37354
27   0.37869  0.78233  0.61183  0.37334
28   0.37878  0.78229  0.61174  0.37310
29   0.37888  0.78224  0.61163  0.37284
30   0.37900  0.78219  0.61152  0.37254
31   0.37913  0.78213  0.61139  0.37221
32   0.37927  0.78206  0.61124  0.37183
33   0.37943  0.78199  0.61108  0.37141
34   0.37961  0.78190  0.61090  0.37094
35   0.37981  0.78181  0.61069  0.37041
36   0.38004  0.78170  0.61046  0.36982
37   0.38029  0.78158  0.61020  0.36915
38   0.38058  0.78145  0.60990  0.36841
39   0.38090  0.78130  0.60957  0.36757
40   0.38126  0.78113  0.60920  0.36663
41   0.38167  0.78094  0.60879  0.36558
42   0.38212  0.78072  0.60832  0.36440
43   0.38263  0.78048  0.60780  0.36307
44   0.38321  0.78021  0.60721  0.36159
45   0.38385  0.77991  0.60655  0.35994
46   0.38457  0.77956  0.60581  0.35809
47   0.38538  0.77918  0.60498  0.35601
48   0.38629  0.77875  0.60404  0.35370
49   0.38730  0.77827  0.60299  0.35112
50   0.38844  0.77772  0.60182  0.34824
51   0.38971  0.77711  0.60050  0.34503
52   0.39113  0.77643  0.59902  0.34146
53   0.39273  0.77566  0.59736  0.33749
```

(continues on next page)

(continued from previous page)

```
54   0.39451  0.77479  0.59550  0.33308
55   0.39649  0.77382  0.59342  0.32819
56   0.39871  0.77273  0.59109  0.32279
57   0.40118  0.77151  0.58848  0.31681
58   0.40393  0.77014  0.58556  0.31024
59   0.40700  0.76860  0.58229  0.30300
60   0.41040  0.76687  0.57864  0.29508
61   0.41417  0.76493  0.57457  0.28641
62   0.41836  0.76276  0.57003  0.27698
63   0.42298  0.76033  0.56496  0.26674
64   0.42809  0.75760  0.55932  0.25569
65   0.43371  0.75455  0.55305  0.24381
66   0.43990  0.75114  0.54609  0.23112
67   0.44667  0.74732  0.53836  0.21764
68   0.45408  0.74305  0.52981  0.20343
69   0.46215  0.73828  0.52036  0.18856
70   0.47091  0.73295  0.50994  0.17313
71   0.48039  0.72701  0.49848  0.15730
72   0.49060  0.72039  0.48590  0.14122
73   0.50155  0.71303  0.47215  0.12511
74   0.51323  0.70483  0.45715  0.10918
75   0.52564  0.69574  0.44085  0.09368
76   0.53873  0.68566  0.42323  0.07887
77   0.55247  0.67450  0.40427  0.06500
78   0.56681  0.66217  0.38396  0.05230
79   0.58166  0.64859  0.36235  0.04096
80   0.59696  0.63365  0.33952  0.03113
81   0.61260  0.61727  0.31556  0.02286
82   0.62848  0.59936  0.29064  0.01616
83   0.64452  0.57985  0.26498  0.01094
84   0.66059  0.55868  0.23882  0.00706
85   0.67662  0.53581  0.21250  0.00431
86   0.69250  0.51122  0.18635  0.00248
87   0.70817  0.48492  0.16079  0.00133
88   0.72354  0.45697  0.13621  0.00066
89   0.73856  0.42748  0.11305  0.00030
90   0.75317  0.39659  0.09168  0.00012
91   0.76735  0.36453  0.07244  0.00005
92   0.78104  0.33158  0.05559  0.00002
93   0.79423  0.29808  0.04128  0.00000
94   0.80688  0.26445  0.02955  0.00000
95   0.81897  0.23116  0.02029  0.00000
96   0.83049  0.19872  0.01330  0.00000
97   0.84143  0.16765  0.00827  0.00000
98   0.85177  0.13850  0.00485  0.00000
99   0.86153  0.11173  0.00266  0.00000
100  0.87068  0.08777  0.00136  0.00000
```

CHAPTER FIFTEEN

SELECT LIFE TABLE

15.1 Select and ultimate life model

A newly selected policyholder is in the best health condition possible, compared to the general population with the same age. The life table can be expanded to tabulate the select period when selection has an effect on mortality. Since this selection process wears off after a few years, the ultimate part of the table can be then be used when select age is assumed to no longer have an effect on mortality.

- Future survival probabilities depend on the individual's current age and on the age at which the individual joined the group (i.e. was *selected*). Current age is written $[x] + s$, where x is the selected age and s is the number of years after selection.
- If an individual joined the group more than d years ago (called the *select period*), future survival probabilities (called the *ultimate mortality*) depend only on current age. The initial selection effect is assumed to have worn off after d years. Current age can be written as $x + s$ after the select period $s \geq d$

Select life tables reflect duration as well as age during the select period. A select and ultimate mortality table is shown in tabular form by listing agex vertically and the selection durations horizontally. If the select period is n, there are n columns followed by a column with ultimate mortality. To find the mortality at a duration after a selection age, we read across the row corresponding to that selection age then continue down the last column.

Notation for select survival models:

${}_tp_{[x]+s}$ = Pr(a life aged $x + s$, selected at age x, survives to age $x + s + t$)

- defines survival probability in the select period

${}_tq_{[x]+s}$ = Pr(a life aged $x + s$, selected at age x, dies before age $x + s + t$)

- defines mortality rate in the select period

$l_{[x]+s} = \dfrac{l_{x+d}}{{}_{d-s}p_{[x]+s}}$ = number of lives, selected at age x, who are aged $x + s$, given that l_{x+d} survived to age $x + d$.

- defines the life table within the select period, by working backwards from the value of l_{x+d} in the ultimate part of the table which only depends on current age.

With a select period d and for $s \geq d$ (i.e. durations beyond the select period) the values of $p_{[x-s]+s}, q_{[x-s]+s}, l_{[x-s]+s}$ depend only on current age x and not on s. So for $s \geq d$, these terms are all equal to and can be written simply as p_x, q_x, l_x respectively.

15.2 Methods

The `SelectLife` class specifies a given select life table to be the survival model. It inherits all the general methods for computing life contingency risks, and overrides those methods where values can be looked up or calculated from select life table entries.

```
from actuarialmath import SelectLife
import describe
describe.methods(SelectLife)
```

```
class SelectLife - Calculate select life table, and iteratively fill in missing␣
↪values

    Args:
      periods : number of select period years
      verbose : whether to echo update steps

    Notes:
      6 types of columns can be loaded and calculated in the select table:

      - 'q' : probability [x]+s dies in one year
      - 'l' : number of lives aged [x]+s
      - 'd' : number of deaths of age [x]+s
      - 'A' : whole life insurance
      - 'a' : whole life annuity
      - 'e' : expected future curtate lifetime of [x]+s

    Methods:
    --------

    set_table(fill, l, d, q, A, a, e):
      Update from table, every age has row for all select durations

    set_select(s, age_selected, fill, l, d, q, A, a, e):
      Update a table column, for a particular duration s in the select period

    fill_table(radix):
      Fills in missing table values (does not check for consistency)

    __getitem__(table):
      Returns values from a select and ultimate table

    frame(table):
      Returns select and ultimate table values as a DataFrame

    l_x(x, s):
      Returns number of lives aged [x]+s computed from select table

    p_x(x, s, t):
      t_p_[x]+s by chain rule: prod(1_p_[x]+s+y) for y in range(t)

    q_x(x, s, t, u):
      t|u_q_[x]+s = [x]+s survives u years, does not survive next t

    e_x(x, s, t, curtate):
      Returns expected life time computed from select table
```

(continues on next page)

(continued from previous page)

```
    A_x(x, s, moment, discrete, kwargs):
      Returns insurance value computed from select table

    a_x(x, s, moment, discrete, kwargs):
      Returns annuity value computed from select table
```

15.3 Examples

The `set_table` method is called to load a life table, by age and duration, with given values of number of lives, number of deaths, mortality rate, insurance, annuity and/or life expectancy. The `set_select` method can also be called instead to update values of the same functions but for a particular duration s only. If the `fill` flag is set to True, `fill_table` is automatically called to fill in any missing values using recursion and identify formulas. All other computational methods can then be called in the usual manner, which will use the survival model provided by the select life table.

SOA Question 3.2:

You are given:

- The following extract from a mortality table with a one-year select period:

x	$l_{[x]}$	$d_{[x]}$	l_{x+1}	$x+1$
65	1000	40	–	66
66	955	45	–	67

- Deaths are uniformly distributed over each year of age

$\overset{\circ}{e}_{[65]} = 15.0$

Calculate $\overset{\circ}{e}_{[66]}$.

```
print("SOA Question 3.2:  (D) 14.7")
e_curtate = SelectLife.e_approximate(e_complete=15)
life = SelectLife(udd=True).set_table(l={65: [1000, None,],
                                         66: [955, None]},
                                      e={65: [e_curtate, None]},
                                      d={65: [40, None,],
                                         66: [45, None]})
print(life.e_r(66))
print(life.frame('e'))
```

```
SOA Question 3.2:  (D) 14.7
14.67801047120419
e_[x]+s:         0          1
Age
65         14.50000  14.104167
66         14.17801  13.879121
```

SOA Question 4.16

You are given the following extract of ultimate mortality rates from a two-year select and ultimate mortality table:

x	q_x
50	0.045
51	0.050
52	0.055
53	0.060

The select mortality rates satisfy the following:

- $q_{[x]} = 0.7q_x$
- $q_{[x]+1} = 0.8q_{x+1}$

You are also given that $i = 0.04$.

Calculate $A^{1}_{[50]:\overline{3|}}$.

```
print("SOA Question 4.16:  (D) .1116")
q = [.045, .050, .055, .060]
q_ = {50+x: [0.7 * q[x] if x < 4 else None,
             0.8 * q[x+1] if x+1 < 4 else None,
             q[x+2] if x+2 < 4 else None]
      for x in range(4)}
life = SelectLife().set_table(q=q_).set_interest(i=.04)
print(life.term_insurance(50, t=3))
```

```
SOA Question 4.16:  (D) .1116
0.1115661982248521
```

SOA Question 4.13

For a 2-year deferred, 2-year term insurance of 2000 on [65], you are given:

- The following select and ultimate mortality table with a 3-year select period:

x	$q_{[x]}$	$q_{[x]+1}$	$q_{[x]+2}$	q_{x+3}	$x+3$
65	0.08	0.10	0.12	0.14	68
66	0.09	0.11	0.13	0.15	69
67	0.10	0.12	0.14	0.16	70
68	0.11	0.13	0.15	0.17	71
69	0.12	0.14	0.16	0.18	72

- $i = 0.04$
- The death benefit is payable at the end of the year of death

Calculate the actuarial present value of this insurance.

```
print("SOA Question 4.13:  (C) 350 ")
life = SelectLife().set_interest(i=0.04)\
                   .set_table(q={65: [.08, .10, .12, .14],
                                 66: [.09, .11, .13, .15],
                                 67: [.10, .12, .14, .16],
                                 68: [.11, .13, .15, .17],
                                 69: [.12, .14, .16, .18]})
print(life.deferred_insurance(65, t=2, u=2, b=2000))
```

```
SOA Question 4.13:  (C) 350
351.0578236056159
```

SOA Question 3.13

A life is subject to the following 3-year select and ultimate table:

$[x]$	$\ell_{[x]}$	$\ell_{[x]+1}$	$\ell_{[x]+2}$	ℓ_{x+3}	$x+3$
55	10,000	9,493	8,533	7,664	58
56	8,547	8,028	6,889	5,630	59
57	7,011	6,443	5,395	3,904	60
58	5,853	4,846	3,548	2,210	61

You are also given:

- $e_{60} = 1$
- Deaths are uniformly distributed over each year of age

Calculate $\overset{\circ}{e}_{[58]+2}$.

```
print("SOA Question 3.13:  (B) 1.6")
life = SelectLife().set_table(l={55: [10000, 9493, 8533, 7664],
                                 56: [8547, 8028, 6889, 5630],
                                 57: [7011, 6443, 5395, 3904],
                                 58: [5853, 4846, 3548, 2210]},
                              e={57: [None, None, None, 1]})
print(life.e_r(58, s=2))
```

```
SOA Question 3.13:  (B) 1.6
1.6003382187147688
```

SOA Question 3.12

X and Y are both age 61. X has just purchased a whole life insurance policy. Y purchased a whole life insurance policy one year ago.

Both X and Y are subject to the following 3-year select and ultimate table:

x	$\ell_{[x]}$	$\ell_{[x]+1}$	$\ell_{[x]+2}$	ℓ_{x+3}	$x+3$
60	10,000	9,600	8,640	7,771	63
61	8,654	8,135	6,996	5,737	64
62	7,119	6,549	5,501	4,016	65
63	5,760	4,954	3,765	2,410	66

The force of mortality is constant over each year of age.

Calculate the difference in the probability of survival to age 64.5 between X and Y.

```
print("SOA Question 3.12: (C) 0.055 ")
life = SelectLife(udd=False).set_table(l={60: [10000, 9600, 8640, 7771],
                                          61: [8654, 8135, 6996, 5737],
                                          62: [7119, 6549, 5501, 4016],
```

(continues on next page)

(continued from previous page)

```
                                            63: [5760, 4954, 3765, 2410]})
print(life.q_r(60, s=1, t=3.5) - life.q_r(61, s=0, t=3.5))
```

```
SOA Question 3.12: (C) 0.055
0.05465655938591829
```

SOA Question 3.7

For a mortality table with a select period of two years, you are given:

x	$q_{[x]}$	$q_{[x]+1}$	q_{x+2}	$x+2$
50	0.0050	0.0063	0.0080	52
51	0.0060	0.0073	0.0090	53
52	0.0070	0.0083	0.0100	54
53	0.0080	0.0093	0.0110	55

The force of mortality is constant between integral ages.

Calculate $1000\ {}_{2.5}q_{[50]+0.4}$.

```
print("SOA Question 3.7: (b) 16.4")
life = SelectLife().set_table(q={50: [.0050, .0063, .0080],
                                    51: [.0060, .0073, .0090],
                                    52: [.0070, .0083, .0100],
                                    53: [.0080, .0093, .0110]})
print(1000*life.q_r(50, s=0, r=0.4, t=2.5))
```

```
SOA Question 3.7: (b) 16.4
16.420207214428586
```

SOA Question 3.6

You are given the following extract from a table with a 3-year select period:

x	$q_{[x]}$	$q_{[x]+1}$	$q_{[x]+2}$	q_{x+3}	$x+3$
60	0.09	0.11	0.13	0.15	63
61	0.10	0.12	0.14	0.16	64
62	0.11	0.13	0.15	0.17	65
63	0.12	0.14	0.16	0.18	66
64	0.13	0.15	0.17	0.19	67

$e_{64} = 5.10$

Calculate $e_{[61]}$.

```
print("SOA Question 3.6:  (D) 5.85")
life = SelectLife().set_table(q={60: [.09, .11, .13, .15],
                                 61: [.1, .12, .14, .16],
                                 62: [.11, .13, .15, .17],
                                 63: [.12, .14, .16, .18],
                                 64: [.13, .15, .17, .19]},
```

(continues on next page)

(continued from previous page)

```
                               e={61: [None, None, None, 5.1]})
print(life.e_x(61))
```

```
SOA Question 3.6:  (D) 5.85
5.846832
```

SOA Question 3.3

You are given:

- An excerpt from a select and ultimate life table with a select period of 2 years:

x	$\ell_{[x]}$	$\ell_{[x]+1}$	ℓ_{x+2}	$x+2$
50	99,000	96,000	93,000	52
51	97,000	93,000	89,000	53
52	93,000	88,000	83,000	54
53	90,000	84,000	78,000	55

- Deaths are uniformly distributed over each year of age

Calculate $10,000\ {}_{2.2}q_{[51]+0.5}$.

```
print("SOA Question 3.3:  (E) 1074")
life = SelectLife().set_table(l={50: [99, 96, 93],
                                 51: [97, 93, 89],
                                 52: [93, 88, 83],
                                 53: [90, 84, 78]})
print(10000*life.q_r(51, s=0, r=0.5, t=2.2))
```

```
SOA Question 3.3:  (E) 1074
1073.684210526316
```

SOA Question 3.1

You are given:

- An excerpt from a select and ultimate life table with a select period of 3 years:

x	$\ell_{[x]}$	$\ell_{[x]+1}$	$\ell_{[x]+2}$	ℓ_{x+3}	$x+3$
60	80,000	79,000	77,000	74,000	63
61	78,000	76,000	73,000	70,000	64
62	75,000	72,000	69,000	67,000	65
63	71,000	68,000	66,000	65,000	66

- Deaths follow a constant force of mortality over each year of age

Calculate $1000\ {}_{23}q_{[60]+0.75}$.

```
print("SOA Question 3.1:  (B) 117")
life = SelectLife().set_table(l={60: [80000, 79000, 77000, 74000],
                                    61: [78000, 76000, 73000, 70000],
                                    62: [75000, 72000, 69000, 67000],
```

(continues on next page)

(continued from previous page)

```
                                    63: [71000, 68000, 66000, 65000]})
print(1000*life.q_r(60, s=0, r=0.75, t=3, u=2))
```

```
SOA Question 3.1:  (B) 117
116.7192429022082
```

show verbose calculations:

```
table={21: [0.00120, 0.00150, 0.00170, 0.00180],
       22: [0.00125, 0.00155, 0.00175, 0.00185],
       23: [0.00130, 0.00160, 0.00180, 0.00195]}
life = SelectLife(verbose=True).set_table(q=table)
print(life.p_x(x=21, s=1, t=4))   # 0.9931
life.frame('l')
```

```
1 l(x=21, s=1) = 99880.0
2 l(x=21, s=2) = 99730.18000000001
3 l(x=21, s=3) = 99560.63869400001
4 l(x=22, s=3) = 99381.4295443508
5 l(x=23, s=3) = 99197.57389969376
6 d(x=21, s=0) = 120.0
7 d(x=21, s=1) = 149.81999999999243
8 d(x=21, s=2) = 169.54130599999917
9 d(x=21, s=3) = 179.20914964920667
10 d(x=22, s=3) = 183.85564465704374
11 l(x=22, s=2) = 99555.65193523747
12 l(x=23, s=2) = 99376.45151241611
13 d(x=22, s=2) = 174.22239088667266
14 d(x=23, s=2) = 178.87761272235366
15 l(x=22, s=1) = 99710.2027494992
16 l(x=23, s=1) = 99535.70864625012
17 d(x=22, s=1) = 154.55081426173274
18 d(x=23, s=1) = 159.2571338340058
19 l(x=22, s=0) = 99834.9964951181
20 l(x=23, s=0) = 99665.27350180245
21 d(x=22, s=0) = 124.79374561889563
22 d(x=23, s=0) = 129.56485555233667
0.9931675400449915
```

```
l_[x]+s:              0             1             2             3
Age
21        100000.000000  99880.000000  99730.180000  99560.638694
22         99834.996495  99710.202749  99555.651935  99381.429544
23         99665.273502  99535.708646  99376.451512  99197.573900
```

CHAPTER
SIXTEEN

MORTALITY LAWS

Another approach to defining a survival model is to fit a parametric function, which may have convenient properties that simplify computations. When using these special mortality laws for lifetime distribution, shortcut formulas may be available without the need for numerical integration.

16.1 Uniform distribution

If deaths are uniformly distributed then insurance benefits are paid out in a level manner. A convenient property of a uniform distribution on $[0, \theta]$ is that it has a midrange mean and median of $\theta/2$

$$l_x \sim \omega - x$$

$${}_tp_x = \frac{\omega - (x+t)}{\omega - x}$$

- number of lives, and survival function, are linearly declining with age

$$\mu_{x+t} = f_x(t) = \frac{1}{\omega - x - t}$$

- the force of mortality and lifetime density function are identical

$$\mathring{e}_x = \frac{\omega - x}{2}$$

- expected future lifetime is the mean of a uniform distribution

$$\mathring{e}_{x:\overline{n|}} = {}_np_x \, n + {}_nq_x \, \frac{n}{2}$$

- expected limited lifetime is the weighted sum of the mean lifetime of deaths (for those who died) and the limit n (for those who survived)

$Var(T_x) = \dfrac{(\omega - x)^2}{12}

- variance of complete future lifetime is the variance of a uniform distribution

$${}_nE_x = v^n \frac{\omega - (x+n)}{\omega - x}$$

- pure endowment shortcut by substituting in formula for the survival function

$$\bar{A}_x = \frac{\bar{a}_{\overline{\omega - x|}}}{\omega - x}$$

- it is not necessary to carry out integration since whole life insurance is equivalent to an annuity-certain that pays $\dfrac{1}{\omega}$ per year. To calculate the second moment of insurance, simply double the force of interest in the annuity-certain in the shortcut formula.

$\bar{A}^{1}_{x:\overline{n|}} = \dfrac{\bar{a}_{\overline{n|}}}{\omega - x}$

- term insurance is an annuity-certain that pays $\dfrac{1}{\omega}$ per year for the term of the insurance.

16.2 Beta distribution

This is a generalized version of the uniform distribution with two parameters α, ω. The uniform is a special case of the Beta distribution with $\alpha = 1$. However, unlike the uniform, its density function is longer constant.

$l_x \sim (\omega - x)^{\alpha}$

$f_x(t) = \dfrac{\alpha(\omega - x - t)^{\alpha-1}}{(\omega - x)\alpha}$

$\mu_x = \dfrac{\alpha}{\omega - x}$

${}_tp_x = (\dfrac{\omega - (x+t)}{\omega - x})^{\alpha}$

$\overset{\circ}{e}_x = \dfrac{\omega - x}{\alpha + 1}$

$Var(T_x) = \dfrac{\alpha(\omega - x)^2}{(1+\alpha)^2(2+\alpha)}$

16.3 Gompertz's Law

Gompertz proposed this law based on observations that the logarithm of the force of mortality appeared to fit well to a straight line for ages above 20.

$\mu_x = Bc^x$

${}_tp_x = e^{\frac{Bc^x}{\ln c}(c^t - 1)}$

- the force of mortality is modeled by two parameters B and $c > 1$

16.4 Makeham's Law

Makeham improved Gompertz's law by adding a third parameters $A \geq -B$. This includes represents a constant element in the force of mortality that does not depend on age.

$\mu_x = A + Bc^x$

${}_tp_x = e^{\frac{Bc^x}{\ln c}(c^t - 1) - At}$

- the force of mortality is modeled with three parameters $c > 1,\ B > 0,\ A \geq -B$

16.5 Methods

The `MortalityLaws` class, and `Beta`, `Uniform`, `Makeham` and `Gompertz` subclasses, specify continuous functions for survival distributions. These classes inherit all the general methods for computing life contingency risks, and override those methods with shortcut formulas that may be available when assuming their respective mortality laws.

```
from actuarialmath import MortalityLaws, Uniform, Beta, Makeham, Gompertz
import describe
describe.methods(MortalityLaws)
describe.methods(Uniform)
describe.methods(Beta)
describe.methods(Gompertz)
describe.methods(Makeham)
```

```
class MortalityLaws - Apply shortcut formulas for special mortality laws

    Methods:
    --------

    l_r(x, s, r):
      Fractional lives given special mortality law: l_[x]+s+r

    p_r(x, s, r, t):
      Fractional age survival probability given special mortality law

    q_r(x, s, r, t, u):
      Fractional age deferred mortality given special mortality law

    mu_r(x, s, r):
      Fractional age force of mortality given special mortality law

    f_r(x, s, r, t):
      fractional age lifetime density given special mortality law

    e_r(x, s, r, t):
      Fractional age future lifetime given special mortality law



class Uniform - Shortcuts with uniform distribution of deaths aka DeMoivre's Law

    Args:
      omega : maximum age

    Examples:
      >>> print(Uniform(95).e_x(30, t=40, curtate=False)) # 27.692


class Beta - Shortcuts with beta distribution of deaths (is Uniform when alpha = 1)

    Args:
      omega : maximum age
      alpha : alpha paramter of beta distribution
      radix : assumed starting number of lives for survival function

    Examples:
```

(continues on next page)

(continued from previous page)

```
      >>> print(Beta(omega=60, alpha=1/3).mu_x(35) * 1000)


class Gompertz - Is Makeham's Law with A = 0

    Args:
      B, c : parameters of Gompertz distribution

    Examples:
      >>> print(Gompertz(B=0.00027, c=1.1).f_x(50, t=10)) # 0.04839


class Makeham - Includes element in force of mortality that does not depend on age

    Args:
      A, B, c : parameters of Makeham distribution

    Examples:
      >>> print(Makeham(A=0.00022, B=2.7e-6, c=1.124).mu_x(60) * 0.9803)  # 0.00316
```

16.6 Examples

The `Uniform` class is initialized with the `omega` parameter to specify the range of the uniform distribution of deaths assumption. All the inherited general computation methods can be accessed; where shortcut formulas are available, then those specific methods are overriden.

```
print('Uniform')
uniform = Uniform(80).set_interest(delta=0.04)
print(uniform.whole_life_annuity(20))           # 15.53
print(uniform.temporary_annuity(20, t=5))       # 4.35
print(Uniform(161).p_x(70, t=1))                # 0.98901
print(Uniform(95).e_x(30, t=40, curtate=False)) # 27.692
print()

uniform = Uniform(omega=80).set_interest(delta=0.04)
print(uniform.E_x(20, t=5))   # .7505
print(uniform.whole_life_insurance(20, discrete=False))       # .3789
print(uniform.term_insurance(20, t=5, discrete=False))        # .0755
print(uniform.endowment_insurance(20, t=5, discrete=False))   # .8260
print(uniform.deferred_insurance(20, u=5, discrete=False))    # .3033
```

```
Uniform
16.03290804858584
4.47503070125663
0.989010989010989
32.30769230769231

0.7505031903214833
0.378867519462745
0.07552885288417432
0.8260320432056576
0.30333866657857067
```

The `Beta` class is initialized with `omega` and `alpha` parameters of the beta distribution of deaths assumption. All the inherited general computation methods can be accessed; where shortcut formulas are available, then those specific methods are overriden.

```
life = Beta(omega=100, alpha=0.5)
print(life.q_x(25, t=1, u=10))                        # 0.0072
print(life.e_x(25))                                   # 50
print(Beta(omega=60, alpha=1/3).mu_x(35) * 1000)   # 13.33
```

```
0.007188905547861446
50.0
13.333333333333332
```

The `Gompertz` or `Makeham` classes are initialized with the respective parameters of the Gompertz and Makeham Laws assumptions. All the inherited general computation methods can be accessed.

```
life = Gompertz(B=0.000005, c=1.10)
p = life.p_x(80, t=10)  # 869.4
print(life.portfolio_percentile(N=1000, mean=p, variance=p*(1-p), prob=0.99))

print(Gompertz(B=0.00027, c=1.1).f_x(50, t=10)) # 0.04839
life = Makeham(A=0.00022, B=2.7e-6, c=1.124)
print(life.mu_x(60) * 0.9803)   # 0.00316
```

```
869.3908338193208
0.048389180223511644
0.0031580641631654026
```

SOA Question 2.3:

You are given that mortality follows Gompertz Law with B = 0.00027 and c = 1.1. Calculate $f_{50}(10)$.

```
print("SOA Question 2.3: (A) 0.0483")
print(Gompertz(B=0.00027, c=1.1).f_x(x=50, t=10))
```

```
SOA Question 2.3: (A) 0.0483
0.048389180223511644
```

SOA Question 2.6

You are given the survival function:

$S_0(x) = \left(1 - \frac{x}{60}\right)^{\frac{1}{3}}, \quad 0 \leq x \leq 60$

Calculate $1000\mu_{35}$.

```
print("# SOA Question 2.6: (C) 13.3")
print(Beta(omega=60, alpha=1/3).mu_x(35) * 1000)
```

```
# SOA Question 2.6: (C) 13.3
13.333333333333332
```

CHAPTER SEVENTEEN

CONSTANT FORCE OF MORTALITY

If force of mortality is constant, then future lifetime is exponentially distributed. The exponential distribution is easy to work with, and has the *memoryless* property that survival probability is independent of age (which is clearly an unrealistic assumption for human mortality).

$${}_tp_x = e^{-\mu t}$$

$${}_tq_x = 1 - e^{-\mu t}$$

- survival functions do not depend on age x; given any desired probability of survival (or death), the time-of-death threshold can also be easily derived by inverting this simple shortcut formula

17.1 Expected future lifetime

$$\overset{\circ}{e}_x = \frac{1}{\mu}$$

- expected future lifetime is the mean of an exponential distribution, and does not depend on age x

$$\overset{\circ}{e}_{x:\overline{n|}} = \frac{1}{\mu}(1 - e^{-\mu n})$$

- temporary life expectancy is expected future life time, minus the probability of surviving n years times the expected remaining lifetime of those survivors

$$Var(T_x) = \frac{1}{\mu^2}$$

- variance of future lifetime is the variance of an exponential distribution

17.2 Pure endowment

$${}_nE_x = e^{-(\mu+\delta)n}$$

- shortcut for pure endowment after substituting in the formula for survival function

17.3 Life insurance

$$\bar{A}_x = \frac{\mu}{\mu + \delta}$$

- continuous whole life insurance, does not depend on age x

$${}_{u|}\bar{A}_x = \frac{\mu}{\mu + \delta} e^{-\mu u}$$

- deferred continuous whole life insurance as pure endowment times whole life insurance

$$\bar{A}_{x:\overline{t|}} = \frac{\mu}{\mu + \delta}(1 - e^{-\mu t})$$

- continuous term life insurance as difference of whole life and deferred whole life insurance shortcut formulas

17.4 Life annuities

$$\bar{a}_x = \frac{1}{\mu + \delta}$$

- whole life annuity, does not depend on age x

$$\bar{a}_{x:\overline{t|}} = \frac{1}{\mu + \delta}(1 - e^{-\mu t})$$

- temporary life annuity as difference of whole life and deferred whole life annuity shortcut formulas

17.5 Net future loss

$$var({}_0L) = \frac{\mu}{2\delta + \mu}$$

- variance of loss at issue on whole life insurance with benefit premium calculated using the equivalence principle equals the second moment of insurance

17.6 Methods

The `ConstantForce` class specifies a constant force of mortality for the survival model. It inherits all the general methods for computing life contingency risks, and overrides those methods with shortcut formulas that are available when assuming an exponential distribution for future lifetime.

```python
from actuarialmath import ConstantForce
import describe
describe.methods(ConstantForce)
```

```
class ConstantForce - Constant force of mortality - memoryless exponential␣
↪distribution of lifetime

    Args:
      mu : constant value of force of mortality
      udd : assume UDD (True) or CFM (False, default) between integer ages
```

(continues on next page)

(continued from previous page)

```
    Examples:
      >>> life = ConstantForce(mu=0.01).set_interest(delta=0.05)
      >>> life.term_insurance(35, t=35, discrete=False) + life.E_x(35, t=35)*0.
↪51791

    Methods:
    --------

    e_x(x, s, t, curtate, moment):
      Expected lifetime E[T_x] is memoryless: does not depend on (x)

    E_x(x, s, t, endowment, moment):
      Shortcut for pure endowment: does not depend on age x

    whole_life_insurance(x, s, moment, b, discrete):
      Shortcut for APV of whole life: does not depend on age x

    temporary_annuity(x, s, t, b, variance, discrete):
      Shortcut for temporary life annuity: does not depend on age x

    term_insurance(x, s, t, b, moment, discrete):
      Shortcut for APV of term life: does not depend on age x

    Z_t(x, prob, discrete):
      Shortcut for T_x (or K_x) given survival probability for insurance

    Y_t(x, prob, discrete):
      Shortcut for T_x (or K_x) given survival probability for annuity
```

17.7 Examples

To use the constant force of mortality assumption, the `ConstantForce` subclass can be initialized with the value of `mu`; then shortcut formulas for computing life contingencies, where available, would be called instead of general inherited methods.

```
life = ConstantForce(mu=0.01).set_interest(delta=0.05)
life.term_insurance(35, t=35, discrete=False), life.E_x(35, t=35)
```

```
(0.14625726195783623, 0.12245642825298157)
```

SOA Question 6.36

For a fully continuous 20-year term insurance policy of 100,000 on (50), you are given:

- Gross premiums, calculated using the equivalence principle, are payable at an annual rate of 4500
- Expenses at an annual rate of R are payable continuously throughout the life of the policy
- $\mu_{50+t} = 0.04$, for $t > 0$
- $\delta = 0.08$

Calculate R.

```
print("SOA Question 6.36:  (B) 500")
life = ConstantForce(mu=0.04).set_interest(delta=0.08)
a = life.temporary_annuity(50, t=20, discrete=False)
A = life.term_insurance(50, t=20, discrete=False)
def fun(R):
    return life.gross_premium(a=a, A=A, initial_premium=R/4500,
                               renewal_premium=R/4500, benefit=100000)
R = life.solve(fun, target=4500, grid=[400, 800])
print(R)
```

```
SOA Question 6.36:  (B) 500
500.0
```

SOA Question 6.31

For a fully continuous whole life insurance policy of 100,000 on (35), you are given:

- The density function of the future lifetime of a newborn: $$
- $\delta = 0.05$
- $\overline{A}_{70} = 0.51791$

Calculate the annual net premium rate for this policy.

```
print("SOA Question 6.31:  (D) 1330")
life = ConstantForce(mu=0.01).set_interest(delta=0.05)
A = life.term_insurance(35, t=35) + life.E_x(35, t=35) * 0.51791 # A_35
A = (life.term_insurance(35, t=35, discrete=False)
        + life.E_x(35, t=35) * 0.51791)     # A_35
P = life.premium_equivalence(A=A, b=100000, discrete=False)
print(P)
```

```
SOA Question 6.31:  (D) 1330
1326.5406293909457
```

SOA Question 6.27

For a special fully continuous whole life insurance on (x), you are given:

- Premiums and benefits:

	First 20 years	After 20 years
Premium Rate	3P	P
Benefit	1,000,000	500,000

- $\mu_{x+t} = 0.03, \quad t \geq 0$
- $\delta = 0.06$

Calculate P using the equivalence principle.

```
print("SOA Question 6.27:  (D) 10310")
life = ConstantForce(mu=0.03).set_interest(delta=0.06)
x = 0
```

(continues on next page)

(continued from previous page)

```
payments = (3 * life.temporary_annuity(x, t=20, discrete=False)
            + life.deferred_annuity(x, u=20, discrete=False))
benefits = (1000000 * life.term_insurance(x, t=20, discrete=False)
            + 500000 * life.deferred_insurance(x, u=20, discrete=False))
P = benefits / payments
print(P)
```

```
SOA Question 6.27:  (D) 10310
10309.617799001708
```

SOA Question 5.4

(40) wins the SOA lottery and will receive both:

- A deferred life annuity of K per year, payable continuously, starting at age $40 + \overset{\circ}{e}_{40}$ and
- An annuity certain of K per year, payable continuously, for $\overset{\circ}{e}_{40}$ years

You are given:

- $\mu = 0.02$
- $\delta = 0.01$
- The actuarial present value of the payments is 10,000

Calculate K.

```
print("SOA Question 5.4:  (A) 213.7")
life = ConstantForce(mu=0.02).set_interest(delta=0.01)
P = 10000 / life.certain_life_annuity(40, u=life.e_x(40, curtate=False),
                                        discrete=False)
print(P)
```

```
SOA Question 5.4:  (A) 213.7
213.74552118275955
```

SOA Question 5.1

You are given:

- $\delta_t = 0.06, \quad t \geq 0$
- $\mu_x(t) = 0.01, \quad t \geq 0$
- Y is the present value random variable for a continuous annuity of 1 per year, payable for the lifetime of (x) with 10 years certain

Calculate $Pr(Y > E[Y])$.

```
print("SOA Question 5.1: (A) 0.705")
life = ConstantForce(mu=0.01).set_interest(delta=0.06)
EY = life.certain_life_annuity(0, u=10, discrete=False)
print(life.p_x(0, t=life.Y_to_t(EY)))  # 0.705
```

```
SOA Question 5.1: (A) 0.705
0.7053680433746505
```

CHAPTER

EIGHTEEN

EXTRA RISK

If the underwriter determines that an individual should be offered insurance but at above standard rates, there are different ways in which we can model the extra mortality risk in a premium calculation.

18.1 Age rating

$(x) \leftarrow (x+k)$

- add years to age, referred to as age rating: the insurer may compensate for extra risk by treating the individual as being older, for example, an impaired life aged 40 might be asked to pay the same premium paid by a non- impaired life aged 45.

18.2 Multiple of mortality rate

$q_x \leftarrow q_x \cdot k$

- multiply mortality rate by a constant, which assumes that lives are subject to mortality rates that are higher than the standard lives' mortality rates.

18.3 Force of mortality

A linear transformation of μ_x affects survival probability in the following ways: Recall that μ is exponentiated to get p, hence adding a constant to μ requires multiplying ${}_tp_x$ by the constant exponentiated, while multiplying μ by a factor requires raising ${}_tp_x$ to that power

$\mu_{x+t} \leftarrow \mu_{x+t} + k \Rightarrow {}_tp_x \leftarrow {}_tp_x\, e^{-kt}$

- add constant to force of mortality, when the extra risk is largely independent of age

$\mu_{x+t} \leftarrow \mu_{x+t} \cdot k \Rightarrow {}_tp_x \leftarrow ({}_tp_x)^k$

- multiply force of mortality by constant

18.4 Methods

The `ExtraRisk` class implements methods to adjust the survival or mortality function by extra risks.

```
from actuarialmath import ExtraRisk, SelectLife, SULT
import describe
describe.methods(ExtraRisk)
```

```
class ExtraRisk - Adjust mortality by extra risk

    Args:
      life : contains original survival and mortality rates
      extra : amount of extra risk to adjust
      risk : adjust by {"ADD_FORCE", "MULTIPLY_FORCE", "ADD_AGE", "MULTIPLY_RATE"}

    Methods:
    --------

    q_x(x, s):
      Return q_[x]+s after adding age rating or multipliying mortality rate

    p_x(x, s):
      Return p_[x]+s after adding or multiplying force of mortality

    __getitem__(col):
      Returns survival function values adjusted by extra risk
```

18.5 Examples

The `ExtraRisk` class is initialized with its `life` argument set to an instance of a survival model, with amount of `extra` risk and type of `risk` (either "ADD_FORCE", "MULTIPLY_FORCE", "ADD_AGE" or "MULTIPLY_RATE") specified. The `q_x` or `p_x` methods can then be called to return the adjusted mortality rate or survival probability respectively, or an entire column of values by age can be retrieved as the item "q" or "p".

SOA Question 5.5

For an annuity-due that pays 100 at the beginning of each year that (45) is alive, you are given:

- Mortality for standard lives follows the Standard Ultimate Life Table
- The force of mortality for standard lives age 45 + t is represented as μ_{45+t}^{SULT}
- The force of mortality for substandard lives age 45 + t, μ_{45+t}^{S}, is defined as:

$$\begin{aligned}\mu_{45+t}^{S} &= \mu_{45+t}^{SULT} + 0.05, \quad 0 \le t < 1\\ &= \mu_{45+t}^{SULT}, \quad t \ge 1\end{aligned}$$

- $i = 0.05$

Calculate the actuarial present value of this annuity for a substandard life age 45.

```
print("SOA Question 5.5: (A) 1699.6")
life = SULT()
extra = ExtraRisk(life=life, extra=0.05, risk="ADD_FORCE")
select = SelectLife(periods=1).set_interest(i=.05)\
```

(continues on next page)

(continued from previous page)

```
                              .set_select(s=0, age_selected=True, q=extra['q'])\
                              .set_select(s=1, age_selected=False, a=life['a'])\
                              .fill_table()
print(100*select['a'][45][0])
```

```
SOA Question 5.5: (A) 1699.6
1699.6076593190103
```

SOA Question 4.19

(80) purchases a whole life insurance policy of 100,000. You are given:

- The policy is priced with a select period of one year
- The select mortality rate equals 80% of the mortality rate from the Standard Ultimate Life Table
- Ultimate mortality follows the Standard Ultimate Life Table
- $i = 0.05$

Calculate the actuarial present value of the death benefits for this insurance

```
print("SOA Question 4.19:  (B) 59050")
life = SULT()
extra = ExtraRisk(life=life, extra=0.8, risk="MULTIPLY_RATE")
select = SelectLife(periods=1).set_interest(i=.05)\
                              .set_select(s=0, age_selected=True, q=extra['q'])\
                              .set_select(s=1, age_selected=False, q=life['q'])\
                              .fill_table()
print(100000*select.whole_life_insurance(80, s=0))
```

```
SOA Question 4.19:  (B) 59050
59050.59973285648
```

Other examples

```
life = SULT()
extra = ExtraRisk(life=life, extra=2, risk="MULTIPLY_FORCE")
print(life.p_x(45), extra.p_x(45))
```

```
0.9992288829941123 0.9984583606096613
```

CHAPTER

NINETEEN

1/M'THLY

A 1/m-thly annuity pays $1/m$ every m-th of a year, while a 1/m-thly insurance pays \$1 of benefits at the end of the $1/m$-th year of death. A new lifetime random variable $K_x^{(m)}$ is introduced to value benefits which depend on the number of complete periods of length $1/m$ years lived by a life (x).

$K_x^{(m)} = \frac{1}{m}\lfloor mT_x \rfloor$

- 1/mthly curtate future lifetime random variable, where $m > 1$ is an integer, is the future lifetime of (x) in years rounded to the lower $\frac{1}{m}$ th of a year.

$Pr[K_x^{(m)} = k] = Pr[k \leq T_x \leq k + \frac{1}{m}] = {}_{k|\frac{1}{m}}q_x = {}_kp_x - {}_{k+\frac{1}{m}}p_x$

- the probability function for $K_x^{(m)}$ can be derived from the associated probabilities for T_x.

19.1 Life Insurance

Whole life insurance

$Z = v^{K_x^{(k)}+1/m}$

- present value random variable of 1/mthly whole life insurance

$A_x^{(m)} = E[Z] = \sum_{k=0}^{\infty} v^{\frac{k+1}{m}} {}_{\frac{k}{m}|\frac{1}{m}}q_x$

- 1/m'thly whole life insurance

$E[Z^2] = E[(v^2)^{K_x^{(k)}+1/m}] = {}^2A_x^{(m)}$

- second moment is also obtained from $A_x^{(m)}$ at double the force of interest

$Var(Z) = E[Z^2] - E[Z]^2 = {}^2A_x^{(m)} - (A_x^{(m)})^2$

- the variance of the present value of the $1/m$'thly insurance benefit can be derived by adjusting the interest rate in the first time

Term life insurance

$Z = 0$ if $K_x^{(m)} \geq t$, else $v^{K_x^{(k)}+1/m}$

- death benefit is payable at the end of the 1/m-th year of death , provided this occurs within t years.

$A^{1\,(m)}_{x:\overline{t|}} = \sum_{k=0}^{mt-1} v^{\frac{k+1}{m}} {}_{\frac{k}{m}|\frac{1}{m}}q_x$

- EPV of 1/m-thly term insurance benefits

19.2 Life Annuity

$$\ddot{a}_x^{(m)} = \sum_{k=0}^{\infty} \frac{1}{m}\, v^{\frac{k}{m}}\ {}_{\frac{k}{m}}p_x$$

- 1/mthly whole life annuity

$$\ddot{a}_{x:\overline{t|}}^{(m)} = \sum_{k=0}^{mt-1} \frac{1}{m}\, v^{\frac{k}{m}}\ {}_{\frac{k}{m}}p_x$$

- 1/mthly temporary life annuity

$$a_x^{(m)} = \ddot{a}_x^{(m)} - \frac{1}{m}$$

- immediate 1/m'thly whole life annuity

$$a_{x:\overline{t|}}^{(m)} = \ddot{a}_{x:\overline{t|}}^{(m)} - \frac{1}{m}(1 - {}_tE_x)$$

- immediate 1/m'thly temporary life annuity

19.3 Life Insurance Twin

Whole and *Temporary Life Annuities* (and *Whole Life* and *Endowment Insurance*) **ONLY**:

$$A_x^{(m)} = 1 - d^{(m)}\, \ddot{a}_x^{(m)} \iff \ddot{a}_x^{(m)} = \frac{1 - A_x^{(m)}}{d^{(m)}}$$

- 1/m'thly whole life annuity due

$$A_{x:\overline{t|}}^{(m)} = 1 - d^{(m)}\, \ddot{a}_{x:\overline{t|}}^{(m)} \iff \ddot{a}_{x:\overline{t|}}^{(m)} = \frac{1 - A_{x:\overline{t|}}^{(m)}}{d^{(m)}}$$

- 1/m'thly temporary annuity due and endowment insurance

19.4 Methods

This `Mthly` class implements methods to compute life insurance and annuity values with 1/mthly benefits.

```python
from actuarialmath import Mthly, Premiums, LifeTable
import describe
describe.methods(Mthly)
```

```
class Mthly - Compute 1/M'thly insurance and annuities

    Args:
      m : number of payments per year
      life : original survival and life contingent functions

    Methods:
    --------

    v_m(k):
      Compute discount rate compounded over k m'thly periods
```

(continues on next page)

(continued from previous page)

```
    p_m(x, s_m, t_m):
      Compute survival probability over m'thly periods

    q_m(x, s_m, t_m, u_m):
      Compute deferred mortality over m'thly periods

    Z_m(x, s, t, benefit, moment):
      Return PV of insurance r.v. Z and probability of death at mthly intervals

    E_x(x, s, t, moment, endowment):
      Compute pure endowment factor

    A_x(x, s, t, u, benefit, moment):
      Compute insurance factor with m'thly benefits

    whole_life_insurance(x, s, moment, b):
      Whole life insurance: A_x

    term_insurance(x, s, t, b, moment):
      Term life insurance: A_x:t^1

    deferred_insurance(x, s, n, b, t, moment):
      Deferred insurance n|_A_x:t^1 = discounted whole life

    endowment_insurance(x, s, t, b, endowment, moment):
      Endowment insurance: A_x:t = term insurance + pure endowment

    immediate_annuity(x, s, t, b):
      Immediate m'thly annuity

    insurance_twin(a):
      Return insurance twin of m'thly annuity

    annuity_twin(A):
      Return value of annuity twin of m'thly insurance

    annuity_variance(A2, A1, b):
      Variance of m'thly annuity from m'thly insurance moments

    whole_life_annuity(x, s, b, variance):
      Whole life m'thly annuity: a_x

    temporary_annuity(x, s, t, b, variance):
      Temporary m'thly life annuity: a_x:t

    deferred_annuity(x, s, u, t, b):
      Deferred m'thly life annuity due n|t_a_x =  n+t_a_x - n_a_x

    immediate_annuity(x, s, t, b):
      Immediate m'thly annuity
```

19.5 Examples

The `Mthly` class is initialized with its `life` argument set to an instance of a survival model with fractional age assumption, and the number of periods m in a year. The `v_m`, `p_m` and `q_m` methods can then be called to compute the discount factor, survival probability and mortality rate for m-thly ages and durations. The `Z_m` methods returns a table of the PV of the insurance random variable and deferred mortality rate by m-thly time of death. Actuarial present values, at integer ages, of insurance and annuities that pay every m-th of a year, can be computed by calling their respective methods in the usual manner.

SOA Question 6.4

For whole life annuities-due of 15 per month on each of 200 lives age 62 with independent future lifetimes, you are given:

- $i = 0.06$
- $A_{62}^{(12)} = 0.2105$ and ${}^2A_{62}^{(12)} = 0.4075$
- π is the single premium to be paid by each of the 200 lives
- S is the present value random variable at time 0 of total payments made to the 200 lives

Using the normal approximation, calculate π such that $Pr(200\pi > S) = 0.90$.

```
print("SOA Question 6.4:  (E) 1893.9")
mthly = Mthly(m=12, life=Premiums().set_interest(i=0.06))
A1, A2 = 0.4075, 0.2105
mean = mthly.annuity_twin(A1)*15*12
var = mthly.annuity_variance(A1=A1, A2=A2, b=15 * 12)
S = Premiums.portfolio_percentile(mean=mean, variance=var, prob=.9, N=200)
print(S / 200)
```

```
SOA Question 6.4:  (E) 1893.9
1893.912859650868
```

SOA Question 4.2

For a special 2-year term insurance policy on (x), you are given:

- Death benefits are payable at the end of the half-year of death
- The amount of the death benefit is 300,000 for the first half-year and increases by 30,000 per half-year thereafter
- $q_x = 0.16$ and $q_{x+1} = 0.23$
- $i^{(2)} = 0.18$
- Deaths are assumed to follow a constant force of mortality between integral ages
- Z is the present value random variable for this insurance

Calculate $Pr(Z > 277,000)$.

```
print("SOA Question 4.2:  (D) 0.18")
life = LifeTable(udd=False).set_table(q={0: 0.16, 1: 0.23})\
                                  .set_interest(i_m=0.18, m=2)
mthly = Mthly(m=2, life=life)
Z = mthly.Z_m(0, t=2, benefit=lambda x,t: 300000 + t*30000*2)
print(Z)
print(Z[Z['Z'] >= 277000]['q'].sum())
```

```
SOA Question 4.2:  (D) 0.18
               Z         q
m
1  275229.357798  0.083485
2  277754.397778  0.076515
3  277986.052822  0.102903
4  276285.832315  0.090297
0.17941813045022975
```

CHAPTER
TWENTY

UDD M’THLY

With the UDD fractional age assumption, we can work with annual insurance and annuity factors A_x and $\ddot{a}_x$, then adjust for a more appropriate frequency $A_x^{(m)}$ and $\ddot{a}_x^{(m)}$ using the following relationships.

20.1 Life insurance

Under UDD, the values of annual A_x can be used to derive exact results for 1/mthly insurance $A_x^{(m)}$.

$$A_x^{(m)} = \frac{i}{i^{(m)}} A_x$$

- discrete whole life insurance

$${}^2A_x^{(m)} = \frac{i^2 - 2i}{(i^{(m)})^2 - 2i^{(m)}}\ {}^2A_x$$

- doubling the force of interest for discrete whole life insurance

$$A^{1\ (m)}_{x:\overline{t|}} = \frac{i}{i^{(m)}} A^{1}_{x:\overline{t|}}$$

- discrete term insurance

$$A^{(m)}_{x:\overline{t|}} = \frac{i}{i^{(m)}} A^{1}_{x:\overline{t|}} +\ {}_tE_x$$

- endowment insurance combines the death and survival benefits, so we need to split off the death benefit to apply the derivations.

$${}_{u|}A_x^{(m)} =\ {}_uE_x \frac{i}{i^{(m)}} A_{x+u}$$

- discrete deferred insurance

20.2 Continuous Life Insurance

Under UDD, continuous life insurance can also be related to annual life insurance factors

$$\overline{A}_x = \frac{i}{\delta} A_x$$

- whole life insurance

$${}^2\overline{A}_x = \frac{i^2 - 2i}{2\delta}\ {}^2A_x$$

- doubling the force of interest for whole life insurance

$$\overline{A}^{1}_{x:\overline{t|}} = \frac{i}{\delta} A^{1}_{x:\overline{t|}}$$

- term life insurance

$$\overline{A}_{x:\overline{t|}} = \frac{i}{\delta} A^{1}_{x:\overline{t|}} + ~_tE_x$$

- endowment insurance

$$_{u|}\overline{A}_x = ~_uE_x~\frac{i}{\delta} A_{x+u}$$

- deferred life insurance

20.3 Interest functions

Under UDD, values of 1/mthly life annuities can be adjusted from annual life annuity factors using interest rate functions $\alpha(m)$ and $\beta(m)$. This can be shown by subtituting in annuity twins into the insurance relationships above.

$$\alpha(m) = \frac{id}{i^{(m)}~d^{(m)}}$$

$$\beta(m) = \frac{i - i^{(m)}}{i^{(m)}~d^{(m)}}$$

20.4 Life annuities

Using the values of $\ddot{a}_x$, and the interest rate functions, to obtain $\ddot{a}^{(m)}_x$ under UDD:

$$\ddot{a}^{(m)}_x = \alpha(m)~\ddot{a}_x - \beta(m)$$

- whole life annuity

$$\ddot{a}^{(m)}_{x:\overline{n|}} = \alpha(m)~\ddot{a}_{x:\overline{n|}} - \beta(m)(1 - ~_tE_x)$$

- temporary life annuity

$$_{u|}\ddot{a}^{(m)}_x = \alpha(m)~_{u|}\ddot{a}_x - \beta(m)~_uE_x$$

- deferred whole life annuity

20.5 Methods

The `UDD` class implements an instance of `Mthly`, which assumes uniform distribution of deaths (UDD) between integer ages, to compute life insurance and annuities with 1/mthly benefits with shortcut formulas.

```python
from actuarialmath import UDD, SULT, Recursion, Contract
import describe
describe.methods(UDD)
```

```
class UDD - 1/mthly shortcuts with UDD assumption

    Args:
      m : number of payments per year
```

(continues on next page)

(continued from previous page)

```
      life : original fractional survival and mortality functions

    Methods:
    --------

    alpha(m, i):
      Derive 1/mthly UDD interest rate beta function value

    beta(m, i):
      Derive 1/mthly UDD interest rate alpha function value

    interest_frame(i):
      Display 1/mthly UDD interest function values
```

20.6 Examples

SOA Question 6.38

For an n-year endowment insurance of 1000 on (x), you are given:

- Death benefits are payable at the moment of death
- Premiums are payable annually at the beginning of each year
- Deaths are uniformly distributed over each year of age
- $i = 0.05$
- ${}_nE_x = 0.172$
- $\bar{A}_{x:\overline{n|}} = 0.192$

Calculate the annual net premium for this insurance.

```python
print("SOA Question 6.38:  (B) 11.3")
x, n = 0, 10
life = Recursion().set_interest(i=0.05)\
                  .set_A(0.192, x=x, t=n, endowment=1, discrete=False)\
                  .set_E(0.172, x=x, t=n)
a = life.temporary_annuity(x, t=n, discrete=False)
print(a)
def fun(a):      # solve for discrete annuity, given continuous
    life = Recursion().set_interest(i=0.05)\
                      .set_a(a, x=x, t=n)\
                      .set_E(0.172, x=x, t=n)
    return UDD(m=0, life=life).temporary_annuity(x, t=n)
a = life.solve(fun, target=a, grid=a)  # discrete annuity
P = life.gross_premium(a=a, A=0.192, benefit=1000)
print(a, P)
```

```
SOA Question 6.38:  (B) 11.3
 *Temporary Annuity a_0(t=10) <--
   a_0(t=10) = [ 1 - A_0(t=10) ] / d(t=10)                       ~annuity twin
      a_0(t=1) = 1                                  ~one-year discrete annuity
      a_1(t=1) = 1                                  ~one-year discrete annuity
```

(continues on next page)

(continued from previous page)

```
16.560714925944584
16.978162620976775 11.308644185253657
```

SOA Question 6.32

For a whole life insurance of 100,000 on (x), you are given:

- Death benefits are payable at the moment of death
- Deaths are uniformly distributed over each year of age
- Premiums are payable monthly
- $i = 0.05$
- $\ddot{a}_x = 9.19$

Calculate the monthly net premium.

```
print("SOA Question 6.32:  (C) 550")
x = 0
life = Recursion().set_interest(i=0.05).set_a(9.19, x=x)
benefits = UDD(m=0, life=life).whole_life_insurance(x)
payments = UDD(m=12, life=life).whole_life_annuity(x)
print(benefits, payments)
print(life.gross_premium(a=payments, A=benefits, benefit=100000)/12)
```

```
SOA Question 6.32:  (C) 550
 *Whole Life Insurance A_0(t=WL) <--
   A_x = 1 - d * a_x                                             ~annuity twin
0.5763261529803323 8.72530251348809
550.4356936711871
```

SOA Question 6.22

For a whole life insurance of 100,000 on (45) with premiums payable monthly for a period of 20 years, you are given:

- The death benefit is paid immediately upon death
- Mortality follows the Standard Ultimate Life Table
- Deaths are uniformly distributed over each year of age
- $i = 0.05$

Calculate the monthly net premium.

```
print("SOA Question 6.22:  (C) 102")
life = SULT(udd=True)
a = UDD(m=12, life=life).temporary_annuity(45, t=20)
A = UDD(m=0, life=life).whole_life_insurance(45)
print(life.gross_premium(A=A, a=a, benefit=100000)/12)
```

```
SOA Question 6.22:  (C) 102
102.40668704849178
```

SOA Question 7.9

For a semi-continuous 20-year endowment insurance of 100,000 on (45), you are given:

- Net premiums of 253 are payable monthly
- Mortality follows the Standard Ultimate Life Table
- Deaths are uniformly distributed over each year of age
- $i = 0.05$

Calculate ${}_{10}V$, the net premium policy value at the end of year 10 for this insurance.

```
print("SOA Question 7.9:  (A) 38100")
sult = SULT(udd=True)
x, n, t = 45, 20, 10
a = UDD(m=12, life=sult).temporary_annuity(x+10, t=n-10)
A = UDD(m=0, life=sult).endowment_insurance(x+10, t=n-10)
print(a, A)
contract = Contract(premium=253*12, endowment=100000, benefit=100000)
print(A*100000 - a*12*253, sult.gross_future_loss(A=A, a=a, contract=contract))
```

```
SOA Question 7.9:  (A) 38100
7.831075686716718 0.6187476755196442
38099.62176709247 38099.62176709246
```

SOA Question 6.49

For a special whole life insurance of 100,000 on (40), you are given:

- The death benefit is payable at the moment of death
- Level gross premiums are payable monthly for a maximum of 20 years
- Mortality follows the Standard Ultimate Life Table
- $i = 0.05$
- Deaths are uniformly distributed over each year of age
- Initial expenses are 200
- Renewal expenses are 4% of each premium including the first
- Gross premiums are calculated using the equivalence principle

Calculate the monthly gross premium.

```
print("SOA Question 6.49:  (C) 86")
sult = SULT(udd=True)
a = UDD(m=12, life=sult).temporary_annuity(40, t=20)
A = sult.whole_life_insurance(40, discrete=False)
P = sult.gross_premium(a=a, A=A, benefit=100000, initial_policy=200,
                        renewal_premium=0.04, initial_premium=0.04)
print(P/12)
```

```
SOA Question 6.49:  (C) 86
85.99177833261696
```

Generate table of interest functions (for FAM-L exam):

```
print("Interest Functions at i=0.05")
print("----------------------------")
print(UDD.interest_frame())
```

```
Interest Functions at i=0.05
----------------------------
       i(m)     d(m)   i/i(m)   d/d(m)  alpha(m)  beta(m)
1   0.05000  0.04762  1.00000  1.00000   1.00000  0.00000
2   0.04939  0.04820  1.01235  0.98795   1.00015  0.25617
4   0.04909  0.04849  1.01856  0.98196   1.00019  0.38272
12  0.04889  0.04869  1.02271  0.97798   1.00020  0.46651
0   0.04879  0.04879  1.02480  0.97600   1.00020  0.50823
```

CHAPTER

TWENTYONE

WOOLHOUSE M’THLY

Woolhouse’s formula is a method of approximating 1/mthly life annuities from annual factors that does not depend on a fractional age assumption. It is based on the Euler-Maclaurin series expansion for the integral of a function. Life insurances may then be computed from twin relationships.

21.1 Life Annuities

$$\ddot{a}^{(m)}_x \approx \ddot{a}_x - \frac{m-1}{2m} - \frac{m^2-1}{12m^2}(\mu_x + \delta)$$

- 1/m’thly whole life annuity using the three-term Woolhouse approximation. The third term is often omitted in practice, which leads to poor approximations in some cases.

$$\ddot{a}^{(m)}_{x:\overline{t|}} \approx \ddot{a}^{(m)}_x - {}_tE_x\ \ddot{a}^{(m)}_{x+t} = \ddot{a}_{x:\overline{t|}} - \frac{m-1}{2m}(1 - {}_tE_x) - \frac{m^2-1}{12m^2}(\mu_x + \delta - {}_tE_x(\mu_{x+t} + \delta))$$

- 1/m’thly temporary life annuity from the difference of whole life Woolhouse approximations

$$\overline{a}_x \approx \ddot{a}_x - \frac{1}{2} - \frac{1}{12}(\mu_x + \delta)$$

- continuous life annuity with Woolhouse approximation when we let $m \to \infty$.

$$\mu_x \approx -\frac{1}{2}(\ln p_{x-1} + \ln p_x)$$

- if the force of mortality μ is not provided for the third Woolhouse term, it can be approximated from survival probabilities at integer ages.

21.2 Methods

Ths `Woolhouse` class implements an instance of `Mthly`, which uses the Woolhouse assumption with either two or three terms, to compute m-thly pay annuities with Woolhouse approximation formulas, from which 1/m-thly life insurance are computed from their twin formulas.

```
from actuarialmath import Woolhouse, SULT, Recursion, UDD, Contract
import describe
describe.methods(Woolhouse)
```

```
class Woolhouse - 1/m'thly shortcuts with Woolhouse approximation

    Args:
```

(continues on next page)

(continued from previous page)

```
      m : number of payments per year
      life : original fractional survival and mortality functions
      three_term : whether to include (True) or ignore (False) third term
      approximate_mu : exact (False), approximate (True) or function for third term

    Methods:
    --------

    mu_x(x, s):
      Computes mu_x or calls approximate_mu for third term
```

21.3 Examples

SOA Question 7.7

For a whole life insurance of 10,000 on (x), you are given:

- Death benefits are payable at the end of the year of death
- A premium of 30 is payable at the start of each month
- Commissions are 5% of each premium
- Expenses of 100 are payable at the start of each year
- $i = 0.05$
- $1000 A_{x+10} = 400$
- ${}_{10}V$ is the gross premium policy value at the end of year 10 for this insurance

Calculate ${}_{10}V$ using the two-term Woolhouse formula for annuities.

```
print("SOA Question 7.7:  (D) 1110")
x = 0
life = Recursion().set_interest(i=0.05).set_A(0.4, x=x+10)
a = Woolhouse(m=12, life=life).whole_life_annuity(x+10)
print(a)
contract = Contract(premium=0, benefit=10000, renewal_policy=100)
V = life.gross_future_loss(A=0.4, contract=contract.renewals())
contract = Contract(premium=30*12, renewal_premium=0.05)
V1 = life.gross_future_loss(a=a, contract=contract.renewals())
print(V, V1, V+V1)
```

```
SOA Question 7.7:  (D) 1110
```

Whole Life Annuity $\ddot{a}_{x+10}$:=

$\ddot{a}_{x+10} = [1 - A_{x+10}]/d$ insurance twin

```
12.141666666666666
5260.0 -4152.028174603174 1107.9718253968258
```

SOA Question 6.25:

For a fully discrete 10-year deferred whole life annuity-due of 1000 per month on (55), you are given:

- The premium, G, will be paid annually at the beginning of each year during the deferral period
- Expenses are expected to be 300 per year for all years, payable at the beginning of the year
- Mortality follows the Standard Ultimate Life Table
- $i = 0.05$
- Using the two-term Woolhouse approximation, the expected loss at issue is -800

Calculate G.

```
print("SOA Question 6.25:  (C) 12330")
life = SULT()
woolhouse = Woolhouse(m=12, life=life)
benefits = woolhouse.deferred_annuity(55, u=10, b=1000 * 12)
expenses = life.whole_life_annuity(55, b=300)
payments = life.temporary_annuity(55, t=10)
print(benefits + expenses, payments)
def fun(G):
    return life.gross_future_loss(A=benefits + expenses, a=payments,
                                  contract=Contract(premium=G))
G = life.solve(fun, target=-800, grid=[12110, 12550])
print(G)
```

```
SOA Question 6.25:  (C) 12330
98042.52569470297 8.019169307712845
12325.781125438532
```

SOA Question 6.15

For a fully discrete whole life insurance of 1000 on (x) with net premiums payable quarterly, you are given:

- $i = 0.05$
- $\ddot{a}_x = 3.4611$
- $P^{(W)}$ and $P^{(UDD)}$ are the annualized net premiums calculated using the 2-term Woolhouse (W) and the uniform distribution of deaths (UDD) assumptions, respectively

Calculate $\dfrac{P^{(UDD)}}{P^{(W)}}$.

```
print("SOA Question 6.15:  (B) 1.002")
x = 0
life = Recursion().set_interest(i=0.05).set_a(3.4611, x=0)
A = life.insurance_twin(3.4611)
udd = UDD(m=4, life=life)
a1 = udd.whole_life_annuity(x=x)
woolhouse = Woolhouse(m=4, life=life)
a2 = woolhouse.whole_life_annuity(x=x)
print(life.gross_premium(a=a1, A=A)/life.gross_premium(a=a2, A=A))
```

```
SOA Question 6.15:  (B) 1.002
1.0022973504113772
```

SOA Question 5.7

You are given:

- $A_{35} = 0.188$
- $A_{65} = 0.498$
- ${}_{30}p_{35} = 0.883$
- $i = 0.04$

Calculate $1000\ddot{a}^{(2)}_{35:\overline{30|}}$ using the two-term Woolhouse approximation.

```
print("SOA Question 5.7:  (C) 17376.7")
life = Recursion().set_interest(i=0.04)
life.set_A(0.188, x=35)
life.set_A(0.498, x=65)
life.set_p(0.883, x=35, t=30)
mthly = Woolhouse(m=2, life=life, three_term=False)
print(mthly.temporary_annuity(35, t=30))
print(1000 * mthly.temporary_annuity(35, t=30))
```

```
SOA Question 5.7:  (C) 17376.7
```

Whole Life Annuity $\ddot{a}_{x+35}$:=
$\ddot{a}_{x+35} = [1 - A_{x+35}]/d$ insurance twin

Whole Life Annuity $\ddot{a}_{x+65}$:=
$\ddot{a}_{x+65} = [1 - A_{x+65}]/d$ insurance twin

Pure Endowment ${}_{30}E_{x+35}$:=
${}_{30}E_{x+35} = {}_{30}p_{x+35} * v^{30}$ pure endowment

```
17.37671459632958
```

Whole Life Annuity $\ddot{a}_{x+35}$:=
$\ddot{a}_{x+35} = [1 - A_{x+35}]/d$ insurance twin

Whole Life Annuity $\ddot{a}_{x+65}$:=
$\ddot{a}_{x+65} = [1 - A_{x+65}]/d$ insurance twin

Pure Endowment ${}_{30}E_{x+35}$:=
${}_{30}E_{x+35} = {}_{30}p_{x+35} * v^{30}$ pure endowment

```
17376.71459632958
```

CHAPTER TWENTYTWO

FAM-L EXAM SOLUTIONS AND HINTS

actuarialmath – Solving Life Contingent Risks with Python

This package implements fundamental methods for modeling life contingent risks, and closely follows traditional topics covered in actuarial exams and standard texts such as the "Fundamentals of Actuarial Math - Long-term" exam syllabus by the Society of Actuaries, and "Actuarial Mathematics for Life Contingent Risks" by Dickson, Hardy and Waters. These code chunks demonstrate how to solve each of the sample FAM-L exam questions released by the SOA.

Sources:

- SOA FAM-L Exam Questions: copy retrieved Aug 2022
- SOA FAM-L Exam Solutions: copy retrieved Aug 2022
- User Guide, or download pdf
- API reference
- Github repo and issues

```
# Uncomment next line to install package
#! pip install actuarialmath
```

```
"""Solutions code and hints for SOA FAM-L sample questions

MIT License.  Copyright 2022-2023, Terence Lim
"""
import math
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from actuarialmath import Interest
from actuarialmath import Life
from actuarialmath import Survival
from actuarialmath import Lifetime
from actuarialmath import Fractional
from actuarialmath import Insurance
from actuarialmath import Annuity
from actuarialmath import Premiums
from actuarialmath import PolicyValues, Contract
from actuarialmath import Reserves
from actuarialmath import Recursion
from actuarialmath import LifeTable
from actuarialmath import SULT
from actuarialmath import SelectLife
from actuarialmath import MortalityLaws, Beta, Uniform, Makeham, Gompertz
```

(continues on next page)

(continued from previous page)

```python
from actuarialmath import ConstantForce
from actuarialmath import ExtraRisk
from actuarialmath import Mthly
from actuarialmath import UDD
from actuarialmath import Woolhouse
```

Helper to compare computed answers to expected solutions

```python
import time
class IsClose:
    """Helper class for testing and reporting if two values are close"""
    def __init__(self, rel_tol : float = 0.01, score : bool = False,
                 verbose: bool = False):
        self.den = self.num = 0
        self.score = score      # whether to count INCORRECTs instead of assert
        self.verbose = verbose  # whether to run silently
        self.incorrect = []     # to keep list of messages for INCORRECT
        self.tol = rel_tol
        self.tic = time.time()

    def __call__(self, solution, answer, question="", rel_tol=None):
        """Compare solution to answer within relative tolerance

        Args:
          solution (str | numeric) : gold label
          answer (str | numeric) : computed answer
          question (str) : label to associate with this test
          rel_tol (float) : relative tolerance to be considered close
        """
        if isinstance(solution, str):
            isclose = (solution == answer)
        else:
            isclose = math.isclose(solution, answer, rel_tol=rel_tol or self.tol)
        self.den += 1
        self.num += isclose
        msg = f"{question} {solution}: {answer}"
        if self.verbose:
            print("-----", msg, "[OK]" if isclose else "[INCORRECT]", "-----")
        if not self.score:
            assert isclose, msg
        if not isclose:
            self.incorrect.append(msg)
        return isclose

    def __str__(self):
        """Display cumulative score and errors"""
        return f"Elapsed: {time.time()-self.tic:.1f} secs\n" \
               + f"Passed:  {self.num}/{self.den}\n" + "\n".join(self.incorrect)
isclose = IsClose(0.01, score=False, verbose=True)
```

22.1 1 Tables

These tables are provided in the FAM-L exam

- Interest Functions at i=0.05
- Normal Distribution Table
- Standard Ultimate Life Table

but you actually do not need them here!

```
print("Interest Functions at i=0.05")
UDD.interest_frame()
```

```
Interest Functions at i=0.05
```

```
       i(m)     d(m)    i/i(m)   d/d(m)   alpha(m)   beta(m)
1   0.05000  0.04762  1.00000  1.00000   1.00000  0.00000
2   0.04939  0.04820  1.01235  0.98795   1.00015  0.25617
4   0.04909  0.04849  1.01856  0.98196   1.00019  0.38272
12  0.04889  0.04869  1.02271  0.97798   1.00020  0.46651
0   0.04879  0.04879  1.02480  0.97600   1.00020  0.50823
```

```
print("Values of z for selected values of Pr(Z<=z)")
print(Life.quantiles_frame().to_string(float_format=lambda x: f"{x:.3f}"))
```

```
Values of z for selected values of Pr(Z<=z)
z         0.842  1.036  1.282  1.645  1.960  2.326  2.576
Pr(Z<=z)  0.800  0.850  0.900  0.950  0.975  0.990  0.995
```

```
print("Standard Ultimate Life Table at i=0.05")
SULT().frame()
```

```
Standard Ultimate Life Table at i=0.05
```

```
          l_x       q_x      a_x      A_x     2A_x   a_x:10   A_x:10   a_x:20
20   100000.0  0.000250  19.9664  0.04922  0.00580  8.0991  0.61433  13.0559  \
21    99975.0  0.000253  19.9197  0.05144  0.00614  8.0990  0.61433  13.0551
22    99949.7  0.000257  19.8707  0.05378  0.00652  8.0988  0.61434  13.0541
23    99924.0  0.000262  19.8193  0.05622  0.00694  8.0986  0.61435  13.0531
24    99897.8  0.000267  19.7655  0.05879  0.00739  8.0983  0.61437  13.0519
..        ...       ...      ...      ...      ...     ...      ...      ...
96    17501.8  0.192887   3.5597  0.83049  0.69991  3.5356  0.83164   3.5597
97    14125.9  0.214030   3.3300  0.84143  0.71708  3.3159  0.84210   3.3300
98    11102.5  0.237134   3.1127  0.85177  0.73356  3.1050  0.85214   3.1127
99     8469.7  0.262294   2.9079  0.86153  0.74930  2.9039  0.86172   2.9079
100    6248.2  0.289584   2.7156  0.87068  0.76427  2.7137  0.87078   2.7156

      A_x:20    5_E_x   10_E_x   20_E_x
20   0.37829  0.78252  0.61224  0.37440
21   0.37833  0.78250  0.61220  0.37429
22   0.37837  0.78248  0.61215  0.37417
```

(continues on next page)

(continued from previous page)

```
23   0.37842  0.78245  0.61210  0.37404
24   0.37848  0.78243  0.61205  0.37390
..       ...      ...      ...      ...
96   0.83049  0.19872  0.01330  0.00000
97   0.84143  0.16765  0.00827  0.00000
98   0.85177  0.13850  0.00485  0.00000
99   0.86153  0.11173  0.00266  0.00000
100  0.87068  0.08777  0.00136  0.00000

[81 rows x 12 columns]
```

22.2 2 Survival models

SOA Question 2.1 : (B) 2.5

You are given:

1. $S_0(t) = (1 - \frac{t}{\omega})^{\frac{1}{4}}, \quad 0 \leq t \leq \omega$
2. $\mu_{65} = \frac{1}{180}$

Calculate e_{106}, the curtate expectation of life at age 106.

hints:

- derive formula for μ from given survival function
- solve for ω given μ_{65}
- calculate e by summing survival probabilities

```
life = Lifetime()
def mu_from_l(omega):   # first solve for omega, given mu_65 = 1/180
    return life.set_survival(l=lambda x,s: (1 - (x+s)/omega)**0.25).mu_x(65)
omega = int(life.solve(mu_from_l, target=1/180, grid=100))
e = life.set_survival(l=lambda x,s:(1 - (x + s)/omega)**.25, maxage=omega)\
        .e_x(106)       # then solve expected lifetime from omega
isclose(2.5, e, question="Q2.1")
```

```
----- Q2.1 2.5: 2.4786080555423604 [OK] -----
```

```
True
```

SOA Question 2.2 : (D) 400

Scientists are searching for a vaccine for a disease. You are given:

1. 100,000 lives age x are exposed to the disease
2. Future lifetimes are independent, except that the vaccine, if available, will be given to all at the end of year 1
3. The probability that the vaccine will be available is 0.2
4. For each life during year 1, q_x = 0.02
5. For each life during year 2, q_{x+1} = 0.01 if the vaccine has been given and q_{x+1} = 0.02 if it has not been given

Calculate the standard deviation of the number of survivors at the end of year 2.

hints:

- calculate survival probabilities for the two scenarios
- apply conditional variance formula (or mixed distribution)

```
p1 = (1. - 0.02) * (1. - 0.01)  # 2_p_x if vaccine given
p2 = (1. - 0.02) * (1. - 0.02)  # 2_p_x if vaccine not given
std = math.sqrt(Life.conditional_variance(p=.2, p1=p1, p2=p2, N=100000))
isclose(400, std, question="Q2.2")
```

```
----- Q2.2 400: 396.5914603215815 [OK] -----
```

```
True
```

SOA Question 2.3 : (A) 0.0483

You are given that mortality follows Gompertz Law with B = 0.00027 and c = 1.1.

Calculate $f_{50}(10)$.

hints:

- Derive formula for f given survival function

```
B, c = 0.00027, 1.1
S = lambda x,s,t: math.exp(-B * c**(x+s) * (c**t - 1)/math.log(c))
life = Survival().set_survival(S=S)
f = life.f_x(x=50, t=10)
isclose(0.0483, f, question="Q2.3")
```

```
----- Q2.3 0.0483: 0.048327399045049846 [OK] -----
```

```
True
```

SOA Question 2.4 : (E) 8.2

You are given $_tq_0 = \frac{t^2}{10{,}000} \quad 0 < t < 100$. Calculate $\mathring{e}_{75:\overline{10|}}$.

hints:

- derive survival probability function $_tp_x$ given $_tq_0$
- compute $\mathring{e}$ by integration

```
def q(t) : return (t**2)/10000. if t < 100 else 1.
e = Lifetime().set_survival(l=lambda x,s: 1 - q(x+s)).e_x(75, t=10, curtate=False)
isclose(8.2, e, question="Q2.4")
```

```
----- Q2.4 8.2: 8.20952380952381 [OK] -----
```

```
True
```

SOA Question 2.5 : (B) 37.1

You are given the following:

1. $e_{40:20} = 18$
2. $e_{60} = 25$
3. ${}_{20}q_{40} = 0.2$
4. $q_{40} = 0.003$

Calculate e_{41}.

hints:

- solve for e_{40} from limited lifetime formula
- compute e_{41} using forward recursion

```
life = Recursion(verbose=True).set_e(25, x=60, curtate=True)\
                              .set_q(0.2, x=40, t=20)\
                              .set_q(0.003, x=40)\
                              .set_e(18, x=40, t=20, curtate=True)
e = life.e_x(41, curtate=True)
isclose(37.1, e, question="Q2.5")
```

$$
\begin{array}{rl}
\texttt{Lifetime}\ e_{x+41} := & \\
e_{x+41} = [\, e_{x+40} - e_{x+40:\overline{1}|}]/\, p_{x+40} & \texttt{forward recursion} \\
e_{x+40} = e_{x+40:\overline{20}|} + {}_{20}p_{x+40} * e_{x+60} & \texttt{backward recursion} \\
{}_{20}p_{x+40} = 1 - {}_{20}q_{x+40} & \texttt{complement of mortality} \\
e_{x+40:\overline{1}|} = p_{x+40} & \texttt{1-year curtate shortcut} \\
p_{x+40} = 1 - q_{x+40} & \texttt{complement of mortality}
\end{array}
$$

```
----- Q2.5 37.1: 37.11434302908726 [OK] -----
```

```
True
```

SOA Question 2.6 : (C) 13.3

You are given the survival function:

$S_0(x) = \left(1 - \frac{x}{60}\right)^{\frac{1}{3}}, \quad 0 \le x \le 60$

Calculate $1000\mu_{35}$.

hints:

- derive force of mortality function μ from given survival function

```
life = Survival().set_survival(l=lambda x,s: (1 - (x+s)/60)**(1/3))
mu = 1000 * life.mu_x(35)
isclose(13.3, mu, question="Q2.6")
```

```
----- Q2.6 13.3: 13.337287043994648 [OK] -----
```

```
True
```

SOA Question 2.7 : (B) 0.1477

You are given the following survival function of a newborn:

$$\begin{aligned} S_0(x) &= 1 - \frac{x}{250}, \quad 0 \le x < 40 \\ &= 1 - \left(\frac{x}{100}\right)^2, \quad 40 \le x \le 100 \end{aligned}$$

Calculate the probability that (30) dies within the next 20 years.

hints:

- calculate from given survival function

```
l = lambda x,s: (1-((x+s)/250) if (x+s)<40 else 1-((x+s)/100)**2)
q = Survival().set_survival(l=l).q_x(30, t=20)
isclose(0.1477, q, question="Q2.7")
```

```
----- Q2.7 0.1477: 0.1477272727272727 [OK] -----
```

```
True
```

SOA Question 2.8 : (C) 0.94

In a population initially consisting of 75% females and 25% males, you are given:

1. For a female, the force of mortality is constant and equals μ
2. For a male, the force of mortality is constant and equals 1.5 μ
3. At the end of 20 years, the population is expected to consist of 85% females and 15% males

Calculate the probability that a female survives one year.

hints:

- relate p_{male} and p_{female} through the common term μ and the given proportions

```
def fun(mu):  # Solve first for mu, given ratio of start and end proportions
    male = Survival().set_survival(mu=lambda x,s: 1.5 * mu)
    female = Survival().set_survival(mu=lambda x,s: mu)
    return (75 * female.p_x(0, t=20)) / (25 * male.p_x(0, t=20))
mu = Survival.solve(fun, target=85/15, grid=0.5)
p = Survival().set_survival(mu=lambda x,s: mu).p_x(0, t=1)
isclose(0.94, p, question="Q2.8")
```

```
----- Q2.8 0.94: 0.9383813306903799 [OK] -----
```

```
True
```

22.3 3 Life tables and selection

SOA Question 3.1 : (B) 117

You are given:

1. An excerpt from a select and ultimate life table with a select period of 3 years:

x	$\ell_{[x]}$	$\ell_{[x]+1}$	$\ell_{[x]+2}$	ℓ_{x+3}	$x+3$
60	80,000	79,000	77,000	74,000	63
61	78,000	76,000	73,000	70,000	64
62	75,000	72,000	69,000	67,000	65
63	71,000	68,000	66,000	65,000	66

2. Deaths follow a constant force of mortality over each year of age

Calculate 1000 ${}_{23}q_{[60]+0.75}$.

hints:

- interpolate with constant force of maturity

```
life = SelectLife().set_table(l={60: [80000, 79000, 77000, 74000],
                                 61: [78000, 76000, 73000, 70000],
                                 62: [75000, 72000, 69000, 67000],
                                 63: [71000, 68000, 66000, 65000]})
q = 1000 * life.q_r(60, s=0, r=0.75, t=3, u=2)
isclose(117, q, question="Q3.1")
```

```
----- Q3.1 117: 116.7192429022082 [OK] -----
```

```
True
```

SOA Question 3.2 : (D) 14.7

You are given:

1. The following extract from a mortality table with a one-year select period:

x	$l_{[x]}$	$d_{[x]}$	l_{x+1}	$x+1$
65	1000	40	–	66
66	955	45	–	67

2. Deaths are uniformly distributed over each year of age

$\mathring{e}_{[65]} = 15.0$

Calculate $\mathring{e}_{[66]}$.

hints:

- UDD $\Rightarrow \mathring{e}_x = e_x + 0.5$
- fill select table using curtate expectations

```
e_curtate = Fractional.e_approximate(e_complete=15)
life = SelectLife(udd=True).set_table(l={65: [1000, None,],
                                         66: [955, None]},
                                      e={65: [e_curtate, None]},
                                      d={65: [40, None,],
                                         66: [45, None]})
e = life.e_r(x=66)
isclose(14.7, e, question="Q3.2")
```

```
----- Q3.2 14.7: 14.67801047120419 [OK] -----
```

```
True
```

SOA Question 3.3 : (E) 1074

You are given:

1. An excerpt from a select and ultimate life table with a select period of 2 years:

x	$\ell_{[x]}$	$\ell_{[x]+1}$	ℓ_{x+2}	$x+2$
50	99,000	96,000	93,000	52
51	97,000	93,000	89,000	53
52	93,000	88,000	83,000	54
53	90,000	84,000	78,000	55

2. Deaths are uniformly distributed over each year of age

Calculate $10,000\ {}_{2.2}q_{[51]+0.5}$.

hints:

- interpolate lives between integer ages with UDD

```
life = SelectLife().set_table(l={50: [99, 96, 93],
                                 51: [97, 93, 89],
                                 52: [93, 88, 83],
                                 53: [90, 84, 78]})
q = 10000 * life.q_r(51, s=0, r=0.5, t=2.2)
isclose(1074, q, question="Q3.3")
```

```
----- Q3.3 1074: 1073.684210526316 [OK] -----
```

```
True
```

SOA Question 3.4 : (B) 815

The SULT Club has 4000 members all age 25 with independent future lifetimes. The mortality for each member follows the Standard Ultimate Life Table.

Calculate the largest integer N, using the normal approximation, such that the probability that there are at least N survivors at age 95 is at least 90%.

hints:

- compute portfolio percentile with N=4000, and mean and variance from binomial distribution

```
sult = SULT()
mean = sult.p_x(25, t=95-25)
var = sult.bernoulli(mean, variance=True)
pct = sult.portfolio_percentile(N=4000, mean=mean, variance=var, prob=0.1)
isclose(815, pct, question="Q3.4")
```

```
----- Q3.4 815: 815.0943255167722 [OK] -----
```

```
True
```

SOA Question 3.5 : (E) 106

You are given:

x	60	61	62	63	64	65	66	67
l_x	99,999	88,888	77,777	66,666	55,555	44,444	33,333	22,222

$a = {}_{3.4|2.5}q_{60}$ assuming a uniform distribution of deaths over each year of age

$b = {}_{3.4|2.5}q_{60}$ assuming a constant force of mortality over each year of age

Calculate $100,000(a - b)$

hints:

- compute mortality rates by interpolating lives between integer ages, with UDD and constant force of mortality assumptions

```
l = [99999, 88888, 77777, 66666, 55555, 44444, 33333, 22222]
a = LifeTable(udd=True).set_table(l={age:l for age,l in zip(range(60, 68), l)})\
                       .q_r(60, u=3.4, t=2.5)
b = LifeTable(udd=False).set_table(l={age:l for age,l in zip(range(60, 68), l)})\
                        .q_r(60, u=3.4, t=2.5)
isclose(106, 100000 * (a - b), question="Q3.5")
```

```
----- Q3.5 106: 106.16575827938624 [OK] -----
```

```
True
```

SOA Question 3.6 : (D) 15.85

You are given the following extract from a table with a 3-year select period:

x	$q_{[x]}$	$q_{[x]+1}$	$q_{[x]+2}$	q_{x+3}	$x+3$
60	0.09	0.11	0.13	0.15	63
61	0.10	0.12	0.14	0.16	64
62	0.11	0.13	0.15	0.17	65
63	0.12	0.14	0.16	0.18	66
64	0.13	0.15	0.17	0.19	67

$e_{64} = 5.10$

Calculate $e_{[61]}$.

hints:

- apply recursion formulas for curtate expectation

```
e = SelectLife().set_table(q={60: [.09, .11, .13, .15],
                              61: [.1, .12, .14, .16],
                              62: [.11, .13, .15, .17],
                              63: [.12, .14, .16, .18],
                              64: [.13, .15, .17, .19]},
                           e={61: [None, None, None, 5.1]})\
                .e_x(61)
isclose(5.85, e, question="Q3.6")
```

```
----- Q3.6 5.85: 5.846832 [OK] -----
```

```
True
```

SOA Question 3.7 : (b) 16.4

For a mortality table with a select period of two years, you are given:

x	$q_{[x]}$	$q_{[x]+1}$	q_{x+2}	$x+2$
50	0.0050	0.0063	0.0080	52
51	0.0060	0.0073	0.0090	53
52	0.0070	0.0083	0.0100	54
53	0.0080	0.0093	0.0110	55

The force of mortality is constant between integral ages.

Calculate $1000\ {}_{2.5}q_{[50]+0.4}$.

hints:

- use deferred mortality formula
- use chain rule for survival probabilities,
- interpolate between integer ages with constant force of mortality

```
life = SelectLife().set_table(q={50: [.0050, .0063, .0080],
                                 51: [.0060, .0073, .0090],
                                 52: [.0070, .0083, .0100],
                                 53: [.0080, .0093, .0110]})
q = 1000 * life.q_r(50, s=0, r=0.4, t=2.5)
isclose(16.4, q, question="Q3.7")
```

```
----- Q3.7 16.4: 16.420207214428586 [OK] -----
```

```
True
```

SOA Question 3.8 : (B) 1505

A club is established with 2000 members, 1000 of exact age 35 and 1000 of exact age 45. You are given:

1. Mortality follows the Standard Ultimate Life Table
2. Future lifetimes are independent
3. N is the random variable for the number of members still alive 40 years after the club is established

Using the normal approximation, without the continuity correction, calculate the smallest n such that $Pr(N \geq n) \leq 0.05$.

hints:

- compute portfolio means and variances from sum of 2000 independent members' means and variances of survival.

```
sult  = SULT()
p1  = sult.p_x(35, t=40)
p2  = sult.p_x(45, t=40)
mean  = sult.bernoulli(p1) * 1000 + sult.bernoulli(p2) * 1000
var  = (sult.bernoulli(p1, variance=True) * 1000
        + sult.bernoulli(p2, variance=True) * 1000)
pct  = sult.portfolio_percentile(mean=mean, variance=var, prob=.95)
isclose(1505, pct, question="Q3.8")
```

```
----- Q3.8 1505: 1504.8328375406456 [OK] -----
```

```
True
```

SOA Question 3.9 : (E) 3850

A father-son club has 4000 members, 2000 of which are age 20 and the other 2000 are age 45. In 25 years, the members of the club intend to hold a reunion.

You are given:

1. All lives have independent future lifetimes.
2. Mortality follows the Standard Ultimate Life Table.

Using the normal approximation, without the continuity correction, calculate the 99th percentile of the number of surviving members at the time of the reunion.

hints:

- compute portfolio means and variances as sum of 4000 independent members' means and variances (of survival)
- retrieve normal percentile

```
sult  = SULT()
p1  = sult.p_x(20, t=25)
p2  = sult.p_x(45, t=25)
mean  = sult.bernoulli(p1) * 2000 + sult.bernoulli(p2) * 2000
var  = (sult.bernoulli(p1, variance=True) * 2000
        + sult.bernoulli(p2, variance=True) * 2000)
pct  = sult.portfolio_percentile(mean=mean, variance=var, prob=.99)
isclose(3850, pct, question="Q3.9")
```

```
----- Q3.9 3850: 3850.144345130047 [OK] -----
```

```
True
```

SOA Question 3.10 : (C) 0.86

A group of 100 people start a Scissor Usage Support Group. The rate at which members enter and leave the group is dependent on whether they are right-handed or left-handed.

You are given the following:

1. The initial membership is made up of 75% left-handed members (L) and 25% right-handed members (R)
2. After the group initially forms, 35 new (L) and 15 new (R) join the group at the start of each subsequent year
3. Members leave the group only at the end of each year
4. q_L = 0.25 for all years
5. q_R = 0.50 for all years

Calculate the proportion of the Scissor Usage Support Group's expected membership that is left-handed at the start of the group's 6th year, before any new members join for that year.

hints:

- reformulate the problem by reversing time: survival to year 6 is calculated in reverse as discounting by the same number of years.

```
interest = Interest(v=0.75)
L = 35*interest.annuity(t=4, due=False) + 75*interest.v_t(t=5)
interest = Interest(v=0.5)
R = 15*interest.annuity(t=4, due=False) + 25*interest.v_t(t=5)
isclose(0.86, L / (L + R), question="Q3.10")
```

```
----- Q3.10 0.86: 0.8578442833761983 [OK] -----
```

```
True
```

SOA Question 3.11 : (B) 0.03

For the country of Bienna, you are given:

1. Bienna publishes mortality rates in biennial form, that is, mortality rates are of the form: ${}_2q_{2x}$, for $x = 0, 1, 2, ...$
2. Deaths are assumed to be uniformly distributed between ages $2x$ and $2x + 2$, for $x = 0, 1, 2, ...$
3. ${}_2q_{50} = 0.02$
4. ${}_2q_{52} = 0.04$

Calculate the probability that (50) dies during the next 2.5 years.

hints:

- calculate mortality rate by interpolating lives assuming UDD

```
life = LifeTable(udd=True).set_table(q={50//2: .02, 52//2: .04})
q = life.q_r(50//2, t=2.5/2)
isclose(0.03, q, question="Q3.11")
```

```
----- Q3.11 0.03: 0.0298 [OK] -----
```

```
True
```

SOA Question 3.12 : (C) 0.055

X and Y are both age 61. X has just purchased a whole life insurance policy. Y purchased a whole life insurance policy one year ago.

Both X and Y are subject to the following 3-year select and ultimate table:

x	$\ell_{[x]}$	$\ell_{[x]+1}$	$\ell_{[x]+2}$	ℓ_{x+3}	$x+3$
60	10,000	9,600	8,640	7,771	63
61	8,654	8,135	6,996	5,737	64
62	7,119	6,549	5,501	4,016	65
63	5,760	4,954	3,765	2,410	66

The force of mortality is constant over each year of age.

Calculate the difference in the probability of survival to age 64.5 between X and Y.

hints:

- compute survival probability by interpolating lives assuming constant force

```
life = SelectLife(udd=False).set_table(l={60: [10000, 9600, 8640, 7771],
                                          61: [8654, 8135, 6996, 5737],
                                          62: [7119, 6549, 5501, 4016],
                                          63: [5760, 4954, 3765, 2410]})
q = life.q_r(60, s=1, t=3.5) - life.q_r(61, s=0, t=3.5)
isclose(0.055, q, question="Q3.12")
```

```
----- Q3.12 0.055: 0.05465655938591829 [OK] -----
```

```
True
```

SOA Question 3.13 : (B) 1.6

A life is subject to the following 3-year select and ultimate table:

$[x]$	$\ell_{[x]}$	$\ell_{[x]+1}$	$\ell_{[x]+2}$	ℓ_{x+3}	$x+3$
55	10,000	9,493	8,533	7,664	58
56	8,547	8,028	6,889	5,630	59
57	7,011	6,443	5,395	3,904	60
58	5,853	4,846	3,548	2,210	61

You are also given:

1. $e_{60} = 1$
2. Deaths are uniformly distributed over each year of age

Calculate $\mathring{e}_{[58]+2}$.

hints:

- compute curtate expectations using recursion formulas
- convert to complete expectation assuming UDD

```
life = SelectLife().set_table(l={55: [10000, 9493, 8533, 7664],
                                 56: [8547, 8028, 6889, 5630],
                                 57: [7011, 6443, 5395, 3904],
                                 58: [5853, 4846, 3548, 2210]},
                              e={57: [None, None, None, 1]})
e = life.e_r(x=58, s=2)
isclose(1.6, e, question="Q3.13")
```

```
----- Q3.13 1.6: 1.6003382187147688 [OK] -----
```

```
True
```

SOA Question 3.14 : (C) 0.345

You are given the following information from a life table:

x	l_x	d_x	p_x	q_x
95	–	–	–	0.40
96	–	–	0.20	–
97	–	72	–	1.00

You are also given:

1. $l_{90} = 1000$ and $l_{93} = 825$
2. Deaths are uniformly distributed over each year of age.

Calculate the probability that (90) dies between ages 93 and 95.5.

hints:

- compute mortality by interpolating lives between integer ages assuming UDD

```
life = LifeTable(udd=True).set_table(l={90: 1000, 93: 825},
                                     d={97: 72},
                                     p={96: .2},
                                     q={95: .4, 97: 1})
q = life.q_r(90, u=93-90, t=95.5 - 93)
isclose(0.345, q, question="Q3.14")
```

```
----- Q3.14 0.345: 0.345 [OK] -----
```

```
True
```

22.4 4 Insurance benefits

SOA Question 4.1 : (A) 0.27212

For a special whole life insurance policy issued on (40), you are given:

1. Death benefits are payable at the end of the year of death
2. The amount of benefit is 2 if death occurs within the first 20 years and is 1 thereafter
3. Z is the present value random variable for the payments under this insurance
4. $i = 0.03$
5.

x	A_x	${}_{20}E_x$
40	0.36987	0.51276
60	0.62567	0.17878

6. $E[Z^2] = 0.24954$

Calculate the standard deviation of Z.

hints:

- solve EPV as sum of term and deferred insurance
- compute variance as difference of second moment and first moment squared

```
life = Recursion().set_interest(i=0.03)
life.set_A(0.36987, x=40).set_A(0.62567, x=60)
life.set_E(0.51276, x=40, t=20).set_E(0.17878, x=60, t=20)
Z2 = 0.24954
A = (2 * life.term_insurance(40, t=20) + life.deferred_insurance(40, u=20))
std = math.sqrt(life.insurance_variance(A2=Z2, A1=A))
isclose(0.27212, std, question="Q4.1")
```

```
----- Q4.1 0.27212: 0.2721117749374753 [OK] -----
```

```
True
```

SOA Question 4.2 : (D) 0.18

or a special 2-year term insurance policy on (x), you are given:

1. Death benefits are payable at the end of the half-year of death
2. The amount of the death benefit is 300,000 for the first half-year and increases by 30,000 per half-year thereafter
3. $q_x = 0.16$ and $q_{x+1} = 0.23$
4. $i^{(2)} = 0.18$
5. Deaths are assumed to follow a constant force of mortality between integral ages
6. Z is the present value random variable for this insurance

Calculate Pr($Z > 277{,}000$) .

hints:

- calculate Z(t) and deferred mortality for each half-yearly t
- sum the deferred mortality probabilities for periods when PV > 277000

```
life = LifeTable(udd=False).set_table(q={0: .16, 1: .23})\
                           .set_interest(i_m=.18, m=2)
mthly = Mthly(m=2, life=life)
Z = mthly.Z_m(0, t=2, benefit=lambda x,t: 300000 + t*30000*2)
p = Z[Z['Z'] >= 277000]['q'].sum()
isclose(0.18, p, question="Q4.2")
```

```
----- Q4.2 0.18: 0.17941813045022975 [OK] -----
```

```
True
```

SOA Question 4.3 : (D) 0.878

You are given:

1. $q_{60} = 0.01$
2. Using $i = 0.05,\ A_{60:\overline{3}|} = 0.86545$
3. Using $i = 0.045$ calculate $A_{60:\overline{3}|}$

hints:

- solve q_{61} from endowment insurance EPV formula
- solve $A_{60:\overline{3}|}$ with new $i = 0.045$ as EPV of endowment insurance benefits.

```
life = Recursion(verbose=True).set_interest(i=0.05)\
                              .set_q(0.01, x=60)\
                              .set_A(0.86545, x=60, t=3, endowment=1)
q = life.q_x(x=61)
A = Recursion(verbose=True).set_interest(i=0.045)\
                           .set_q(0.01, x=60)\
                           .set_q(q, x=61)\
                           .endowment_insurance(60, t=3)
isclose(0.878, A, question="Q4.3")
```

$$
\begin{aligned}
&\texttt{Mortality}\ q_{x+61}\ := \\
&q_{x+61} = 1 - p_{x+61} && \texttt{complement survival} \\
&p_{x+61} = [v - A_{x+61:\overline{2}|}]/[v * [1 - A_{x+62:\overline{1}|}]] && \texttt{insurance recursion} \\
&A^{1}_{x+61:\overline{2}|} = [\, A^{1}_{x+60:\overline{3}|}/v - q_{x+60} * b]/\ p_{x+60} && \texttt{forward recursion} \\
&p_{x+60} = 1 - q_{x+60} && \texttt{complement of mortality}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{Endowment Insurance}\ A_{x+60:\overline{3}|}\ := \\
&A^{1}_{x+60:\overline{3}|} = v * [\, q_{x+60} * b + p_{x+60} * A^{1}_{x+61:\overline{2}|}] && \texttt{backward recursion} \\
&A^{1}_{x+61:\overline{2}|} = v * [\, q_{x+61} * b + p_{x+61} * A^{1}_{x+62:\overline{1}|}] && \texttt{backward recursion} \\
&p_{x+61} = 1 - q_{x+61} && \texttt{complement of mortality} \\
&E_{x+60} = p_{x+60} * v && \texttt{pure endowment} \\
&p_{x+60} = 1 - q_{x+60} && \texttt{complement of mortality}
\end{aligned}
$$

```
----- Q4.3 0.878: 0.8777667236003878 [OK] -----
```

```
True
```

SOA Question 4.4 : (A) 0.036

For a special increasing whole life insurance on (40), payable at the moment of death, you are given :

1. The death benefit at time t is $b_t = 1 + 0.2t, \quad t \geq 0$
2. The interest discount factor at time t is $v(t) = (1 + 0.2t) - 2, \quad t \geq 0$
3. ${}_tp_{40}\ \mu_{40+t} = 0.025$ if $0 \leq t < 40$, otherwise 0
4. Z is the present value random variable for this insurance

Calculate Var(Z).

hints:

- integrate to find EPV of Z and Z^2
- variance is difference of second moment and first moment squared

```
x = 40
life = Insurance().set_survival(f=lambda *x: 0.025, maxage=x+40)\
                  .set_interest(v_t=lambda t: (1 + .2*t)**(-2))
def benefit(x,t): return 1 + .2 * t
A1 = life.A_x(x, benefit=benefit, discrete=False)
A2 = life.A_x(x, moment=2, benefit=benefit, discrete=False)
var = A2 - A1**2
isclose(0.036, var, question="Q4.4")
```

```
----- Q4.4 0.036: 0.03567680106032681 [OK] -----
```

```
True
```

SOA Question 4.5 : (C) 35200

For a 30-year term life insurance of 100,000 on (45), you are given:

1. The death benefit is payable at the moment of death
2. Mortality follows the Standard Ultimate Life Table
3. $\delta = 0.05$
4. Deaths are uniformly distributed over each year of age

Calculate the 95th percentile of the present value of benefits random variable for this insurance

hints:

- interpolate between integer ages with UDD, and find lifetime that mortality rate exceeded
- compute PV of death benefit paid at that time.

```
sult = SULT(udd=True).set_interest(delta=0.05)
Z = 100000 * sult.Z_from_prob(45, prob=0.95, discrete=False)
isclose(35200, Z, question="Q4.5")
```

```
----- Q4.5 35200: 35187.95203719653 [OK] -----
```

```
True
```

SOA Question 4.6 : (B) 29.85

For a 3-year term insurance of 1000 on (70), you are given:

1. q_{70+k}^{SULT} is the mortality rate from the Standard Ultimate Life Table, for k = 0,1,2
2. q_{70+k} is the mortality rate used to price this insurance, for k = 0,1, 2
3. $q_{70+k} = (0.95)^k q_{70+k}^{SULT}$, for k = 0,1, 2
4. *i* = 0.05

Calculate the single net premium.

hints:

- calculate adjusted mortality rates
- compute term insurance as EPV of benefits

```
sult = SULT()
life = LifeTable().set_interest(i=0.05)\
                  .set_table(q={70+k: .95**k * sult.q_x(70+k) for k in range(3)})
A = life.term_insurance(70, t=3, b=1000)
isclose(29.85, A, question="Q4.6")
```

```
----- Q4.6 29.85: 29.84835110355902 [OK] -----
```

```
True
```

SOA Question 4.7 : (B) 0.06

For a 25-year pure endowment of 1 on (*x*), you are given:

1. *Z* is the present value random variable at issue of the benefit payment
2. *Var (Z)* = 0.10 *E[Z]*
3. $_{25}p_x = 0.57$

Calculate the annual effective interest rate.

hints:

- use Bernoulli shortcut formula for variance of pure endowment Z
- solve for i, since p is given.

```
def fun(i):
    life = Recursion(verbose=False).set_interest(i=i)\
                                   .set_p(0.57, x=0, t=25)
    return 0.1*life.E_x(0, t=25) - life.E_x(0, t=25, moment=life.VARIANCE)
i = Recursion.solve(fun, target=0, grid=[0.058, 0.066])
isclose(0.06, i, question="Q4.7")
```

```
----- Q4.7 0.06: 0.06008023738770262 [OK] -----
```

```
True
```

SOA Question 4.8 : (C) 191

For a whole life insurance of 1000 on (50), you are given :

1. The death benefit is payable at the end of the year of death
2. Mortality follows the Standard Ultimate Life Table
3. $i = 0.04$ in the first year, and $i = 0.05$ in subsequent years

Calculate the actuarial present value of this insurance.

hints:

- use insurance recursion with special interest rate $i = 0.04$ in first year.

```python
def v_t(t): return 1.04**(-t) if t < 1 else 1.04**(-1) * 1.05**(-t+1)
A = SULT().set_interest(v_t=v_t).whole_life_insurance(50, b=1000)
isclose(191, A, question="Q4.8")
```

```
----- Q4.8 191: 191.1281281882354 [OK] -----
```

```
True
```

SOA Question 4.9 : (D) 0.5

You are given:

1. $A_{35:\overline{15|}} = 0.39$
2. $A^{1}_{35:\overline{15|}} = 0.25$
3. $A_{35} = 0.32$

Calculate A_{50}.

hints:

- solve ${}_{15}E_{35}$ from endowment insurance minus term insurance
- solve implicitly from whole life as term plus deferred insurance

```python
E = Recursion().set_A(0.39, x=35, t=15, endowment=1)\
               .set_A(0.25, x=35, t=15)\
               .E_x(35, t=15)
life = Recursion(verbose=False).set_A(0.32, x=35)\
                                .set_E(E, x=35, t=15)
def fun(A): return life.set_A(A, x=50).term_insurance(35, t=15)
A = life.solve(fun, target=0.25, grid=[0.35, 0.55])
isclose(0.5, A, question="Q4.9")
```

Pure Endowment ${}_{15}E_{x+35}$:=

${}_{15}E_{x+35} = A_{x+35:\overline{15|}} - A^{1}_{x+35:\overline{15|}}$ endowment insurance minus term

```
----- Q4.9 0.5: 0.5 [OK] -----
```

```
True
```

SOA Question 4.10 : (D)

The present value random variable for an insurance policy on (x) is expressed as: $$

Determine which of the following is a correct expression for $E[Z]$.

(A) $_{10|}\overline{A}_x +_{20|} \overline{A}_x -_{30|} \overline{A}_x$

(B) $\overline{A}_x +_{20} E_x \overline{A}_{x+20} - 2\ {}_{30}E_x \overline{A}_{x+30}$

(C) $_{10}E_x \overline{A}_x +_{20} E_x \overline{A}_{x+20} - 2\ {}_{30}E_x \overline{A}_{x+30}$

(D) $_{10}E_x \overline{A}_{x+10} +_{20} E_x \overline{A}_{x+20} - 2\ {}_{30}E_x \overline{A}_{x+30}$

(E) $_{10}E_x [\overline{A}_x +_{10} E_{x+10} + \overline{A}_{x+20} -_{10} E_{x+20} + \overline{A}_{x+30}]$

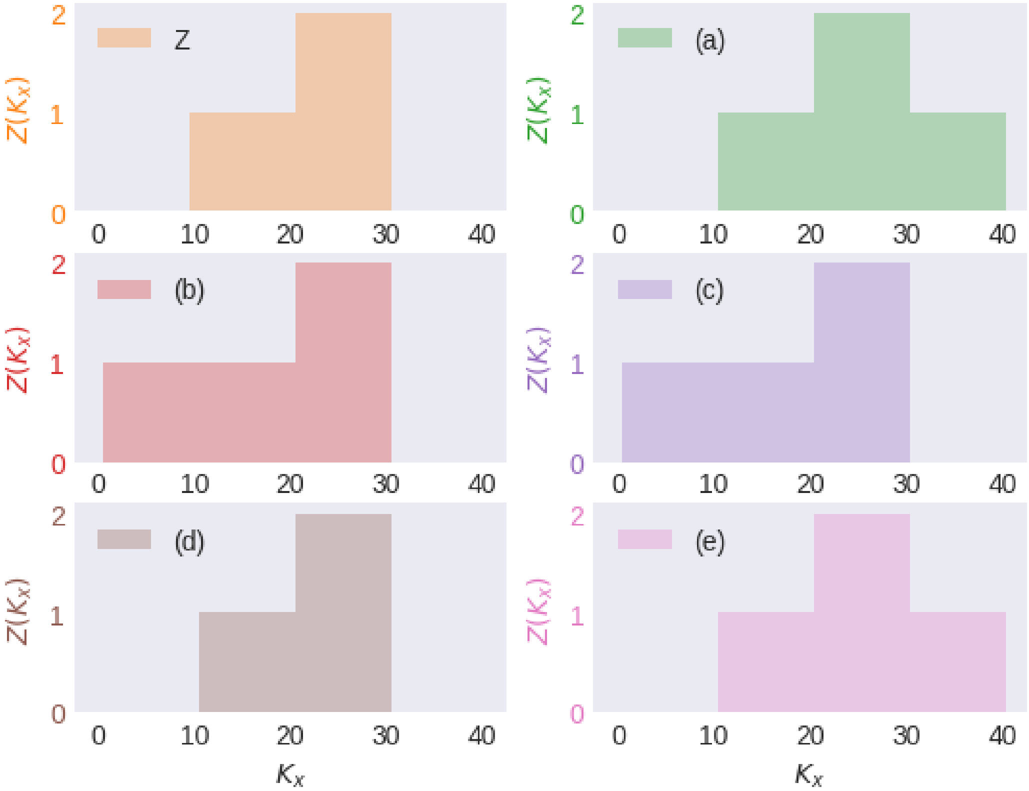
hints:

- draw and compare benefit diagrams

```
life = Insurance().set_interest(i=0.0).set_survival(S=lambda x,s,t: 1, maxage=40)
def fun(x, t):
    if 10 <= t <= 20: return life.interest.v_t(t)
    elif 20 < t <= 30: return 2 * life.interest.v_t(t)
    else: return 0
def A(x, t):  # Z_x+k (t-k)
    return life.interest.v_t(t - x) * (t > x)
x = 0
benefits=[lambda x,t: (life.E_x(x, t=10) * A(x+10, t)
                            + life.E_x(x, t=20)* A(x+20, t)
                            - life.E_x(x, t=30) * A(x+30, t)),
          lambda x,t: (A(x, t)
                       + life.E_x(x, t=20) * A(x+20, t)
                       - 2 * life.E_x(x, t=30) * A(x+30, t)),
          lambda x,t: (life.E_x(x, t=10) * A(x, t)
                       + life.E_x(x, t=20) * A(x+20, t)
                       - 2 * life.E_x(x, t=30) * A(x+30, t)),
          lambda x,t: (life.E_x(x, t=10) * A(x+10, t)
                       + life.E_x(x, t=20) * A(x+20, t)
                       - 2 * life.E_x(x, t=30) * A(x+30, t)),
          lambda x,t: (life.E_x(x, t=10)
                       * (A(x+10, t)
                          + life.E_x(x+10, t=10) * A(x+20, t)
                          - life.E_x(x+20, t=10) * A(x+30, t)))]
fig, ax = plt.subplots(3, 2)
ax = ax.ravel()
for i, b in enumerate([fun] + benefits):
    life.Z_plot(0, benefit=b, ax=ax[i], color=f"C{i+1}", title='')
    ax[i].legend(["(" + "abcde"[i-1] + ")" if i else "Z"])
z = [sum(abs(b(0, t) - fun(0, t)) for t in range(40)) for b in benefits]
ans = "ABCDE"[np.argmin(z)]
isclose('D', ans, question="Q4.10")
```

```
----- Q4.10 D: D [OK] -----
```

```
True
```



SOA Question 4.11 : (A) 143385

You are given:

1. Z_1 is the present value random variable for an n-year term insurance of 1000 issued to (x)
2. Z_2 is the present value random variable for an n-year endowment insurance of 1000 issued to (x)
3. For both Z_1 and Z_2 the death benefit is payable at the end of the year of death
4. $E[Z_1] = 528$
5. $Var(Z_2) = 15,000$
6. $A_{x:\overset{1}{\overline{n|}}} = 0.209$
7. ${}^2A_{x:\overset{1}{\overline{n|}}} = 0.136$

Calculate $Var(Z_1)$.

hints:

- compute endowment insurance = term insurance + pure endowment
- apply formula of variance as the difference of second moment and first moment squared.

```
A1 = 528/1000    # E[Z1]  term insurance
C1 = 0.209       # E[pure_endowment]
C2 = 0.136       # E[pure_endowment^2]
```

(continues on next page)

(continued from previous page)

```
B1 = A1 + C1     # endowment = term + pure_endowment
def fun(A2):
    B2 = A2 + C2   # double force of interest
    return Insurance.insurance_variance(A2=B2, A1=B1)
A2 = Insurance.solve(fun, target=15000/(1000*1000), grid=[143400, 279300])
var = Insurance.insurance_variance(A2=A2, A1=A1, b=1000)
isclose(143385, var, question="Q4.11")
```

```
----- Q4.11 143385: 143384.99999999997 [OK] -----
```

```
True
```

SOA Question 4.12 : (C) 167

For three fully discrete insurance products on the same (x), you are given:

- Z_1 is the present value random variable for a 20-year term insurance of 50
- Z_2 is the present value random variable for a 20-year deferred whole life insurance of 100
- Z_3 is the present value random variable for a whole life insurance of 100.
- $E[Z_1] = 1.65$ and $E[Z_2] = 10.75$
- $Var(Z_1) = 46.75$ and $Var(Z_2) = 50.78$

Calculate $Var(Z_3)$.

hints:

- since $Z_1,\ Z_2$ are non-overlapping, $E[Z_1\ Z_2] = 0$ for computing $Cov(Z_1, Z_2)$
- whole life is sum of term and deferred, hence equals variance of components plus twice their covariance

```
cov = Life.covariance(a=1.65, b=10.75, ab=0)   # E[Z1 Z2] = 0 nonoverlapping
var = Life.variance(a=2, b=1, var_a=46.75, var_b=50.78, cov_ab=cov)
isclose(167, var, question="Q4.12")
```

```
----- Q4.12 167: 166.82999999999998 [OK] -----
```

```
True
```

SOA Question 4.13 : (C) 350

For a 2-year deferred, 2-year term insurance of 2000 on [65], you are given:

1. The following select and ultimate mortality table with a 3-year select period:

x	$q_{[x]}$	$q_{[x]+1}$	$q_{[x]+2}$	q_{x+3}	$x+3$
65	0.08	0.10	0.12	0.14	68
66	0.09	0.11	0.13	0.15	69
67	0.10	0.12	0.14	0.16	70
68	0.11	0.13	0.15	0.17	71
69	0.12	0.14	0.16	0.18	72

2. $i = 0.04$

3. The death benefit is payable at the end of the year of death

Calculate the actuarial present value of this insurance.

hints:

- compute term insurance as EPV of benefits

```
life = SelectLife().set_table(q={65: [.08, .10, .12, .14],
                                 66: [.09, .11, .13, .15],
                                 67: [.10, .12, .14, .16],
                                 68: [.11, .13, .15, .17],
                                 69: [.12, .14, .16, .18]})\
                   .set_interest(i=.04)
A = life.deferred_insurance(65, t=2, u=2, b=2000)
isclose(350, A, question="Q4.13")
```

```
----- Q4.13 350: 351.0578236056159 [OK] -----
```

```
True
```

SOA Question 4.14 : (E) 390000

A fund is established for the benefit of 400 workers all age 60 with independent future lifetimes. When they reach age 85, the fund will be dissolved and distributed to the survivors.

The fund will earn interest at a rate of 5% per year.

The initial fund balance, F, is determined so that the probability that the fund will pay at least 5000 to each survivor is 86%, using the normal approximation.

Mortality follows the Standard Ultimate Life Table.

Calculate F.

hints:

- discount (by interest rate $i = 0.05$) the value at the portfolio percentile, of the sum of 400 bernoulli r.v. with survival probability ${}_{25}p_{60}$

```
sult = SULT()
p = sult.p_x(60, t=85-60)
mean = sult.bernoulli(p)
var = sult.bernoulli(p, variance=True)
F = sult.portfolio_percentile(mean=mean, variance=var, prob=.86, N=400)
F *= 5000 * sult.interest.v_t(85-60)
isclose(390000, F, question="Q4.14")
```

```
----- Q4.14 390000: 389322.86778416135 [OK] -----
```

```
True
```

SOA Question 4.15 : (E) 0.0833

For a special whole life insurance on (x), you are given :

- Death benefits are payable at the moment of death
- The death benefit at time t is $b_t = e^{0.02t}$, for $t \geq 0$

- $\mu_{x+t} = 0.04$, for $t \geq 0$
- $\delta = 0.06$
- Z is the present value at issue random variable for this insurance.

Calculate $Var(Z)$.

hints:

- this special benefit function has effect of reducing actuarial discount rate to use in constant force of mortality shortcut formulas

```
life = Insurance().set_survival(mu=lambda *x: 0.04).set_interest(delta=0.06)
benefit = lambda x,t: math.exp(0.02*t)
A1 = life.A_x(0, benefit=benefit, discrete=False)
A2 = life.A_x(0, moment=2, benefit=benefit, discrete=False)
var = life.insurance_variance(A2=A2, A1=A1)
isclose(0.0833, var, question="Q4.15")
```

```
----- Q4.15 0.0833: 0.08333333333333331 [OK] -----
```

```
True
```

SOA Question 4.16 : (D) 0.11

You are given the following extract of ultimate mortality rates from a two-year select and ultimate mortality table:

x	q_x
50	0.045
51	0.050
52	0.055
53	0.060

The select mortality rates satisfy the following:

1. $q_{[x]} = 0.7 q_x$
2. $q_{[x]+1} = 0.8 q_{x+1}$

You are also given that $i = 0.04$.

Calculate $A^{1}_{[50]:\overline{3}|}$.

hints:

- compute EPV of future benefits with adjusted mortality rates

```
q = [.045, .050, .055, .060]
q = {50 + x: [q[x] * 0.7 if x < len(q) else None,
              q[x+1] * 0.8 if x + 1 < len(q) else None,
              q[x+2] if x + 2 < len(q) else None]
     for x in range(4)}
life = SelectLife().set_table(q=q).set_interest(i=.04)
A = life.term_insurance(50, t=3)
isclose(0.1116, A, question="Q4.16")
```

```
----- Q4.16 0.1116: 0.1115661982248521 [OK] -----
```

```
True
```

SOA Question 4.17 : (A) 1126.7

For a special whole life policy on (48), you are given:

1. The policy pays 5000 if the insured's death is before the median curtate future lifetime at issue and 10,000 if death is after the median curtate future lifetime at issue
2. Mortality follows the Standard Ultimate Life Table
3. Death benefits are paid at the end of the year of death
4. i = 0.05

Calculate the actuarial present value of benefits for this policy.

hints:

- find future lifetime with 50% survival probability
- compute EPV of special whole life as sum of term and deferred insurance, that have different benefit amounts before and after median lifetime.

```
sult = SULT()
median = sult.Z_t(48, prob=0.5, discrete=False)
def benefit(x,t): return 5000 if t < median else 10000
A = sult.A_x(48, benefit=benefit)
isclose(1130, A, question="Q4.17")
```

```
----- Q4.17 1130: 1126.774772894844 [OK] -----
```

```
True
```

SOA Question 4.18 : (A) 81873

You are given that T, the time to first failure of an industrial robot, has a density f(t) given by

$$\begin{aligned} f(t) &= 0.1, \quad 0 \leq t < 2 \\ &= 0.4t^{-2}, \quad t \leq t < 10 \end{aligned}$$

with $f(t)$ undetermined on $[10, \infty)$.

Consider a supplemental warranty on this robot that pays 100,000 at the time T of its first failure if $2 \leq T \leq 10$, with no benefits payable otherwise. You are also given that $\delta = 5\%$. Calculate the 90th percentile of the present value of the future benefits under this warranty.

hints:

- find values of limits such that integral of lifetime density function equals required survival probability

```
def f(x,s,t): return 0.1 if t < 2 else 0.4*t**(-2)
life = Insurance().set_interest(delta=0.05)\
                  .set_survival(f=f, maxage=10)
def benefit(x,t): return 0 if t < 2 else 100000
prob = 0.9 - life.q_x(0, t=2)
```

(continues on next page)

(continued from previous page)

```
T = life.Z_t(0, prob=prob)
Z = life.Z_from_t(T, discrete=False) * benefit(0, T)
isclose(81873, Z, question="Q4.18")
```

```
----- Q4.18 81873: 81873.07530779815 [OK] -----
```

```
True
```

SOA Question 4.19 : (B) 59050

(80) purchases a whole life insurance policy of 100,000. You are given:

1. The policy is priced with a select period of one year
2. The select mortality rate equals 80% of the mortality rate from the Standard Ultimate Life Table
3. Ultimate mortality follows the Standard Ultimate Life Table
4. $i = 0.05$

Calculate the actuarial present value of the death benefits for this insurance

hints:

- calculate adjusted mortality for the one-year select period
- compute whole life insurance using backward recursion formula

```
life = SULT()
q = ExtraRisk(life=life, extra=0.8, risk="MULTIPLY_RATE")['q']
select = SelectLife(periods=1).set_select(s=0, age_selected=True, q=q)\
                              .set_select(s=1, age_selected=False, q=life['q'])\
                              .set_interest(i=.05)\
                              .fill_table()
A = 100000 * select.whole_life_insurance(80, s=0)
isclose(59050, A, question="Q4.19")
```

```
----- Q4.19 59050: 59050.59973285648 [OK] -----
```

```
True
```

22.5 5 Annuities

SOA Question 5.1 : (A) 0.705

You are given:

1. $\delta_t = 0.06, \quad t \geq 0$
2. $\mu_x(t) = 0.01, \quad t \geq 0$
3. Y is the present value random variable for a continuous annuity of 1 per year, payable for the lifetime of (x) with 10 years certain

Calculate $Pr(Y > E[Y])$.

hints:

- sum annuity certain and deferred life annuity with constant force of mortality shortcut
- apply equation for PV annuity r.v. Y to infer lifetime
- compute survival probability from constant force of mortality function.

```
life = ConstantForce(mu=0.01).set_interest(delta=0.06)
EY = life.certain_life_annuity(0, u=10, discrete=False)
p = life.p_x(0, t=life.Y_to_t(EY))
isclose(0.705, p, question="Q5.1")  # 0.705
```

```
----- Q5.1 0.705: 0.7053680433746505 [OK] -----
```

```
True
```

SOA Question 5.2 : (B) 9.64

You are given:

1. $A_x = 0.30$
2. $A_{x+n} = 0.40$
3. $A_{\overset{1}{x:\overline{n|}}} = 0.35$
4. *i* = 0.05

Calculate $a_{x:\overline{n|}}$.

hints:

- compute term life as difference of whole life and deferred insurance
- compute twin annuity-due, and adjust to an immediate annuity.

```
x, n = 0, 10
a = Recursion().set_interest(i=0.05)\
               .set_A(0.3, x)\
               .set_A(0.4, x+n)\
               .set_E(0.35, x, t=n)\
               .immediate_annuity(x, t=n)
isclose(9.64, a, question="Q5.2")
```

Whole Life Annuity $\ddot{a}_x$:=
$\ddot{a}_x = [1 - A_x]/d$ insurance twin

Whole Life Annuity $\ddot{a}_{x+10}$:=
$\ddot{a}_{x+10} = [1 - A_{x+10}]/d$ insurance twin

```
----- Q5.2 9.64: 9.639999999999999 [OK] -----
```

```
True
```

SOA Question 5.3 : (C) 6.239

You are given:

- Mortality follows the Standard Ultimate Life Table
- Deaths are uniformly distributed over each year of age
- i = 0.05

Calculate $\frac{d}{dt}(\overline{I}\overline{a})_{40:\overline{t}|}$ at $t = 10.5$.

hints:

- Differential reduces to be the EPV of the benefit payment at the upper time limit.

```
t = 10.5
E = t * SULT().E_r(40, t=t)
isclose(6.239, E, question="Q5.3")
```

```
----- Q5.3 6.239: 6.23871918627528 [OK] -----
```

```
True
```

SOA Question 5.4 : (A) 213.7

(40) wins the SOA lottery and will receive both:

- A deferred life annuity of K per year, payable continuously, starting at age $40 + \overset{\circ}{e}_{40}$ and
- An annuity certain of K per year, payable continuously, for $\overset{\circ}{e}_{40}$ years

You are given:

1. $\mu = 0.02$
2. $\delta = 0.01$
3. The actuarial present value of the payments is 10,000

Calculate K.

hints:

- compute certain and life annuity factor as the sum of a certain annuity and a deferred life annuity.
- solve for amount of annual benefit that equals given EPV

```
life = ConstantForce(mu=0.02).set_interest(delta=0.01)
u = life.e_x(40, curtate=False)
P = 10000 / life.certain_life_annuity(40, u=u, discrete=False)
isclose(213.7, P, question="Q5.4") # 213.7
```

```
----- Q5.4 213.7: 213.74552118275955 [OK] -----
```

```
True
```

SOA Question 5.5 : (A) 1699.6

For an annuity-due that pays 100 at the beginning of each year that (45) is alive, you are given:

1. Mortality for standard lives follows the Standard Ultimate Life Table

2. The force of mortality for standard lives age 45 + t is represented as μ_{45+t}^{SULT}
3. The force of mortality for substandard lives age 45 + t, μ_{45+t}^{S}, is defined as:

$$\begin{aligned}\mu_{45+t}^{S} &= \mu_{45+t}^{SULT} + 0.05, \quad 0 \leq t < 1 \\ &= \mu_{45+t}^{SULT}, \quad t \geq 1\end{aligned}$$

4. $i = 0.05$

Calculate the actuarial present value of this annuity for a substandard life age 45.

hints:

- adjust mortality rate for the extra risk
- compute annuity by backward recursion.

```
life = SULT()    # start with SULT life table
q = ExtraRisk(life=life, extra=0.05, risk="ADD_FORCE")['q']
select = SelectLife(periods=1).set_select(s=0, age_selected=True, q=q)\
                              .set_select(s=1, age_selected=False, a=life['a'])\
                              .set_interest(i=0.05)\
                              .fill_table()
a = 100 * select['a'][45][0]
isclose(1700, a, question="Q5.5")
```

```
----- Q5.5 1700: 1699.6076593190103 [OK] -----
```

```
True
```

SOA Question 5.6 : (D) 1200

For a group of 100 lives age x with independent future lifetimes, you are given:

1. Each life is to be paid 1 at the beginning of each year, if alive
2. $A_x = 0.45$
3. ${}^2A_x = 0.22$
4. $i = 0.05$
5. Y is the present value random variable of the aggregate payments.

Using the normal approximation to Y, calculate the initial size of the fund needed to be 95% certain of being able to make the payments for these life annuities.

hints:

- compute mean and variance of EPV of whole life annuity from whole life insurance twin and variance identities.
- portfolio percentile of the sum of $N = 100$ life annuity payments

```
life = Annuity().set_interest(i=0.05)
var = life.annuity_variance(A2=0.22, A1=0.45)
mean = life.annuity_twin(A=0.45)
fund = life.portfolio_percentile(mean, var, prob=.95, N=100)
isclose(1200, fund, question="Q5.6")
```

```
----- Q5.6 1200: 1200.6946732201702 [OK] -----
```

```
True
```

SOA Question 5.7 : (C)

You are given:

1. $A_{35} = 0.188$
2. $A_{65} = 0.498$
3. $_{30}p_{35} = 0.883$
4. $i = 0.04$

Calculate $1000\ddot{a}^{(2)}_{35:\overline{30|}}$ using the two-term Woolhouse approximation.

hints:

- compute endowment insurance from relationships of whole life, temporary and deferred insurances.
- compute temporary annuity from insurance twin
- apply Woolhouse approximation

```
life = Recursion().set_interest(i=0.04)\
                  .set_A(0.188, x=35)\
                  .set_A(0.498, x=65)\
                  .set_p(0.883, x=35, t=30)
mthly = Woolhouse(m=2, life=life, three_term=False)
a = 1000 * mthly.temporary_annuity(35, t=30)
isclose(17376.7, a, question="Q5.7")
```

Whole Life Annuity $\ddot{a}_{x+35}$:=
$\ddot{a}_{x+35} = [1 - A_{x+35}]/d$ insurance twin

Whole Life Annuity $\ddot{a}_{x+65}$:=
$\ddot{a}_{x+65} = [1 - A_{x+65}]/d$ insurance twin

Pure Endowment $_{30}E_{x+35}$:=
$_{30}E_{x+35} = {}_{30}p_{x+35} * v^{30}$ pure endowment

```
----- Q5.7 17376.7: 17376.71459632958 [OK] -----
```

```
True
```

SOA Question 5.8 : (C) 0.92118

For an annual whole life annuity-due of 1 with a 5-year certain period on (55), you are given:

1. Mortality follows the Standard Ultimate Life Table
2. i = 0.05

Calculate the probability that the sum of the undiscounted payments actually made under this annuity will exceed the expected present value, at issue, of the annuity.

hints:

- calculate EPV of certain and life annuity.

- find survival probability of lifetime s.t. sum of annual payments exceeds EPV

```
sult = SULT()
a = sult.certain_life_annuity(55, u=5)
p = sult.p_x(55, t=math.floor(a))
isclose(0.92118, p, question="Q5.8")
```

```
----- Q5.8 0.92118: 0.9211799771029529 [OK] -----
```

```
True
```

SOA Question 5.9 : (C) 0.015

hints:

- express both EPV's expressed as forward recursions
- solve for unknown constant k.

```
x, p = 0, 0.9  # set arbitrary p_x = 0.9
a = Recursion().set_a(21.854, x=x)\
               .set_p(p, x=x)\
               .whole_life_annuity(x+1)
life = Recursion(verbose=False).set_a(22.167, x=x)
def fun(k): return a - life.set_p((1 + k) * p, x=x).whole_life_annuity(x + 1)
k = life.solve(fun, target=0, grid=[0.005, 0.025])
isclose(0.015, k, question="Q5.9")
```

$$\begin{array}{ll} \texttt{Whole Life Annuity}\ \ddot{a}_{x+1} := & \\ \ddot{a}_{x+1} = [\,\ddot{a}_x - 1] /\ E_x & \texttt{forward recursion} \\ E_x = \ p_x * v & \texttt{pure endowment} \end{array}$$

```
----- Q5.9 0.015: 0.015009110961925157 [OK] -----
```

```
True
```

22.6 6 Premium Calculation

SOA Question 6.1 : (D) 35.36

6.1. You are given the following information about a special fully discrete 2-payment, 2-year term insurance on (80):

(i) Mortality follows the Standard Ultimate Life Table

(ii) $i = 0.03$

(iii) The death benefit is 1000 plus a return of all premiums paid without interest

(iv) Level premiums are calculated using the equivalence principle

Calculate the net premium for this special insurance.

[A modified version of Question 22 on the Fall 2012 exam]

hints:

- solve net premium such that EPV annuity = EPV insurance + IA factor for returns of premiums without interest

```
P = SULT().set_interest(i=0.03)\
          .net_premium(80, t=2, b=1000, return_premium=True)
isclose(35.36, P, question="Q6.1")
```

```
----- Q6.1 35.36: 35.35922286190033 [OK] -----
```

```
True
```

SOA Question 6.2 : (E) 3604

6.2. For a fully discrete 10-year term life insurance policy on (*x*), you are given:

(i) Death benefits are 100,000 plus the return of all gross premiums paid without interest

(ii) Expenses are 50% of the first year's gross premium, 5% of renewal gross premiums and 200 per policy expenses each year

(iii) Expenses are payble at the beginnig of the year

(iv) $A^{1}_{x:\overline{10|}} = 0.17094$

(v) $(IA)^{1}_{x:\overline{10|}} = 0.96728$

(vi) $\ddot{a}^{1}_{x:\overline{10|}} = 6.8865$

Calculate the gross premium using the equivalence principle.

[Question 25 on the Fall 2012 exam]

hints:

- EPV return of premiums without interest = Premium $\times$ IA factor
- solve for gross premiums such that EPV premiums = EPV benefits and expenses

```
life = Premiums()
A, IA, a = 0.17094, 0.96728, 6.8865
P = life.gross_premium(a=a, A=A, IA=IA, benefit=100000,
                       initial_premium=0.5, renewal_premium=.05,
                       renewal_policy=200, initial_policy=200)
isclose(3604, P, question="Q6.2")
```

```
----- Q6.2 3604: 3604.229940320728 [OK] -----
```

```
True
```

SOA Question 6.3 : (C) 0.390

S, now age 65, purchased a 20-year deferred whole life annuity-due of 1 per year at age 45. You are given:

1. Equal annual premiums, determined using the equivalence principle, were paid at the beginning of each year during the deferral period
2. Mortality at ages 65 and older follows the Standard Ultimate Life Table
3. $i = 0.05$
4. Y is the present value random variable at age 65 for S's annuity benefits

Calculate the probability that Y is less than the actuarial accumulated value of S's premiums.

hints:

- solve lifetime t such that PV annuity certain = PV whole life annuity at age 65
- calculate mortality rate through the year before curtate lifetime

```
life = SULT()
t = life.Y_to_t(life.whole_life_annuity(65))
q = 1 - life.p_x(65, t=math.floor(t) - 1)
isclose(0.39, q, question="Q6.3")
```

```
----- Q6.3 0.39: 0.39039071872030084 [OK] -----
```

```
True
```

SOA Question 6.4 : (E) 1890

For whole life annuities-due of 15 per month on each of 200 lives age 62 with independent future lifetimes, you are given:

1. $i = 0.06$
2. $A_{62}^{12} = 0.4075$ and ${}^2A_{62}^{(12)} = 0.2105$
3. π is the single premium to be paid by each of the 200 lives
4. S is the present value random variable at time 0 of total payments made to the 200 lives

Using the normal approximation, calculate π such at $Pr(200\pi > S) = 0.90$

```
mthly = Mthly(m=12, life=Annuity().set_interest(i=0.06))
A1, A2 = 0.4075, 0.2105
mean = mthly.annuity_twin(A1) * 15 * 12
var = mthly.annuity_variance(A1=A1, A2=A2, b=15 * 12)
S = Annuity.portfolio_percentile(mean=mean, variance=var, prob=.9, N=200) / 200
isclose(1890, S, question="Q6.4")
```

```
----- Q6.4 1890: 1893.912859650868 [OK] -----
```

```
True
```

SOA Question 6.5 : (D) 33

For a fully discrete whole life insurance of 1000 on (30), you are given:

1. Mortality follows the Standard Ultimate Life Table
2. $i = 0.05$
3. The premium is the net premium

Calculate the first year for which the expected present value at issue of that year's premium is less than the expected present value at issue of that year's benefit.

```
life = SULT()
P = life.net_premium(30, b=1000)
def gain(k):
    return life.Y_x(30, t=k) * P - life.Z_x(30, t=k) * 1000
```

(continues on next page)

(continued from previous page)

```
k = min([k for k in range(100) if gain(k) < 0]) + 1  # add 1 because k=0 is first␣
↪policy year
isclose(33, k, question="Q6.5")
```

```
----- Q6.5 33: 33 [OK] -----
```

```
True
```

SOA Question 6.6 : (B) 0.79

For fully discrete whole life insurance policies of 10,000 issued on 600 lives with independent future lifetimes, each age 62, you are given:

1. Mortality follows the Standard Ultimate Life Table
2. $i = 0.05$
3. Expenses of 5% of the first year gross premium are incurred at issue
4. Expenses of 5 per policy are incurred at the beginning of each policy year
5. The gross premium is 103% of the net premium.
6. ${}_0L$ is the aggregate present value of future loss at issue random variable

Calculate $Pr({}_0L < 40,000)$, using the normal approximation.

```
life = SULT()
P = life.net_premium(62, b=10000)
contract = Contract(premium=1.03*P,
                    renewal_policy=5,
                    initial_policy=5,
                    initial_premium=0.05,
                    benefit=10000)
L = life.gross_policy_value(62, contract=contract)
var = life.gross_policy_variance(62, contract=contract)
prob = life.portfolio_cdf(mean=L, variance=var, value=40000, N=600)
isclose(.79, prob, question="Q6.6")
```

```
----- Q6.6 0.79: 0.7914321142683509 [OK] -----
```

```
True
```

SOA Question 6.7 : (C) 2880

For a special fully discrete 20-year endowment insurance on (40), you are given:

1. The only death benefit is the return of annual net premiums accumulated with interest at 5% to the end of the year of death
2. The endowment benefit is 100,000
3. Mortality follows the Standard Ultimate Life Table
4. $i = 0.05$

Calculate the annual net premium.

```
life = SULT()
a = life.temporary_annuity(40, t=20)
A = life.E_x(40, t=20)
IA = a - life.interest.annuity(t=20) * life.p_x(40, t=20)
G = life.gross_premium(a=a, A=A, IA=IA, benefit=100000)
isclose(2880, G, question="Q6.7")
```

```
----- Q6.7 2880: 2880.2463991134578 [OK] -----
```

```
True
```

SOA Question 6.8 : (B) 9.5

For a fully discrete whole life insurance on (60), you are given:

1. Mortality follows the Standard Ultimate Life Table
2. $i = 0.05$
3. The expected company expenses, payable at the beginning of the year, are:

- 50 in the first year
- 10 in years 2 through 10
- 5 in years 11 through 20
- 0 after year 20

Calculate the level annual amount that is actuarially equivalent to the expected company expenses.

hints:

- calculate EPV of expenses as deferred life annuities
- solve for level premium

```
life = SULT()
initial_cost = (50 + 10 * life.deferred_annuity(60, u=1, t=9)
                + 5 * life.deferred_annuity(60, u=10, t=10))
P = life.net_premium(60, initial_cost=initial_cost)
isclose(9.5, P, question="Q6.8")
```

```
----- Q6.8 9.5: 9.526003201821927 [OK] -----
```

```
True
```

SOA Question 6.9 : (D) 647

```
life = SULT()
a = life.temporary_annuity(50, t=10)
A = life.term_insurance(50, t=20)
initial_cost = 25 * life.deferred_annuity(50, u=10, t=10)
P = life.gross_premium(a=a, A=A, benefit=100000,
                       initial_premium=0.42, renewal_premium=0.12,
                       initial_policy=75 + initial_cost, renewal_policy=25)
isclose(647, P, question="Q6.9")
```

```
----- Q6.9 647: 646.8608151974504 [OK] -----
```

```
True
```

SOA Question 6.10 : (D) 0.91

For a fully discrete 3-year term insurance of 1000 on (*x*), you are given:

1. $p_x = 0.975$
2. $i = 0.06$
3. The actuarial present value of the death benefit is 152.85
4. The annual net premium is 56.05

Calculate p_{x+2}.

```
x = 0
life = Recursion(depth=5).set_interest(i=0.06)\
                         .set_p(0.975, x=x)\
                         .set_a(152.85/56.05, x=x, t=3)\
                         .set_A(152.85, x=x, t=3, b=1000)
p = life.p_x(x=x+2)
isclose(0.91, p, question="Q6.10")
```

$$
\begin{array}{ll}
\texttt{Survival}\ p_{x+2}\ := & \\
p_{x+2} = E_{x+2}/v & \texttt{one-year pure endowment} \\
E_{x+2} = A_{x+2:\overline{1}|} - A^{1}_{x+2:\overline{1}|} & \texttt{endowment insurance minus term} \\
A^{1}_{x+2:\overline{1}|} = [\, A^{1}_{x+1:\overline{2}|}/v - q_{x+1} * b]/\, p_{x+1} & \texttt{forward recursion} \\
p_{x+1} = [\, \ddot{a}_{x+1:\overline{2}|} - 1]/[v * \ddot{a}_{x+2:\overline{1}|}] & \texttt{annuity recursion} \\
\ddot{a}_{x+1:\overline{2}|} = [\, \ddot{a}_{x:\overline{3}|} - 1]/\, E_x & \texttt{forward recursion} \\
A^{1}_{x+1:\overline{2}|} = [\, A^{1}_{x:\overline{3}|}/v - q_x * b]/\, p_x & \texttt{forward recursion} \\
E_x = p_x * v & \texttt{pure endowment}
\end{array}
$$

```
----- Q6.10 0.91: 0.9097382950525702 [OK] -----
```

```
True
```

SOA Question 6.11 : (C) 0.041

```
life = Recursion().set_interest(i=0.04)
A = life.set_A(0.39788, 51)\
        .set_q(0.0048, 50)\
        .whole_life_insurance(50)
P = life.gross_premium(A=A, a=life.annuity_twin(A=A))
A = life.set_q(0.048, 50).whole_life_insurance(50)
loss = A - life.annuity_twin(A) * P
isclose(0.041, loss, question="Q6.11")
```

$$
\begin{array}{ll}
\texttt{Whole Life Insurance}\ A_{x+50}\ := & \\
A_{x+50} = v * [\, q_{x+50} * b + p_{x+50} * A_{x+51}] & \texttt{backward recursion} \\
p_{x+50} = 1 - q_{x+50} & \texttt{complement of mortality}
\end{array}
$$

$\texttt{Whole Life Insurance}\ A_{x+50}$:=

$A_{x+50} = v * [\, q_{x+50} * b + \ p_{x+50} * \ A_{x+51}]$ `backward recursion`

$p_{x+50} = 1 - \ q_{x+50}$ `complement of mortality`

```
----- Q6.11 0.041: 0.04069206883563675 [OK] -----
```

```
True
```

SOA Question 6.12 : (E) 88900

For a fully discrete whole life insurance of 1000 on (x), you are given:

1. The following expenses are incurred at the beginning of each year:

	Year 1	Years 2+
Percent of premium	75%	10%
Maintenance expenses	10	2

2. An additional expense of 20 is paid when the death benefit is paid
3. The gross premium is determined using the equivalence principle
4. $i = 0.06$
5. $\ddot{a}_x = 12.0$
6. ${}^2A_x = 0.14$

Calculate the variance of the loss at issue random variable.

```
life = PolicyValues().set_interest(i=0.06)
a = 12
A = life.insurance_twin(a)
contract = Contract(benefit=1000, settlement_policy=20,
                        initial_policy=10, initial_premium=0.75,
                        renewal_policy=2, renewal_premium=0.1)
contract.premium = life.gross_premium(A=A, a=a, **contract.premium_terms)
L = life.gross_variance_loss(A1=A, A2=0.14, contract=contract)
isclose(88900, L, question="Q6.12")
```

```
----- Q6.12 88900: 88862.59592874818 [OK] -----
```

```
True
```

SOA Question 6.13 : (D) -400

For a fully discrete whole life insurance of 10,000 on (45), you are given:

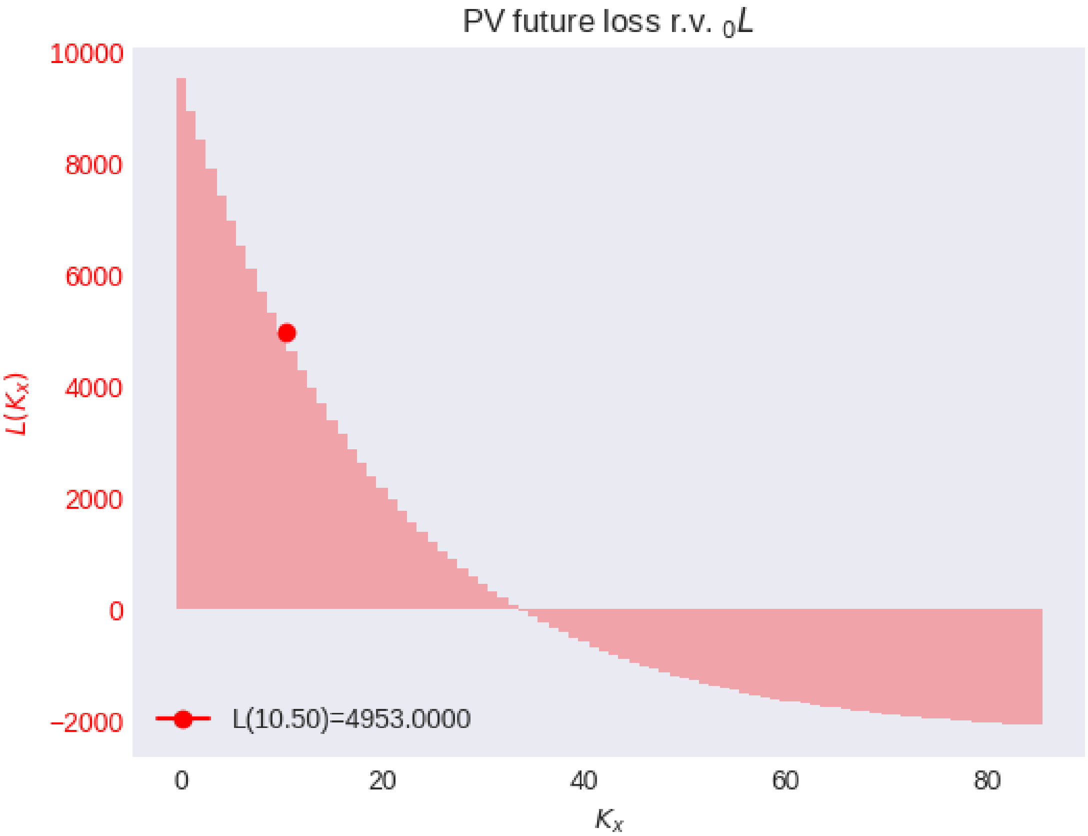
1. Commissions are 80% of the first year premium and 10% of subsequent premiums. There are no other expenses
2. Mortality follows the Standard Ultimate Life Table
3. $i = 0.05$
4. ${}_0L$ denotes the loss at issue random variable
5. If $T_{45} = 10.5$, then ${}_0L = 4953$

Calculate $E[{}_0L]$.

```
life = SULT().set_interest(i=0.05)
A = life.whole_life_insurance(45)
contract = Contract(benefit=10000, initial_premium=.8, renewal_premium=.1)
def fun(P):   # Solve for premium, given Loss(t=0) = 4953
    return life.L_from_t(t=10.5, contract=contract.set_contract(premium=P))
contract.set_contract(premium=life.solve(fun, target=4953, grid=100))
L = life.gross_policy_value(45, contract=contract)
life.L_plot(x=45, T=10.5, contract=contract)
isclose(-400, L, question="Q6.13")
```

```
----- Q6.13 -400: -400.94447599879277 [OK] -----
```

```
True
```



SOA Question 6.14 : (D) 1150

For a special fully discrete whole life insurance of 100,000 on (40), you are given:

1. The annual net premium is P for years 1 through 10, $0.5P$ for years 11 through 20, and 0 thereafter
2. Mortality follows the Standard Ultimate Life Table
3. $i = 0.05$

Calculate P.

```
life = SULT().set_interest(i=0.05)
a = life.temporary_annuity(40, t=10) + 0.5*life.deferred_annuity(40, u=10, t=10)
A = life.whole_life_insurance(40)
P = life.gross_premium(a=a, A=A, benefit=100000)
isclose(1150, P, question="Q6.14")
```

```
----- Q6.14 1150: 1148.5800555155263 [OK] -----
```

```
True
```

SOA Question 6.15 : (B) 1.002

For a fully discrete whole life insurance of 1000 on (x) with net premiums payable quarterly, you are given:

1. $i = 0.05$
2. $\ddot{a}_x = 3.4611$
3. $P^{(W)}$ and $P^{(UDD)}$ are the annualized net premiums calculated using the 2-term Woolhouse (W) and the uniform distribution of deaths (UDD) assumptions, respectively

Calculate $\dfrac{P^{(UDD)}}{P^{(W)}}$.

```
life = Recursion().set_interest(i=0.05).set_a(3.4611, x=0)
A = life.insurance_twin(3.4611)
udd = UDD(m=4, life=life)
a1 = udd.whole_life_annuity(x=x)
woolhouse = Woolhouse(m=4, life=life)
a2 = woolhouse.whole_life_annuity(x=x)
P = life.gross_premium(a=a1, A=A)/life.gross_premium(a=a2, A=A)
isclose(1.002, P, question="Q6.15")
```

```
----- Q6.15 1.002: 1.0022973504113772 [OK] -----
```

```
True
```

SOA Question 6.16 : (A) 2408.6

For a fully discrete 20-year endowment insurance of 100,000 on (30), you are given:

1. d = 0.05
2. Expenses, payable at the beginning of each year, are:

	First Year	First Year	Renewal Years	Renewal Years
	Percent of Premium	Per Policy	Percent of Premium	Per Policy
Taxes	4%	0	4%	0
Sales Commission	35%	0	2%	0
Policy Maintenance	0%	250	0%	50

3. The net premium is 2143

Calculate the gross premium using the equivalence principle.

```
life = Premiums().set_interest(d=0.05)
A = life.insurance_equivalence(premium=2143, b=100000)
a = life.annuity_equivalence(premium=2143, b=100000)
p = life.gross_premium(A=A, a=a, benefit=100000, settlement_policy=0,
                       initial_policy=250, initial_premium=0.04 + 0.35,
                       renewal_policy=50, renewal_premium=0.04 + 0.02)
isclose(2410, p, question="Q6.16")
```

```
----- Q6.16 2410: 2408.575206281868 [OK] -----
```

```
True
```

SOA Question 6.17 : (A) -30000

An insurance company sells special fully discrete two-year endowment insurance policies to smokers (S) and non-smokers (NS) age x. You are given:

1. The death benefit is 100,000; the maturity benefit is 30,000
2. The level annual premium for non-smoker policies is determined by the equivalence principle
3. The annual premium for smoker policies is twice the non-smoker annual premium
4. $\mu^{NS}_{x+t} = 0.1. \quad t > 0$
5. $q^{S}_{x+k} = 1.5 q^{NS}_{x+k}$, for $k = 0, 1$
6. $i = 0.08$

Calculate the expected present value of the loss at issue random variable on a smoker policy.

```
x = 0
life = ConstantForce(mu=0.1).set_interest(i=0.08)
A = life.endowment_insurance(x, t=2, b=100000, endowment=30000)
a = life.temporary_annuity(x, t=2)
P = life.gross_premium(a=a, A=A)
life1 = Recursion().set_interest(i=0.08)\
                   .set_q(life.q_x(x, t=1) * 1.5, x=x, t=1)\
                   .set_q(life.q_x(x+1, t=1) * 1.5, x=x+1, t=1)
contract = Contract(premium=P*2, benefit=100000, endowment=30000)
L = life1.gross_policy_value(x, t=0, n=2, contract=contract)
isclose(-30000, L, question="Q6.17")
```

$$
\begin{array}{ll}
\texttt{Term Insurance } A^{1}_{x:\overline{2|}} \ := & \\
A^{1}_{x:\overline{2|}} = A_{x:\overline{2|}} - {}_{2}E_x & \texttt{endowment insurance - pure} \\
{}_{2}E_x = {}_{2}p_x * v^2 & \texttt{pure endowment} \\
{}_{2}p_x = p_{x+1} * p_x & \texttt{survival chain rule} \\
A^{1}_{x:\overline{2|}} = v * [\, q_x * b + p_x * A^{1}_{x+1:\overline{1|}}] & \texttt{backward recursion} \\
p_{x+1} = 1 - q_{x+1} & \texttt{complement of mortality} \\
E_x = p_x * v & \texttt{pure endowment} \\
p_x = 1 - q_x & \texttt{complement of mortality}
\end{array}
$$

$$
\begin{array}{ll}
\texttt{Temporary Annuity } \ddot{a}_{x:\overline{2|}} \ := & \\
\ddot{a}_{x:\overline{2|}} = 1 + E_x * \ddot{a}_{x+1:\overline{1|}} & \texttt{backward recursion} \\
E_x = p_x * v & \texttt{pure endowment} \\
p_x = 1 - q_x & \texttt{complement of mortality}
\end{array}
$$

Pure Endowment ${}_2E_x$:=
${}_2E_x = {}_2p_x * v^2$ pure endowment
${}_2p_x = p_{x+1} * p_x$ survival chain rule
$p_x = 1 - q_x$ complement of mortality
$p_{x+1} = 1 - q_{x+1}$ complement of mortality

```
----- Q6.17 -30000: -30107.42633581115 [OK] -----
```

```
True
```

SOA Question 6.18 : (D) 166400

For a 20-year deferred whole life annuity-due with annual payments of 30,000 on (40), you are given:

1. The single net premium is refunded without interest at the end of the year of death if death occurs during the deferral period
2. Mortality follows the Standard Ultimate Life Table
3. $i = 0.05$

Calculate the single net premium for this annuity.

```
life = SULT().set_interest(i=0.05)
def fun(P):
    A = (life.term_insurance(40, t=20, b=P)
         + life.deferred_annuity(40, u=20, b=30000))
    return life.gross_premium(a=1, A=A) - P
P = life.solve(fun, target=0, grid=[162000, 168800])
isclose(166400, P, question="Q6.18")
```

```
----- Q6.18 166400: 166362.83871487685 [OK] -----
```

```
True
```

SOA Question 6.19 : (B) 0.033

```
life = SULT()
contract = Contract(initial_policy=.2, renewal_policy=.01)
a = life.whole_life_annuity(50)
A = life.whole_life_insurance(50)
contract.premium = life.gross_premium(A=A, a=a, **contract.premium_terms)
L = life.gross_policy_variance(50, contract=contract)
isclose(0.033, L, question="Q6.19")
```

```
----- Q6.19 0.033: 0.03283273381910885 [OK] -----
```

```
True
```

SOA Question 6.20 : (B) 459

For a special fully discrete 3-year term insurance on (75), you are given:

1. The death benefit during the first two years is the sum of the net premiums paid without interest
2. The death benefit in the third year is 10,000

x	p_x
75	0.90
76	0.88
77	0.85

3. $i = 0.04$

Calculate the annual net premium.

```
life = LifeTable().set_interest(i=.04).set_table(p={75: .9, 76: .88, 77: .85})
a = life.temporary_annuity(75, t=3)
IA = life.increasing_insurance(75, t=2)
A = life.deferred_insurance(75, u=2, t=1)
def fun(P): return life.gross_premium(a=a, A=P*IA + A*10000) - P
P = life.solve(fun, target=0, grid=[449, 489])
isclose(459, P, question="Q6.20")
```

```
----- Q6.20 459: 458.83181728297353 [OK] -----
```

```
True
```

SOA Question 6.21 : (C) 100

```
life = Recursion(verbose=False).set_interest(d=0.04)
life.set_A(0.7, x=75, t=15, endowment=1)
life.set_E(0.11, x=75, t=15)
def fun(P):
    return (P * life.temporary_annuity(75, t=15) -
            life.endowment_insurance(75, t=15, b=1000, endowment=15*float(P)))
P = life.solve(fun, target=0, grid=(80, 120))
isclose(100, P, question="Q6.21")
```

```
----- Q6.21 100: 100.85470085470084 [OK] -----
```

```
True
```

SOA Question 6.22 : (C) 102

For a whole life insurance of 100,000 on (45) with premiums payable monthly for a period of 20 years, you are given:

1. The death benefit is paid immediately upon death
2. Mortality follows the Standard Ultimate Life Table
3. Deaths are uniformly distributed over each year of age
4. $i = 0.05$

Calculate the monthly net premium.

```
life=SULT(udd=True)
a = UDD(m=12, life=life).temporary_annuity(45, t=20)
A = UDD(m=0, life=life).whole_life_insurance(45)
P = life.gross_premium(A=A, a=a, benefit=100000) / 12
isclose(102, P, question="Q6.22")
```

```
----- Q6.22 102: 102.40668704849178 [OK] -----
```

```
True
```

SOA Question 6.23 : (D) 44.7

```
x = 0
life = Recursion().set_a(15.3926, x=x)\
                  .set_a(10.1329, x=x, t=15)\
                  .set_a(14.0145, x=x, t=30)
def fun(P):
    per_policy = 30 + (30 * life.whole_life_annuity(x))
    per_premium = (0.6 + 0.1*life.temporary_annuity(x, t=15)
                    + 0.1*life.temporary_annuity(x, t=30))
    a = life.temporary_annuity(x, t=30)
    return (P * a) - (per_policy + per_premium * P)
P = life.solve(fun, target=0, grid=[30.3, 49.5])
isclose(44.7, P, question="Q6.23")
```

```
----- Q6.23 44.7: 44.70806635781144 [OK] -----
```

```
True
```

SOA Question 6.24 : (E) 0.30

For a fully continuous whole life insurance of 1 on (x), you are given:

1. L is the present value of the loss at issue random variable if the premium rate is determined by the equivalence principle
2. L^* is the present value of the loss at issue random variable if the premium rate is 0.06
3. $\delta = 0.07$
4. $\overline{A}_x = 0.30$
5. $Var(L) = 0.18$

Calculate $Var(L^*)$.

```
life = PolicyValues().set_interest(delta=0.07)
x, A1 = 0, 0.30   # Policy for first insurance
P = life.premium_equivalence(A=A1, discrete=False)  # Need its premium
contract = Contract(premium=P, discrete=False)
def fun(A2):  # Solve for A2, given Var(Loss)
    return life.gross_variance_loss(A1=A1, A2=A2, contract=contract)
A2 = life.solve(fun, target=0.18, grid=0.18)

contract = Contract(premium=0.06, discrete=False) # Solve second insurance
var = life.gross_variance_loss(A1=A1, A2=A2, contract=contract)
isclose(0.304, var, question="Q6.24")
```

```
----- Q6.24 0.304: 0.30419999999999975 [OK] -----
```

```
True
```

SOA Question 6.25 : (C) 12330

For a fully discrete 10-year deferred whole life annuity-due of 1000 per month on (55), you are given:

1. The premium, G, will be paid annually at the beginning of each year during the deferral period
2. Expenses are expected to be 300 per year for all years, payable at the beginning of the year
3. Mortality follows the Standard Ultimate Life Table
4. $i = 0.05$
5. Using the two-term Woolhouse approximation, the expected loss at issue is -800

Calculate G.

```
life = SULT()
woolhouse = Woolhouse(m=12, life=life)
benefits = woolhouse.deferred_annuity(55, u=10, b=1000 * 12)
expenses = life.whole_life_annuity(55, b=300)
payments = life.temporary_annuity(55, t=10)
def fun(P):
    return life.gross_future_loss(A=benefits + expenses, a=payments,
                                  contract=Contract(premium=P))
P = life.solve(fun, target=-800, grid=[12110, 12550])
isclose(12330, P, question="Q6.25")
```

```
----- Q6.25 12330: 12325.781125438532 [OK] -----
```

```
True
```

SOA Question 6.26 : (D) 180

For a special fully discrete whole life insurance policy of 1000 on (90), you are given:

1. The first year premium is 0
2. *P* is the renewal premium
3. Mortality follows the Standard Ultimate Life Table
4. $i = 0.05$
5. Premiums are calculated using the equivalence principle

Calculate *P*.

```
life = SULT().set_interest(i=0.05)
def fun(P):
    return P - life.net_premium(90, b=1000, initial_cost=P)
P = life.solve(fun, target=0, grid=[150, 190])
isclose(180, P, question="Q6.26")
```

```
----- Q6.26 180: 180.03164891315885 [OK] -----
```

```
True
```

SOA Question 6.27 : (D) 10310

For a special fully continuous whole life insurance on (x), you are given:

1. Premiums and benefits:

	First 20 years	After 20 years
Premium Rate	3P	P
Benefit	1,000,000	500,000

2. $\mu_{x+t} = 0.03, \quad t \geq 0$
3. $\delta = 0.06$

Calculate P using the equivalence principle.

```
life = ConstantForce(mu=0.03).set_interest(delta=0.06)
x = 0
payments = (3 * life.temporary_annuity(x, t=20, discrete=False)
            + life.deferred_annuity(x, u=20, discrete=False))
benefits = (1000000 * life.term_insurance(x, t=20, discrete=False)
            + 500000 * life.deferred_insurance(x, u=20, discrete=False))
P = benefits / payments
isclose(10310, P, question="Q6.27")
```

```
----- Q6.27 10310: 10309.617799001708 [OK] -----
```

```
True
```

SOA Question 6.28 : (B) 36

```
life = SULT().set_interest(i=0.05)
a = life.temporary_annuity(40, t=5)
A = life.whole_life_insurance(40)
P = life.gross_premium(a=a, A=A, benefit=1000,
                       initial_policy=10, renewal_premium=.05,
                       renewal_policy=5, initial_premium=.2)
isclose(36, P, question="Q6.28")
```

```
----- Q6.28 36: 35.72634219391481 [OK] -----
```

```
True
```

SOA Question 6.29 : (B) 20.5

(35) purchases a fully discrete whole life insurance policy of 100,000. You are given:

1. The annual gross premium, calculated using the equivalence principle, is 1770
2. The expenses in policy year 1 are 50% of premium and 200 per policy
3. The expenses in policy years 2 and later are 10% of premium and 50 per policy
4. All expenses are incurred at the beginning of the policy year
5. $i = 0.035$

Calculate $\ddot{a}_{35}$.

```
life = Premiums().set_interest(i=0.035)
def fun(a):
    return life.gross_premium(A=life.insurance_twin(a=a), a=a,
                              initial_policy=200, initial_premium=.5,
                              renewal_policy=50, renewal_premium=.1,
                              benefit=100000)
a = life.solve(fun, target=1770, grid=[20, 22])
isclose(20.5, a, question="Q6.29")
```

```
----- Q6.29 20.5: 20.480268314431726 [OK] -----
```

```
True
```

SOA Question 6.30 : (A) 900

For a fully discrete whole life insurance of 100 on (x), you are given:

1. The first year expense is 10% of the gross annual premium
2. Expenses in subsequent years are 5% of the gross annual premium
3. The gross premium calculated using the equivalence principle is 2.338
4. $i = 0.04$
5. $\ddot{a}_x = 16.50$
6. ${}^2A_x = 0.17$

Calculate the variance of the loss at issue random variable.

```
life = PolicyValues().set_interest(i=0.04)
contract = Contract(premium=2.338,
                    benefit=100,
                    initial_premium=.1,
                    renewal_premium=0.05)
var = life.gross_variance_loss(A1=life.insurance_twin(16.50),
                               A2=0.17, contract=contract)
isclose(900, var, question="Q6.30")
```

```
----- Q6.30 900: 908.141412994607 [OK] -----
```

```
True
```

SOA Question 6.31 : (D) 1330

For a fully continuous whole life insurance policy of 100,000 on (35), you are given:

1. The density function of the future lifetime of a newborn: $$
2. $\delta = 0.05$
3. $\overline{A}_{70} = 0.51791$

Calculate the annual net premium rate for this policy.

```
life = ConstantForce(mu=0.01).set_interest(delta=0.05)
A = (life.term_insurance(35, t=35, discrete=False)
     + life.E_x(35, t=35)*0.51791)      # A_35
P = life.premium_equivalence(A=A, b=100000, discrete=False)
isclose(1330, P, question="Q6.31")
```

```
----- Q6.31 1330: 1326.5406293909457 [OK] -----
```

```
True
```

SOA Question 6.32 : (C) 550

For a whole life insurance of 100,000 on (x), you are given:

1. Death benefits are payable at the moment of death
2. Deaths are uniformly distributed over each year of age
3. Premiums are payable monthly
4. $i = 0.05$
5. $\ddot{a}_x = 9.19$

Calculate the monthly net premium.

```
x = 0
life = Recursion().set_interest(i=0.05).set_a(9.19, x=x)
benefits = UDD(m=0, life=life).whole_life_insurance(x)
payments = UDD(m=12, life=life).whole_life_annuity(x)
P = life.gross_premium(a=payments, A=benefits, benefit=100000)/12
isclose(550, P, question="Q6.32")
```

Whole Life Insurance A_x :=

$\ddot{a}_x = [1 - A_x]/d$ annuity twin

```
----- Q6.32 550: 550.4356936711871 [OK] -----
```

```
True
```

SOA Question 6.33 : (B) 0.13

An insurance company sells 15-year pure endowments of 10,000 to 500 lives, each age x, with independent future lifetimes. The single premium for each pure endowment is determined by the equivalence principle.

You are given:

1. $i = 0.03$
2. $\mu_x(t) = 0.02t, \quad t \geq 0$
3. ${}_0L$ is the aggregate loss at issue random variable for these pure endowments.

Using the normal approximation without continuity correction, calculate $Pr({}_0L) > 50,000)$.

```
life = Insurance().set_survival(mu=lambda x,t: 0.02*t).set_interest(i=0.03)
x = 0
var = life.E_x(x, t=15, moment=life.VARIANCE, endowment=10000)
p = 1- life.portfolio_cdf(mean=0, variance=var, value=50000, N=500)
isclose(0.13, p, question="Q6.33", rel_tol=0.02)
```

```
----- Q6.33 0.13: 0.12828940905648634 [OK] -----
```

```
True
```

SOA Question 6.34 : (A) 23300

For a fully discrete whole life insurance policy on (61), you are given:

1. The annual gross premium using the equivalence principle is 500
2. Initial expenses, incurred at policy issue, are 15% of the premium
3. Renewal expenses, incurred at the beginning of each year after the first, are 3% of the premium
4. Mortality follows the Standard Ultimate Life Table
5. $i = 0.05$

Calculate the amount of the death benefit.

```
life = SULT()
def fun(benefit):
    A = life.whole_life_insurance(61)
    a = life.whole_life_annuity(61)
    return life.gross_premium(A=A, a=a, benefit=benefit,
                              initial_premium=0.15, renewal_premium=0.03)
b = life.solve(fun, target=500, grid=[23300, 23700])
isclose(23300, b, question="Q6.34")
```

```
----- Q6.34 23300: 23294.288659265632 [OK] -----
```

```
True
```

SOA Question 6.35 : (D) 530

For a fully discrete whole life insurance policy of 100,000 on (35), you are given:

1. First year commissions are 19% of the annual gross premium
2. Renewal year commissions are 4% of the annual gross premium
3. Mortality follows the Standard Ultimate Life Table
4. $i = 0.05$

Calculate the annual gross premium for this policy using the equivalence principle.

```
sult = SULT()
A = sult.whole_life_insurance(35, b=100000)
a = sult.whole_life_annuity(35)
P = sult.gross_premium(a=a, A=A, initial_premium=.19, renewal_premium=.04)
isclose(530, P, question="Q6.35")
```

```
----- Q6.35 530: 534.4072234303344 [OK] -----
```

```
True
```

SOA Question 6.36 : (B) 500

```
life = ConstantForce(mu=0.04).set_interest(delta=0.08)
a = life.temporary_annuity(50, t=20, discrete=False)
A = life.term_insurance(50, t=20, discrete=False)
def fun(R):
    return life.gross_premium(a=a, A=A, initial_premium=R/4500,
                              renewal_premium=R/4500, benefit=100000)
R = life.solve(fun, target=4500, grid=[400, 800])
isclose(500, R, question="Q6.36")
```

```
----- Q6.36 500: 500.0 [OK] -----
```

```
True
```

SOA Question 6.37 : (D) 820

For a fully discrete whole life insurance policy of 50,000 on (35), with premiums payable for a maximum of 10 years, you are given:

1. Expenses of 100 are payable at the end of each year including the year of death
2. Mortality follows the Standard Ultimate Life Table
3. $i = 0.05$

Calculate the annual gross premium using the equivalence principle.

```
sult = SULT()
benefits = sult.whole_life_insurance(35, b=50000 + 100)
expenses = sult.immediate_annuity(35, b=100)
a = sult.temporary_annuity(35, t=10)
P = (benefits + expenses) / a
isclose(820, P, question="Q6.37")
```

```
----- Q6.37 820: 819.7190338249138 [OK] -----
```

```
True
```

SOA Question 6.38 : (B) 11.3

For an n-year endowment insurance of 1000 on (x), you are given:

1. Death benefits are payable at the moment of death
2. Premiums are payable annually at the beginning of each year
3. Deaths are uniformly distributed over each year of age
4. $i = 0.05$
5. ${}_nE_x = 0.172$

6. $\overline{A}_{x:\overline{n|}} = 0.192$

Calculate the annual net premium for this insurance.

```
x, n = 0, 10
life = Recursion().set_interest(i=0.05)\
                  .set_A(0.192, x=x, t=n, endowment=1, discrete=False)\
                  .set_E(0.172, x=x, t=n)
a = life.temporary_annuity(x, t=n, discrete=False)

def fun(a):   # solve for discrete annuity, given continuous
    life = Recursion(verbose=False).set_interest(i=0.05)\
                                   .set_a(a, x=x, t=n)\
                                   .set_E(0.172, x=x, t=n)
    return UDD(m=0, life=life).temporary_annuity(x, t=n)
a = life.solve(fun, target=a, grid=a)  # discrete annuity
P = life.gross_premium(a=a, A=0.192, benefit=1000)
isclose(11.3, P, question="Q6.38")
```

$$\texttt{Temporary Annuity } a_{x:\overline{10|}} := \ddot{a}_{x:\overline{10|}} = [1 - A_{x:\overline{10|}}]/d \qquad \texttt{annuity twin}$$

```
----- Q6.38 11.3: 11.308644185253657 [OK] -----
```

```
True
```

SOA Question 6.39 : (A) 29

XYZ Insurance writes 10,000 fully discrete whole life insurance policies of 1000 on lives age 40 and an additional 10,000 fully discrete whole life policies of 1000 on lives age 80.

XYZ used the following assumptions to determine the net premiums for these policies:

1. Mortality follows the Standard Ultimate Life Table
2. i = 0.05

During the first ten years, mortality did follow the Standard Ultimate Life Table.

Calculate the average net premium per policy in force received at the beginning of the eleventh year.

```
sult = SULT()
P40 = sult.premium_equivalence(sult.whole_life_insurance(40), b=1000)
P80 = sult.premium_equivalence(sult.whole_life_insurance(80), b=1000)
p40 = sult.p_x(40, t=10)
p80 = sult.p_x(80, t=10)
P = (P40 * p40 + P80 * p80) / (p80 + p40)
isclose(29, P, question="Q6.39")
```

```
----- Q6.39 29: 29.033866427845496 [OK] -----
```

```
True
```

SOA Question 6.40 : (C) 116

For a special fully discrete whole life insurance, you are given:

1. The death benefit is $1000(1.03)^k$ for death in policy year k, for $k = 1, 2, 3...$
2. $q_x = 0.05$
3. $i = 0.06$
4. $\ddot{a}_{x+1} = 7.00$
5. The annual net premium for this insurance at issue age x is 110

Calculate the annual net premium for this insurance at issue age $x + 1$.

```
# - standard formula discounts/accumulates by too much (i should be smaller)
x = 0
life = Recursion().set_interest(i=0.06).set_a(7, x=x+1).set_q(0.05, x=x)
a = life.whole_life_annuity(x)
A = 110 * a / 1000
life = Recursion().set_interest(i=0.06).set_A(A, x=x).set_q(0.05, x=x)
A1 = life.whole_life_insurance(x+1)
P = life.gross_premium(A=A1 / 1.03, a=7) * 1000
isclose(116, P, question="Q6.40")
```

Whole Life Annuity $\ddot{a}_x$:=
$\ddot{a}_x = 1 + E_x * \ddot{a}_{x+1}$ backward recursion
$E_x = p_x * v$ pure endowment
$p_x = 1 - q_x$ complement of mortality

Whole Life Insurance A_{x+1} :=
$A_{x+1} = [A_x/v - q_x * b]/ p_x$ forward recursion
$p_x = 1 - q_x$ complement of mortality

```
----- Q6.40 116: 116.51945397474269 [OK] -----
```

```
True
```

SOA Question 6.41 : (B) 1417

For a special fully discrete 2-year term insurance on (x), you are given:

1. $q_x = 0.01$
2. $q_{x+1} = 0.02$
3. $i = 0.05$
4. The death benefit in the first year is 100,000
5. Both the benefits and premiums increase by 1% in the second year

Calculate the annual net premium in the first year.

```
x = 0
life = LifeTable().set_interest(i=0.05).set_table(q={x:.01, x+1:.02})
a = 1 + life.E_x(x, t=1) * 1.01
A = life.deferred_insurance(x, u=0, t=1) + 1.01*life.deferred_insurance(x, u=1, t=1)
P = 100000 * A / a
isclose(1417, P, question="Q6.41")
```

```
----- Q6.41 1417: 1416.9332301924137 [OK] -----
```

```
True
```

SOA Question 6.42 : (D) 0.113

```
x = 0
life = ConstantForce(mu=0.06).set_interest(delta=0.06)
contract = Contract(discrete=True, premium=315.8,
                    T=3, endowment=1000, benefit=1000)
L = [life.L_from_t(t, contract=contract) for t in range(3)]     # L(t)
Q = [life.q_x(x, u=u, t=1) for u in range(3)]               # prob(die in year t)
Q[-1] = 1 - sum(Q[:-1])    # follows SOA Solution: incorrectly treats endowment!
p = sum([q for (q, l) in zip (Q, L) if l > 0])
isclose(0.113, p, question="Q6.42")
```

```
----- Q6.42 0.113: 0.11307956328284252 [OK] -----
```

```
True
```

SOA Question 6.43 : (C) 170

For a fully discrete, 5-payment 10-year term insurance of 200,000 on (30), you are given:

1. Mortality follows the Standard Ultimate Life Table
2. The following expenses are incurred at the beginning of each respective year:

	Percent of Premium	Per Policy	Percent of Premium	Per Policy
	Year 1	Year 1	Years 2 - 10	Years 2 - 10
Taxes	5%	0	5%	0
Commissions	30%	0	10%	0
Maintenance	0%	8	0%	4

3. i = 0.05
4. $\ddot{a}_{30:\overline{5|}} = 4.5431$

Calculate the annual gross premium using the equivalence principle.

- although 10-year term, premiums only paid first first years: separately calculate the EPV of per-policy maintenance expenses in years 6-10 and treat as additional initial expense

```
sult = SULT()
a = sult.temporary_annuity(30, t=5)
A = sult.term_insurance(30, t=10)
other_expenses = 4 * sult.deferred_annuity(30, u=5, t=5)
P = sult.gross_premium(a=a, A=A, benefit=200000, initial_premium=0.35,
                       initial_policy=8 + other_expenses, renewal_policy=4,
                       renewal_premium=0.15)
isclose(170, P, question="Q6.43")
```

```
----- Q6.43 170: 171.22371939459944 [OK] -----
```

```
True
```

SOA Question 6.44 : (D) 2.18

```
life = Recursion().set_interest(i=0.05)\
                  .set_IA(0.15, x=50, t=10)\
                  .set_a(17, x=50)\
                  .set_a(15, x=60)\
                  .set_E(0.6, x=50, t=10)
A = life.deferred_insurance(50, u=10)
IA = life.increasing_insurance(50, t=10)
a = life.temporary_annuity(50, t=10)
P = life.gross_premium(a=a, A=A, IA=IA, benefit=100)
isclose(2.2, P, question="Q6.44")
```

Whole Life Insurance A_{x+60} :=
$\ddot{a}_{x+60} = [1 - A_{x+60}]/d$ annuity twin

Whole Life Insurance A_{x+60} :=
$\ddot{a}_{x+60} = [1 - A_{x+60}]/d$ annuity twin

Whole Life Insurance A_{x+60} :=
$\ddot{a}_{x+60} = [1 - A_{x+60}]/d$ annuity twin

```
----- Q6.44 2.2: 2.183803457688809 [OK] -----
```

```
True
```

SOA Question 6.45 : (E) 690

For a fully continuous whole life insurance of 100,000 on (35), you are given:

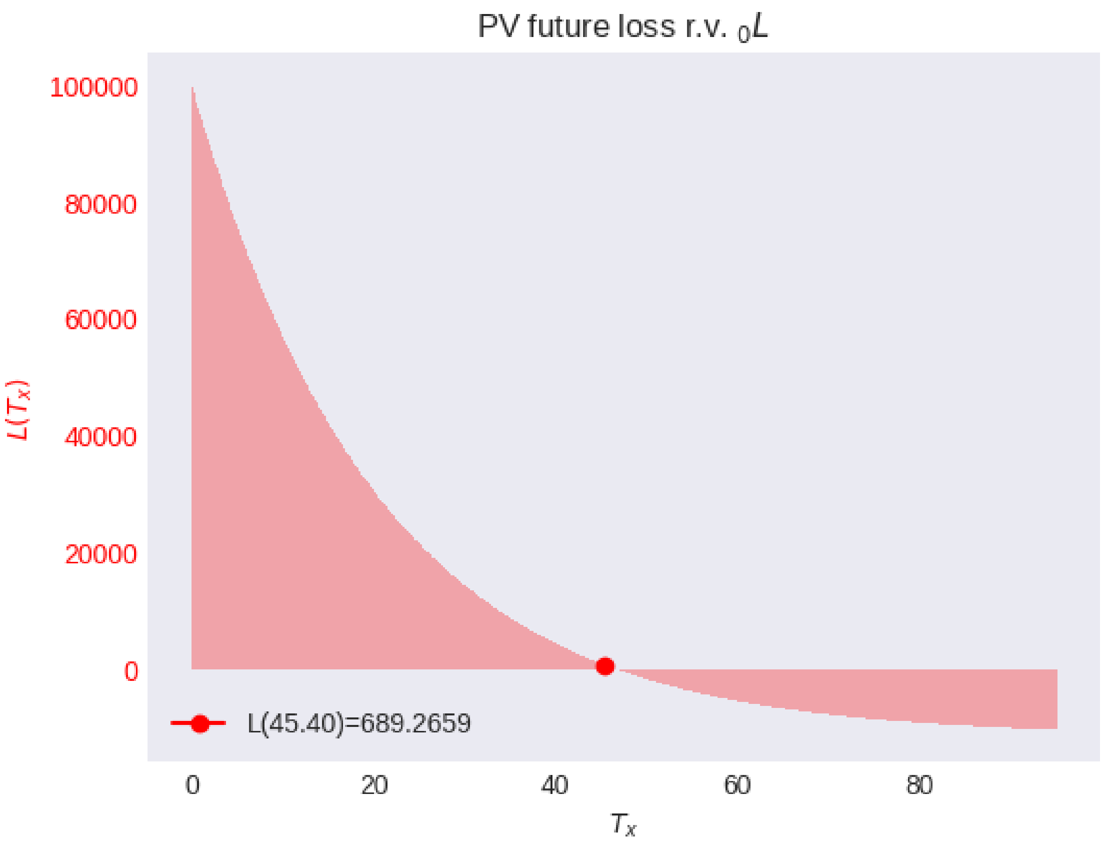
1. The annual rate of premium is 560
2. Mortality follows the Standard Ultimate Life Table
3. Deaths are uniformly distributed over each year of age
4. $i = 0.05$

Calculate the 75th percentile of the loss at issue random variable for this policy.

```
life = SULT(udd=True)
contract = Contract(benefit=100000, premium=560, discrete=False)
L = life.L_from_prob(x=35, prob=0.75, contract=contract)
life.L_plot(x=35, contract=contract,
            T=life.L_to_t(L=L, contract=contract))
isclose(690, L, question="Q6.45")
```

```
----- Q6.45 690: 689.2659416264196 [OK] -----
```

```
True
```



SOA Question 6.46 : (E) 208

```
life = Recursion().set_interest(i=0.05)\
                  .set_IA(0.51213, x=55, t=10)\
                  .set_a(12.2758, x=55)\
                  .set_a(7.4575, x=55, t=10)
A = life.deferred_annuity(55, u=10)
IA = life.increasing_insurance(55, t=10)
a = life.temporary_annuity(55, t=10)
P = life.gross_premium(a=a, A=A, IA=IA, benefit=300)
isclose(208, P, question="Q6.46")
```

```
----- Q6.46 208: 208.12282139036515 [OK] -----
```

```
True
```

SOA Question 6.47 : (D) 66400

For a 10-year deferred whole life annuity-due with payments of 100,000 per year on (70), you are given:

1. Annual gross premiums of G are payable for 10 years
2. First year expenses are 75% of premium
3. Renewal expenses for years 2 and later are 5% of premium during the premium paying period

4. Mortality follows the Standard Ultimate Life Table
5. i = 0.05

Calculate G using the equivalence principle.

```
sult = SULT()
a = sult.temporary_annuity(70, t=10)
A = sult.deferred_annuity(70, u=10)
P = sult.gross_premium(a=a, A=A, benefit=100000, initial_premium=0.75,
                        renewal_premium=0.05)
isclose(66400, P, question="Q6.47")
```

```
----- Q6.47 66400: 66384.13293704337 [OK] -----
```

```
True
```

SOA Question 6.48 : (A) 3195

For a special fully discrete 5-year deferred 3-year term insurance of 100,000 on (x) you are given:

1. There are two premium payments, each equal to P . The first is paid at the beginning of the first year and the second is paid at the end of the 5-year deferral period
2. $p_x = 0.95$
3. $q_{x+5} = 0.02$
4. $q_{x+6} = 0.03$
5. $q_{x+7} = 0.04$
6. $i = 0.06$

Calculate P using the equivalence principle.

```
x = 0
life = Recursion(depth=5).set_interest(i=0.06)\
                         .set_p(.95, x=x, t=5)\
                         .set_q(.02, x=x+5)\
                         .set_q(.03, x=x+6)\
                         .set_q(.04, x=x+7)
a = 1 + life.E_x(x, t=5)
A = life.deferred_insurance(x, u=5, t=3)
P = life.gross_premium(A=A, a=a, benefit=100000)
isclose(3195, P, question="Q6.48")
```

Pure Endowment ${}_5E_x$:=

${}_5E_x = {}_5p_x * v^5$ pure endowment

Pure Endowment ${}_5E_x$:=

${}_5E_x = {}_5p_x * v^5$ pure endowment

$$\begin{array}{rll}
\texttt{Term Insurance } A^{1}_{x+5:\overline{3|}} := & & \\
& A^{1}_{x+5:\overline{1|}} = A_{x+5:\overline{1|}} - E_{x+5} & \texttt{endowment insurance - pure} \\
& A_{x+5:\overline{3|}} = A^{1}_{x+5:\overline{3|}} + {}_{3}E_{x+5} & \texttt{term plus pure endowment} \\
& {}_{3}E_{x+5} = {}_{3}p_{x+5} * v^{3} & \texttt{pure endowment} \\
& {}_{3}p_{x+5} = {}_{2}p_{x+6} * p_{x+5} & \texttt{survival chain rule} \\
& {}_{2}p_{x+6} = p_{x+7} * p_{x+6} & \texttt{survival chain rule} \\
& A^{1}_{x+5:\overline{3|}} = A_{x+5:\overline{3|}} - {}_{3}E_{x+5} & \texttt{endowment insurance - pure} \\
& {}_{3}E_{x+5} = E_{x+5} * {}_{2}E_{x+6} & \texttt{pure endowment chain rule} \\
& {}_{2}E_{x+6} = E_{x+6} * E_{x+7} & \texttt{pure endowment chain rule} \\
& E_{x+7} = p_{x+7} * v & \texttt{pure endowment} \\
& E_{x+6} = p_{x+6} * v & \texttt{pure endowment} \\
& E_{x+5} = p_{x+5} * v & \texttt{pure endowment} \\
& p_{x+7} = 1 - q_{x+7} & \texttt{complement of mortality} \\
& A^{1}_{x+5:\overline{3|}} = v * [\, q_{x+5} * b + p_{x+5} * A^{1}_{x+6:\overline{2|}}] & \texttt{backward recursion} \\
& A^{1}_{x+6:\overline{2|}} = v * [\, q_{x+6} * b + p_{x+6} * A^{1}_{x+7:\overline{1|}}] & \texttt{backward recursion} \\
& p_{x+6} = 1 - q_{x+6} & \texttt{complement of mortality} \\
& p_{x+5} = 1 - q_{x+5} & \texttt{complement of mortality}
\end{array}$$

$$\begin{array}{rll}
\texttt{Term Insurance } A^{1}_{x+5:\overline{3|}} := & & \\
& A^{1}_{x+5:\overline{1|}} = A_{x+5:\overline{1|}} - E_{x+5} & \texttt{endowment insurance - pure} \\
& A_{x+5:\overline{3|}} = A^{1}_{x+5:\overline{3|}} + {}_{3}E_{x+5} & \texttt{term plus pure endowment} \\
& {}_{3}E_{x+5} = {}_{3}p_{x+5} * v^{3} & \texttt{pure endowment} \\
& {}_{3}p_{x+5} = {}_{2}p_{x+6} * p_{x+5} & \texttt{survival chain rule} \\
& {}_{2}p_{x+6} = p_{x+7} * p_{x+6} & \texttt{survival chain rule} \\
& A^{1}_{x+5:\overline{3|}} = A_{x+5:\overline{3|}} - {}_{3}E_{x+5} & \texttt{endowment insurance - pure} \\
& {}_{3}E_{x+5} = E_{x+5} * {}_{2}E_{x+6} & \texttt{pure endowment chain rule} \\
& {}_{2}E_{x+6} = E_{x+6} * E_{x+7} & \texttt{pure endowment chain rule} \\
& E_{x+7} = p_{x+7} * v & \texttt{pure endowment} \\
& E_{x+6} = p_{x+6} * v & \texttt{pure endowment} \\
& E_{x+5} = p_{x+5} * v & \texttt{pure endowment} \\
& p_{x+7} = 1 - q_{x+7} & \texttt{complement of mortality} \\
& A^{1}_{x+5:\overline{3|}} = v * [\, q_{x+5} * b + p_{x+5} * A^{1}_{x+6:\overline{2|}}] & \texttt{backward recursion} \\
& A^{1}_{x+6:\overline{2|}} = v * [\, q_{x+6} * b + p_{x+6} * A^{1}_{x+7:\overline{1|}}] & \texttt{backward recursion} \\
& p_{x+6} = 1 - q_{x+6} & \texttt{complement of mortality} \\
& p_{x+5} = 1 - q_{x+5} & \texttt{complement of mortality}
\end{array}$$

```
----- Q6.48 3195: 3195.118917658744 [OK] -----
```

```
True
```

SOA Question 6.49 : (C) 86

For a special whole life insurance of 100,000 on (40), you are given:

1. The death benefit is payable at the moment of death
2. Level gross premiums are payable monthly for a maximum of 20 years
3. Mortality follows the Standard Ultimate Life Table
4. $i = 0.05$
5. Deaths are uniformly distributed over each year of age
6. Initial expenses are 200

7. Renewal expenses are 4% of each premium including the first
8. Gross premiums are calculated using the equivalence principle

Calculate the monthly gross premium.

```
sult = SULT(udd=True)
a = UDD(m=12, life=sult).temporary_annuity(40, t=20)
A = sult.whole_life_insurance(40, discrete=False)
P = sult.gross_premium(a=a, A=A, benefit=100000, initial_policy=200,
                        renewal_premium=0.04, initial_premium=0.04) / 12
isclose(86, P, question="Q6.49")
```

```
----- Q6.49 86: 85.99177833261696 [OK] -----
```

```
True
```

SOA Question 6.50 : (A) -47000

On July 15, 2017, XYZ Corp buys fully discrete whole life insurance policies of 1,000 on each of its 10,000 workers, all age 35. It uses the death benefits to partially pay the premiums for the following year.

You are given:

1. Mortality follows the Standard Ultimate Life Table
2. $i = 0.05$
3. The insurance is priced using the equivalence principle

Calculate XYZ Corp's expected net cash flow from these policies during July 2018.

```
life = SULT()
P = life.premium_equivalence(a=life.whole_life_annuity(35), b=1000)
a = life.deferred_annuity(35, u=1, t=1)
A = life.term_insurance(35, t=1, b=1000)
cash = (A - a * P) * 10000 / life.interest.v
isclose(-47000, cash, question="Q6.50")
```

```
----- Q6.50 -47000: -46948.2187697819 [OK] -----
```

```
True
```

SOA Question 6.51 : (D) 34700

```
life = Recursion().set_DA(0.4891, x=62, t=10)\
                  .set_A(0.0910, x=62, t=10)\
                  .set_a(12.2758, x=62)\
                  .set_a(7.4574, x=62, t=10)
IA = life.increasing_insurance(62, t=10)
A = life.deferred_annuity(62, u=10)
a = life.temporary_annuity(62, t=10)
P = life.gross_premium(a=a, A=A, IA=IA, benefit=50000)
isclose(34700, P, question="Q6.51")
```

Increasing Insurance $(IA)_{x+62:\overline{10|}}$:=
$(IA)_{x+62:\overline{10|}} = 11\ A^{1}_{x+62:\overline{10|}} - (DA)_{x+62:\overline{10|}}$ varying insurance identity

```
----- Q6.51 34700: 34687.207544453246 [OK] -----
```

```
True
```

SOA Question 6.52 : (D) 50.80

For a fully discrete 10-payment whole life insurance of H on (45), you are given:

1. Expenses payable at the beginning of each year are as follows:

Expense Type	First Year	Years 2-10	Years 11+
Per policy	100	20	10
% of Premium	105%	5%	0%

2. Mortality follows the Standard Ultimate Life Table
3. i = 0.05
4. The gross annual premium, calculated using the equivalence principle, is of the form $G = gH + f$, where g is the premium rate per 1 of insurance and f is the per policy fee

Calculate f.

- set face value benefits to 0

```
sult = SULT()
a = sult.temporary_annuity(45, t=10)
other_cost = 10 * sult.deferred_annuity(45, u=10)
P = sult.gross_premium(a=a, A=0, benefit=0,     # set face value H = 0
                       initial_premium=1.05, renewal_premium=0.05,
                       initial_policy=100 + other_cost, renewal_policy=20)
isclose(50.8, P, question="Q6.52")
```

```
----- Q6.52 50.8: 50.80135534704229 [OK] -----
```

```
True
```

SOA Question 6.53 : (D) 720

A warranty pays 2000 at the end of the year of the first failure if a washing machine fails within three years of purchase. The warranty is purchased with a single premium, G, paid at the time of purchase of the washing machine. You are given:

1. 10% of the washing machines that are working at the start of each year fail by the end of that year
2. *i* = 0.08
3. The sales commission is 35% of G
4. G is calculated using the equivalence principle

Calculate G.

```
x = 0
life = LifeTable().set_interest(i=0.08).set_table(q={x:.1, x+1:.1, x+2:.1})
A = life.term_insurance(x, t=3)
P = life.gross_premium(a=1, A=A, benefit=2000, initial_premium=0.35)
isclose(720, P, question="Q6.53")
```

```
----- Q6.53 720: 720.1646090534978 [OK] -----
```

```
True
```

SOA Question 6.54 : (A) 25440

For a fully discrete whole life insurance of 200,000 on (45), you are given:

1. Mortality follows the Standard Ultimate Life Table.
2. $i = 0.05$
3. The annual premium is determined using the equivalence principle.

Calculate the standard deviation of ${}_0L$, the present value random variable for the loss at issue.

[A modified version of Question 12 on the Fall 2017 exam]

```
life = SULT()
std = math.sqrt(life.net_policy_variance(45, b=200000))
isclose(25440, std, question="Q6.54")
```

```
----- Q6.54 25440: 25441.694847703857 [OK] -----
```

```
True
```

22.7 7 Policy Values

SOA Question 7.1 : (C) 11150

For a special fully discrete whole life insurance on (40), you are given:

1. The death benefit is 50,000 in the first 20 years and 100,000 thereafter
2. Level net premiums of 875 are payable for 20 years
3. Mortality follows the Standard Ultimate Life Table
4. $i = 0.05$

Calculate ${}_{10}V$ the net premium policy value at the end of year 10 for this insurance.

```
life = SULT()
x, n, t = 40, 20, 10
A = (life.whole_life_insurance(x+t, b=50000)
     + life.deferred_insurance(x+t, u=n-t, b=50000))
a = life.temporary_annuity(x+t, t=n-t, b=875)
L = life.gross_future_loss(A=A, a=a)
isclose(11150, L, question="Q7.1")
```

```
----- Q7.1 11150: 11152.108749338717 [OK] -----
```

```
True
```

SOA Question 7.2 : (C) 1152

```
x = 0
life = Recursion(verbose=False).set_interest(i=.1)\
                                .set_q(0.15, x=x)\
                                .set_q(0.165, x=x+1)\
                                .set_reserves(T=2, endowment=2000)

def fun(P):  # solve P s.t. V is equal backwards and forwards
    policy = dict(t=1, premium=P, benefit=lambda t: 2000, reserve_benefit=True)
    return life.t_V_backward(x, **policy) - life.t_V_forward(x, **policy)
P = life.solve(fun, target=0, grid=[1070, 1230])
isclose(1152, P, question="Q7.2")
```

```
----- Q7.2 1152: 1151.5151515151515 [OK] -----
```

```
True
```

SOA Question 7.3 : (E) 730

```
x = 0  # x=0 is (90) and interpret every 3 months as t=1 year
life = LifeTable().set_interest(i=0.08/4)\
                  .set_table(l={0:1000, 1:898, 2:800, 3:706})\
                  .set_reserves(T=8, V={3: 753.72})
V = life.t_V_backward(x=0, t=2, premium=60*0.9, benefit=lambda t: 1000)
V = life.set_reserves(V={2: V})\
        .t_V_backward(x=0, t=1, premium=0, benefit=lambda t: 1000)
isclose(730, V, question="Q7.3")
```

```
----- Q7.3 730: 729.998398765594 [OK] -----
```

```
True
```

SOA Question 7.4 : (B) -74

For a special fully discrete whole life insurance on (40), you are given:

1. The death benefit is 1000 during the first 11 years and 5000 thereafter
2. Expenses, payable at the beginning of the year, are 100 in year 1 and 10 in years 2 and later
3. π is the level annual premium, determined using the equivalence principle
4. $G = 1.02 \times \pi$ is the level annual gross premium
5. Mortality follows the Standard Ultimate Life Table
6. $i = 0.05$
7. ${}_{11}E_{40} = 0.57949$

Calculate the gross premium policy value at the end of year 1 for this insurance.

hints:

- split benefits into two policies

```
life = SULT()
P = life.gross_premium(a=life.whole_life_annuity(40),
                       A=life.whole_life_insurance(40),
                       initial_policy=100, renewal_policy=10,
                       benefit=1000)
P += life.gross_premium(a=life.whole_life_annuity(40),
                        A=life.deferred_insurance(40, u=11),
                        benefit=4000)    # for deferred portion
contract = Contract(benefit=1000, premium=1.02*P,
                    renewal_policy=10, initial_policy=100)
V = life.gross_policy_value(x=40, t=1, contract=contract)
contract = Contract(benefit=4000, premium=0)
A = life.deferred_insurance(41, u=10)
V += life.gross_future_loss(A=A, a=0, contract=contract)    # for deferred portion
isclose(-74, V, question="Q7.4")
```

```
----- Q7.4 -74: -73.942155695248 [OK] -----
```

```
True
```

SOA Question 7.5 : (E) 1900

For a fully discrete whole life insurance of 10,000 on (x), you are given:

1. Deaths are uniformly distributed over each year of age
2. The net premium is 647.46
3. The net premium policy value at the end of year 4 is 1405.08
4. q_{x+4} = 0.04561
5. i = 0.03

Calculate the net premium policy value at the end of 4.5 years.

```
x = 0
life = Recursion(udd=True).set_interest(i=0.03)\
                          .set_q(0.04561, x=x+4)\
                          .set_reserves(T=3, V={4: 1405.08})
V = life.r_V_forward(x, s=4, r=0.5, benefit=10000, premium=647.46)
isclose(1900, V, question="Q7.5")
```

```
----- Q7.5 1900: 1901.766021537228 [OK] -----
```

```
True
```

Answer 7.6: (E) -25.4

```
life = SULT()
P = life.net_premium(45, b=2000)
contract = Contract(benefit=2000, initial_premium=.25, renewal_premium=.05,
                    initial_policy=2*1.5 + 30, renewal_policy=2*.5 + 10)
```

(continues on next page)

(continued from previous page)

```
G = life.gross_premium(a=life.whole_life_annuity(45), **contract.premium_terms)
gross = life.gross_policy_value(45, t=10, contract=contract.set_contract(premium=G))
net = life.net_policy_value(45, t=10, b=2000)
V = gross - net
isclose(-25.4, V, question="Q7.6")
```

```
----- Q7.6 -25.4: -25.44920289521204 [OK] -----
```

```
True
```

SOA Question 7.7 : (D) 1110

For a whole life insurance of 10,000 on (x), you are given:

1. Death benefits are payable at the end of the year of death
2. A premium of 30 is payable at the start of each month
3. Commissions are 5% of each premium
4. Expenses of 100 are payable at the start of each year
5. $i = 0.05$
6. $1000A_{x+10} = 400$
7. ${}_{10}V$ is the gross premium policy value at the end of year 10 for this insurance

Calculate ${}_{10}V$ using the two-term Woolhouse formula for annuities.

```
x = 0
life = Recursion().set_interest(i=0.05).set_A(0.4, x=x+10)
a = Woolhouse(m=12, life=life).whole_life_annuity(x+10)
contract = Contract(premium=0, benefit=10000, renewal_policy=100)
V = life.gross_future_loss(A=0.4, contract=contract.renewals())
contract = Contract(premium=30*12, renewal_premium=0.05)
V += life.gross_future_loss(a=a, contract=contract.renewals())
isclose(1110, V, question="Q7.7")
```

Whole Life Annuity $\ddot{a}_{x+10}$:=	
$\ddot{a}_{x+10} = [1 - A_{x+10}]/d$	insurance twin

```
----- Q7.7 1110: 1107.9718253968258 [OK] -----
```

```
True
```

SOA Question 7.8 : (C) 29.85

```
sult = SULT()
x = 70
q = {x: [sult.q_x(x+k)*(.7 + .1*k) for k in range(3)] + [sult.q_x(x+3)]}
life = Recursion().set_interest(i=.05)\
                  .set_q(sult.q_x(70)*.7, x=x)\
```

(continues on next page)

(continued from previous page)

```
                  .set_reserves(T=3)
V = life.t_V(x=70, t=1, premium=35.168, benefit=lambda t: 1000)
isclose(29.85, V, question="Q7.8")
```

Survival p_{x+70} :=

$p_{x+70} = 1 - q_{x+70}$ complement of mortality

```
----- Q7.8 29.85: 29.85469179271202 [OK] -----
```

```
True
```

SOA Question 7.9 : (A) 38100

For a semi-continuous 20-year endowment insurance of 100,000 on (45), you are given:

1. Net premiums of 253 are payable monthly
2. Mortality follows the Standard Ultimate Life Table
3. Deaths are uniformly distributed over each year of age
4. $i = 0.05$

Calculate ${}_{10}V$, the net premium policy value at the end of year 10 for this insurance.

```
sult = SULT(udd=True)
x, n, t = 45, 20, 10
a = UDD(m=12, life=sult).temporary_annuity(x=x+10, t=n-t)
A = UDD(m=0, life=sult).endowment_insurance(x=x+10, t=n-t)
contract = Contract(premium=253*12, endowment=100000, benefit=100000)
V = sult.gross_future_loss(A=A, a=a, contract=contract)
isclose(38100, V, question="Q7.9")
```

```
----- Q7.9 38100: 38099.62176709246 [OK] -----
```

```
True
```

SOA Question 7.10 : (C) -970

For a fully discrete whole life insurance of 100,000 on (45), you are given:

1. Mortality follows the Standard Ultimate Life Table
2. $i = 0.05$
3. Commission expenses are 60% of the first year's gross premium and 2% of renewal gross premiums
4. Administrative expenses are 500 in the first year and 50 in each renewal year
5. All expenses are payable at the start of the year
6. The gross premium, calculated using the equivalence principle, is 977.60

Calculate ${}_{5}V^{e}$, the expense reserve at the end of year 5 for this insurance.

```
life = SULT()
G = 977.6
P = life.net_premium(45, b=100000)
contract = Contract(benefit=0, premium=G-P, renewal_policy=.02*G + 50)
V = life.gross_policy_value(45, t=5, contract=contract)
isclose(-970, V, question="Q7.10")
```

```
----- Q7.10 -970: -971.8909301877826 [OK] -----
```

```
True
```

SOA Question 7.11 : (B) 1460

```
life = Recursion().set_interest(i=0.05).set_a(13.4205, x=55)
contract = Contract(benefit=10000)
def fun(P):
    return life.L_from_t(t=10, contract=contract.set_contract(premium=P))
P = life.solve(fun, target=4450, grid=400)
V = life.gross_policy_value(45, t=10, contract=contract.set_contract(premium=P))
isclose(1460, V, question="Q7.11")
```

Whole Life Insurance A_{x+55} :=
$\ddot{a}_{x+55} = [1 - A_{x+55}]/d$ annuity twin

```
----- Q7.11 1460: 1459.9818035330218 [OK] -----
```

```
True
```

SOA Question 7.12 : (E) 4.09

For a special fully discrete 25-year endowment insurance on (44), you are given:

1. The death benefit is (26−k) for death in year ,k for k = 1,2,3,…,25
2. The endowment benefit in year 25 is 1
3. Net premiums are level
4. q_{55}= 0.15
5. i = 0.04
6. ${}_{11}V$ the net premium policy value at the end of year 11, is 5.00
7. ${}_{24}V$ the net premium policy value at the end of year 24, is 0.60

Calculate ${}_{12}V$ the net premium policy value at end of year 12.

```
benefit = lambda k: 26 - k
x = 44
life = Recursion().set_interest(i=0.04)\
                  .set_q(0.15, x=55)\
                  .set_reserves(T=25, endowment=1, V={11: 5.})
def fun(P):  # solve for net premium, from final year recursion
```

(continues on next page)

(continued from previous page)

```
    return life.t_V(x=x, t=24, premium=P, benefit=benefit)
P = life.solve(fun, target=0.6, grid=0.5)    # solved net premium
V = life.t_V(x, t=12, premium=P, benefit=benefit)  # recursion formula
isclose(4.09, V, question="Q7.12")
```

Survival p_{x+55} :=

$p_{x+55} = 1 - q_{x+55}$ complement of mortality

```
----- Q7.12 4.09: 4.089411764705883 [OK] -----
```

```
True
```

SOA Question 7.13 : (A) 180

```
life = SULT()
V = life.FPT_policy_value(40, t=10, n=30, endowment=1000, b=1000)
isclose(180, V, question="Q7.13")
```

```
----- Q7.13 180: 180.1071785904076 [OK] -----
```

```
True
```

SOA Question 7.14 : (A) 2200

For a fully discrete whole life insurance of 100,000 on (45), you are given:

1. The gross premium policy value at duration 5 is 5500 and at duration 6 is 7100
2. $q_{50} = 0.009$
3. $i = 0.05$
4. Renewal expenses at the start of each year are 50 plus 4% of the gross premium.
5. Claim expenses are 200.

Calculate the gross premium.

```
x = 45
life = Recursion(verbose=False).set_interest(i=0.05)\
                               .set_q(0.009, x=50)\
                               .set_reserves(T=10, V={5: 5500})
def fun(P):  # solve for net premium,
    return life.t_V(x=x, t=6, premium=P*0.96 - 50, benefit=lambda t: 100000+200)
P = life.solve(fun, target=7100, grid=[2200, 2400])
isclose(2200, P, question="Q7.14")
```

```
----- Q7.14 2200: 2197.8174603174602 [OK] -----
```

```
True
```

SOA Question 7.15 : (E) 50.91

For a fully discrete whole life insurance of 100 on (x), you are given:

1. $q_{x+15} = 0.10$
2. Deaths are uniformly distributed over each year of age
3. $i = 0.05$
4. ${}_tV$ denotes the net premium policy value at time t
5. ${}_{16}V = 49.78$

Calculate 15.6.

```
x = 0
V = Recursion(udd=True).set_interest(i=0.05)\
                        .set_q(0.1, x=x+15)\
                        .set_reserves(T=3, V={16: 49.78})\
                        .r_V_backward(x, s=15, r=0.6, benefit=100)
isclose(50.91, V, question="Q7.15")
```

```
----- Q7.15 50.91: 50.91362826922369 [OK] -----
```

```
True
```

SOA Question 7.16 : (D) 380

```
life = SelectLife().set_interest(v=.95)\
                   .set_table(A={86: [683/1000]},
                              q={80+k: [.01*(k+1)] for k in range(6)})
x, t, n = 80, 3, 5
A = life.whole_life_insurance(x+t)
a = life.temporary_annuity(x+t, t=n-t)
V = life.gross_future_loss(A=A, a=a, contract=Contract(benefit=1000, premium=130))
isclose(380, V, question="Q7.16")
```

```
----- Q7.16 380: 381.6876905200001 [OK] -----
```

```
True
```

SOA Question 7.17 : (D) 1.018

```
x = 0
life = Recursion().set_interest(v=math.sqrt(0.90703))\
                  .set_q(0.02067, x=x+10)\
                  .set_A(0.52536, x=x+11)\
                  .set_A(0.30783, x=x+11, moment=2)
A1 = life.whole_life_insurance(x+10)
A2 = life.whole_life_insurance(x+10, moment=2)
ratio = (life.insurance_variance(A2=A2, A1=A1)
         / life.insurance_variance(A2=0.30783, A1=0.52536))
isclose(1.018, ratio, question="Q7.17")
```

`Whole Life Insurance` A_{x+10} $:=$

$A_{x+10} = v * [\, q_{x+10} * b + \, p_{x+10} * \, A_{x+11}]$ `backward recursion`

$p_{x+10} = 1 - \, q_{x+10}$ `complement of mortality`

`Whole Life Insurance` ${}^2A_{x+10}$ $:=$

${}^2A_{x+10} = v^2 * [\, q_{x+10} * b^2 + \, p_{x+10} * \, {}^2A_{x+11}]$ `backward recursion`

$p_{x+10} = 1 - \, q_{x+10}$ `complement of mortality`

```
----- Q7.17 1.018: 1.0182465434445054 [OK] -----
```

```
True
```

SOA Question 7.18 : (A) 17.1

For a fully discrete whole life insurance of 1 on (*x*), you are given:

1. The net premium policy value at the end of the first year is 0.012
2. q_x = 0.009
3. *i* = 0.04

Calculate $\ddot{a}_x$

```
x = 10
life = Recursion(verbose=False).set_interest(i=0.04).set_q(0.009, x=x)
def fun(a):
    return life.set_a(a, x=x).net_policy_value(x, t=1)
a = life.solve(fun, target=0.012, grid=[17.1, 19.1])
isclose(17.1, a, question="Q7.18")
```

```
----- Q7.18 17.1: 17.07941929974385 [OK] -----
```

```
True
```

SOA Question 7.19 : (D) 720

For a fully discrete whole life insurance of 100,000 on (40) you are given:

1. Expenses incurred at the beginning of the first year are 300 plus 50% of the first year premium
2. Renewal expenses, incurred at the beginning of the year, are 10% of each of the renewal premiums
3. Mortality follows the Standard Ultimate Life Table
4. *i* = 0.05
5. Gross premiums are calculated using the equivalence principle

Calculate the gross premium policy value for this insurance immediately after the second premium and associated renewal expenses are paid.

```
life = SULT()
contract = Contract(benefit=100000,
                    initial_policy=300,
                    initial_premium=.5,
```

(continues on next page)

(continued from previous page)

```
                    renewal_premium=.1)
P = life.gross_premium(A=life.whole_life_insurance(40), **contract.premium_terms)
A = life.whole_life_insurance(41)
a = life.immediate_annuity(41)   # after premium and expenses are paid
V = life.gross_future_loss(A=A,
                           a=a,
                           contract=contract.set_contract(premium=P).renewals())
isclose(720, V, question="Q7.19")
```

```
----- Q7.19 720: 722.7510208759086 [OK] -----
```

```
True
```

SOA Question 7.20 : (E) -277.23

For a fully discrete whole life insurance of 1000 on (35), you are given:

1. First year expenses are 30% of the gross premium plus 300
2. Renewal expenses are 4% of the gross premium plus 30
3. All expenses are incurred at the beginning of the policy year
4. Gross premiums are calculated using the equivalence principle
5. The gross premium policy value at the end of the first policy year is *R*
6. Using the Full Preliminary Term Method, the modified reserve at the end of the first policy year is *S*
7. Mortality follows the Standard Ultimate Life Table
8. $i = 0.05$

Calculate *R*–*S* .

```
life = SULT()
S = life.FPT_policy_value(35, t=1, b=1000)  # is 0 for FPT at t=0,1
contract = Contract(benefit=1000,
                    initial_premium=.3,
                    initial_policy=300,
                    renewal_premium=.04,
                    renewal_policy=30)
G = life.gross_premium(A=life.whole_life_insurance(35), **contract.premium_terms)
R = life.gross_policy_value(35, t=1, contract=contract.set_contract(premium=G))
isclose(-277.23, R - S, question="Q7.20")
```

```
----- Q7.20 -277.23: -277.19303323929216 [OK] -----
```

```
True
```

SOA Question 7.21 : (D) 11866

```
life = SULT()
x, t, u = 55, 9, 10
P = life.gross_premium(IA=0.14743,
                       a=life.temporary_annuity(x, t=u),
```

(continues on next page)

(continued from previous page)

```
                        A=life.deferred_annuity(x, u=u),
                        benefit=1000)
contract = Contract(initial_policy=life.term_insurance(x+t, t=1, b=10*P),
                     premium=P,
                     benefit=1000)
a = life.temporary_annuity(x+t, t=u-t)
A = life.deferred_annuity(x+t, u=u-t)
V = life.gross_future_loss(A=A, a=a, contract=contract)
isclose(11866, V, question="Q7.21")
```

```
----- Q7.21 11866: 11866.30158100453 [OK] -----
```

```
True
```

SOA Question 7.22 : (C) 46.24

```
life = PolicyValues().set_interest(i=0.06)
contract = Contract(benefit=8, premium=1.250)
def fun(A2):
    return life.gross_variance_loss(A1=0, A2=A2, contract=contract)
A2 = life.solve(fun, target=20.55, grid=20.55/8**2)
contract = Contract(benefit=12, premium=1.875)
var = life.gross_variance_loss(A1=0, A2=A2, contract=contract)
isclose(46.2, var, question="Q7.22")
```

```
----- Q7.22 46.2: 46.2375 [OK] -----
```

```
True
```

SOA Question 7.23 : (D) 233

```
life = Recursion().set_interest(i=0.04).set_p(0.995, x=25)
A = life.term_insurance(25, t=1, b=10000)
def fun(beta):  # value of premiums in first 20 years must be equal
    return beta * 11.087 + (A - beta)
beta = life.solve(fun, target=216 * 11.087, grid=[140, 260])
isclose(233, beta, question="Q7.23")
```

$$\begin{aligned} \texttt{Term Insurance } A^{1}_{x+25:\overline{1|}} * 10000 &:= \\ A^{1}_{x+25:\overline{1|}} &= A_{x+25:\overline{1|}} - E_{x+25} && \texttt{endowment insurance - pure} \\ E_{x+25} &= p_{x+25} * v && \texttt{pure endowment} \end{aligned}$$

```
----- Q7.23 233: 232.64747466274176 [OK] -----
```

```
True
```

SOA Question 7.24 : (C) 680

For a fully discrete whole life insurance policy of 1,000,000 on (50), you are given:

1. The annual gross premium, calculated using the equivalence principle, is 11,800
2. Mortality follows the Standard Ultimate Life Table
3. $i = 0.05$

Calculate the expense loading, P for this policy.

```
life = SULT()
P = life.premium_equivalence(A=life.whole_life_insurance(50), b=1000000)
isclose(680, 11800 - P, question="Q7.24")
```

```
----- Q7.24 680: 680.291823645397 [OK] -----
```

```
True
```

SOA Question 7.25 : (B) 3947.37

```
life = SelectLife().set_interest(i=.04)\
                   .set_table(A={55: [.23, .24, .25],
                                 56: [.25, .26, .27],
                                 57: [.27, .28, .29],
                                 58: [.20, .30, .31]})
V = life.FPT_policy_value(55, t=3, b=100000)
isclose(3950, V, question="Q7.25")
```

```
----- Q7.25 3950: 3947.3684210526353 [OK] -----
```

```
True
```

SOA Question 7.26 : (D) 28540

- backward = forward reserve recursion

```
x = 0
life = Recursion(verbose=False).set_interest(i=.05)\
                               .set_p(0.85, x=x)\
                               .set_p(0.85, x=x+1)\
                               .set_reserves(T=2, endowment=50000)
def benefit(k): return k * 25000
def fun(P):  # solve P s.t. V is equal backwards and forwards
    policy = dict(t=1, premium=P, benefit=benefit, reserve_benefit=True)
    return life.t_V_backward(x, **policy) - life.t_V_forward(x, **policy)
P = life.solve(fun, target=0, grid=[27650, 28730])
isclose(28540, P, question="Q7.26")
```

```
----- Q7.26 28540: 28542.392566782808 [OK] -----
```

```
True
```

SOA Question 7.27 : (B) 213

```
x = 0
life = Recursion(verbose=False).set_interest(i=0.03)\
                                .set_q(0.008, x=x)\
                                .set_reserves(V={0: 0})
def fun(G):  # Solve gross premium from expense reserves equation
    return life.t_V(x=x, t=1, premium=G - 187, benefit=lambda t: 0,
                    per_policy=10 + 0.25*G)
G = life.solve(fun, target=-38.70, grid=[200, 252])
isclose(213, G, question="Q7.27")
```

```
----- Q7.27 213: 212.970355987055 [OK] -----
```

```
True
```

SOA Question 7.28 : (D) 24.3

```
life = SULT()
PW = life.net_premium(65, b=1000)    # 20_V=0 => P+W is net premium for A_65
P = life.net_premium(45, t=20, b=1000)   # => P is net premium for A_45:20
isclose(24.3, PW - P, question="Q7.28")
```

```
----- Q7.28 24.3: 24.334725400123975 [OK] -----
```

```
True
```

SOA Question 7.29 : (E) 2270

For a fully discrete whole life insurance of *B* on *(x)*, you are given:

1. Expenses, incurred at the beginning of each year, equal 30 in the first year and 5 in subsequent years
2. The net premium policy value at the end of year 10 is 2290
3. Gross premiums are calculated using the equivalence principle
4. $i = 0.04$
5. $\ddot{a}_x = 14.8$
6. $\ddot{a}_{x+10} = 11.4$

Calculate ${}_{10}V^g$, the gross premium policy value at the end of year 10.

```
x = 0
life = Recursion(verbose=False).set_interest(i=0.04)\
                                .set_a(14.8, x=x)\
                                .set_a(11.4, x=x+10)
def fun(B):
    return life.net_policy_value(x, t=10, b=B)
B = life.solve(fun, target=2290, grid=2290*10)  # Solve benefit B given net 10_V
contract = Contract(initial_policy=30, renewal_policy=5, benefit=B)
G = life.gross_premium(a=life.whole_life_annuity(x), **contract.premium_terms)
V = life.gross_policy_value(x, t=10, contract=contract.set_contract(premium=G))
isclose(2270, V, question="Q7.29")
```

```
----- Q7.29 2270: 2270.743243243244 [OK] -----
```

```
True
```

SOA Question 7.30 : (E) 9035

Ten years ago J, then age 25, purchased a fully discrete 10-payment whole life policy of 10,000.

All actuarial calculations for this policy were based on the following:

1. Mortality follows the Standard Ultimate Life Table
2. $i = 0.05$
3. The equivalence principle

In addition:

1. L_{10} is the present value of future losses random variable at time 10
2. At the end of policy year 10, the interest rate used to calculate L_{10} is changed to 0%

Calculate the increase in $E[L_{10}]$ that results from this change.

```
b = 10000  # premiums=0 after t=10
L = SULT().set_interest(i=0.05).whole_life_insurance(x=35, b=b)
V = SULT().set_interest(i=0).whole_life_insurance(x=35, b=b)
isclose(9035, V - L, question="Q7.30")
```

```
----- Q7.30 9035: 9034.654127845053 [OK] -----
```

```
True
```

SOA Question 7.31 : (E) 0.310

For a fully discrete 3-year endowment insurance of 1000 on (x), you are given:

1. Expenses, payable at the beginning of the year, are:

Year(s)	Percent of Premium	Per Policy
1	20%	15
2 and 3	8%	5

2. The expense reserve at the end of year 2 is –23.64
3. The gross annual premium calculated using the equivalence principle is G = 368.
4. $G = 1000P_{x:\overline{3|}} + P^e$, where P^e is the expense loading

Calculate $P_{x:\overline{3|}}$.

```
x = 0
life = Reserves().set_reserves(T=3)
G = 368.05
def fun(P):  # solve net premium expense reserve equation
    return life.t_V(x, t=2, premium=G-P, benefit=lambda t:0, per_policy=5+0.08*G)
P = life.solve(fun, target=-23.64, grid=[.29, .31]) / 1000
isclose(0.310, P, question="Q7.31")
```

```
----- Q7.31 0.31: 0.309966 [OK] -----
```

```
True
```

SOA Question 7.32 : (B) 1.4

For two fully continuous whole life insurance policies on (x), you are given:

	Death Benefit	Annual Premium Rate	Variance of the PV of Future Loss at t
Policy A	1	0.10	0.455
Policy B	2	0.16	-

- $\delta = 0.06$

Calculate the variance of the present value of future loss at t for Policy B.

```
life = PolicyValues().set_interest(i=0.06)
contract = Contract(benefit=1, premium=0.1)
def fun(A2):
    return life.gross_variance_loss(A1=0, A2=A2, contract=contract)
A2 = life.solve(fun, target=0.455, grid=0.455)
contract = Contract(benefit=2, premium=0.16)
var = life.gross_variance_loss(A1=0, A2=A2, contract=contract)
isclose(1.39, var, question="Q7.32")
```

```
----- Q7.32 1.39: 1.3848168384380901 [OK] -----
```

```
True
```

Final Score

```
from datetime import datetime
print(datetime.now())
print(isclose)
```

```
2023-08-08 14:57:24.992543
Elapsed: 5.9 secs
Passed:  136/136
```