Solving Actuarial Math with Python

Terence Lim

Dec 01, 2022

CONTENTS

1 Life Contingent Risks **3**
- 1.1 Interest rates . . . 3
- 1.2 Probability . . . 3
- 1.3 Examples . . . 4
- 1.4 Documentation . . . 5

2 Survival **7**
- 2.1 Survival function . . . 7
- 2.2 Mortality rate . . . 7
- 2.3 Force of mortality . . . 7
- 2.4 Lifetime distribution . . . 7
- 2.5 Examples . . . 8
- 2.6 Documentation . . . 8

3 Future Lifetime **11**
- 3.1 Complete expectation of life . . . 11
- 3.2 Curtate expectation of life . . . 11
- 3.3 Temporary expectation of life . . . 11
- 3.4 Examples . . . 11
- 3.5 Documentation . . . 12

4 Fractional Ages **13**
- 4.1 Uniform distribution of deaths . . . 13
- 4.2 Constant force of mortality . . . 13
- 4.3 Examples . . . 13
- 4.4 Documentation . . . 14

5 Insurance **15**
- 5.1 Pure endowment . . . 15
- 5.2 Life insurance . . . 15
- 5.3 Variances . . . 15
- 5.4 Varying insurance . . . 16
- 5.5 Present value random variable Z . . . 16
- 5.6 Examples . . . 17
- 5.7 Documentation . . . 21

6 Annuities **25**
- 6.1 Life annuities . . . 25
- 6.2 Insurance twin . . . 25
- 6.3 Annuity Twin . . . 26
- 6.4 Immediate annuities . . . 26

6.5 Variances . . . 26
6.6 Varying annuities . . . 27
6.7 Present value random variable Y . . . 27
6.8 Examples . . . 27
6.9 Documentation . . . 29

7 Premiums **33**
7.1 Equivalence principle . . . 33
7.2 Net premium . . . 33
7.3 Gross premium . . . 34
7.4 Examples . . . 34
7.5 Documentation . . . 35

8 Policy Values **37**
8.1 Net policy value . . . 37
8.2 Gross policy value . . . 37
8.3 Variance of future loss . . . 37
8.4 Expense reserve . . . 38
8.5 Present value of loss random variable L . . . 38
8.6 Examples . . . 38
8.7 Documentation . . . 41

9 Reserves **45**
9.1 Recursion . . . 45
9.2 Interim reserves . . . 45
9.3 Modified reserves . . . 45
9.4 Examples . . . 45
9.5 Documentation . . . 46

10 Mortality Laws **49**
10.1 Uniform and Beta . . . 49
10.2 Gompertz and Makeham . . . 50
10.3 Examples . . . 50
10.4 Documentation . . . 51

11 Constant Force **53**
11.1 Constant force of mortality . . . 53
11.2 Examples . . . 53
11.3 Documentation . . . 56

12 Life Table **59**
12.1 Examples . . . 59
12.2 Documentation . . . 62

13 SULT **65**
13.1 Standard ultimate life table . . . 65
13.2 Pure endowment . . . 65
13.3 Temporary Annuity . . . 65
13.4 Examples . . . 65
13.5 Documentation . . . 71

14 Select Life Table **73**
14.1 Select and ultimate life table . . . 73
14.2 Examples . . . 73
14.3 Documentation . . . 76

15 Recursion **79**
15.1 Chain rule 79
15.2 Expected future lifetime 79
15.3 Insurance 79
15.4 Annuities 80
15.5 Examples 80
15.6 Documentation 81

16 M’thly **85**
16.1 1/m’thly insurance 85
16.2 Annuity twin 85
16.3 Immediate annuity 85
16.4 Examples 85
16.5 Documentation 86

17 UDD M’thly **89**
17.1 Annuities 89
17.2 Insurance 89
17.3 Examples 90
17.4 Documentation 92

18 Woolhouse M’thly **93**
18.1 Annuities 93
18.2 Examples 93
18.3 Documentation 95

19 Adjust Mortality **97**
19.1 Extra mortality risk 97
19.2 Examples 97
19.3 Documentation 98

20 FAM-L Solutions and Hints **99**
20.1 Tables 100
20.2 2 Survival models 101
20.3 3 Life tables and selection 103
20.4 4 Insurance benefits 107
20.5 5 Annuities 114
20.6 6 Premium Calculation 116
20.7 7 Policy Values 130

Actuarial Math - Life Contingent Risks

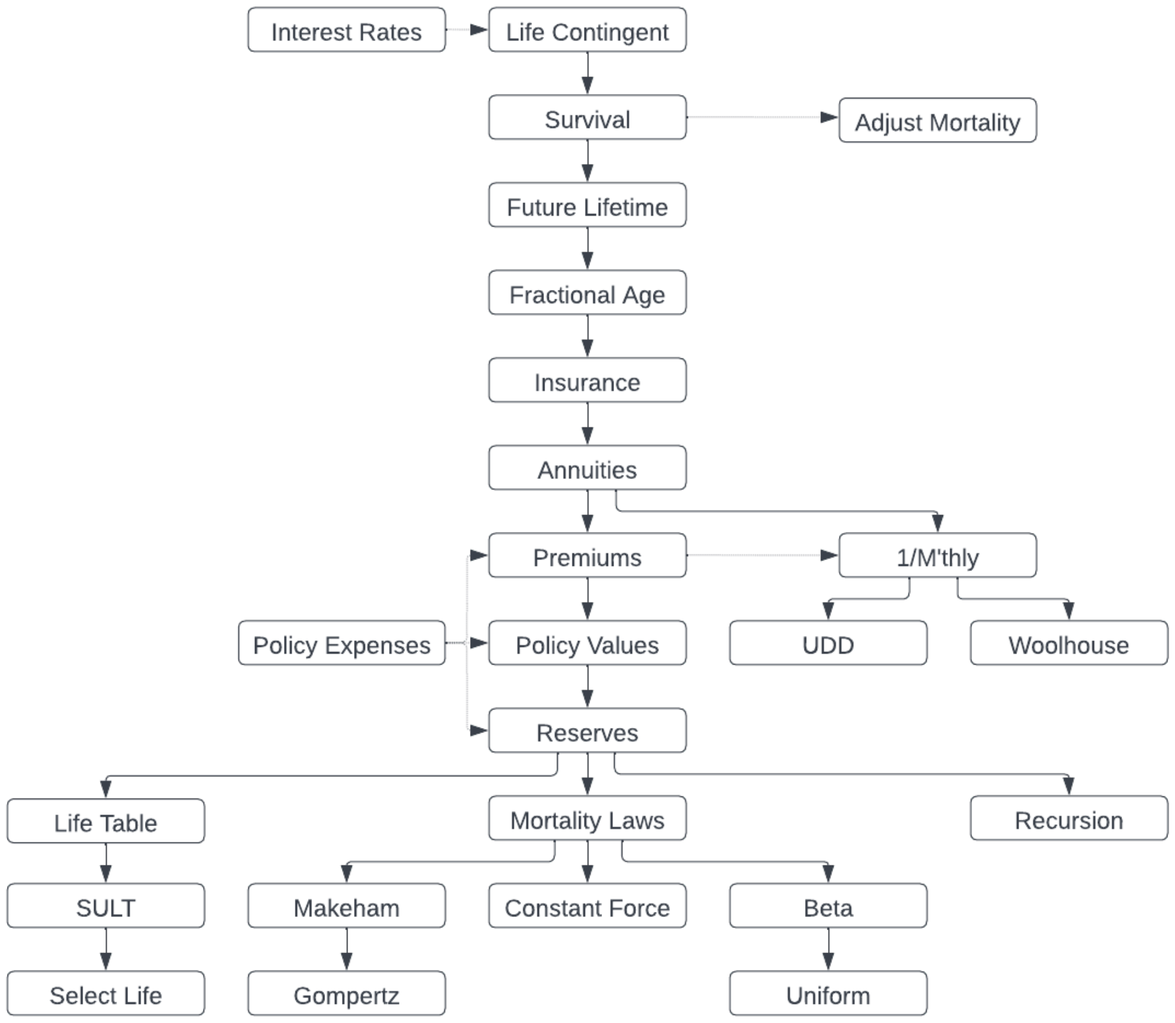
This `actuarialmath` package implements in Python the general formulas, recursive relationships and shortcut equations for Fundamentals of Long Term Actuarial Mathematics, to solve the SOA sample FAM-L questions and more.

- The concepts are developed hierarchically in object-oriented Python.
- Each module incrementally introduces the formulas used.
- The SOA sample questions (released in August 2022) are solved in an executable Google Colab Notebook.

Enjoy!

Terence Lim

Concepts and Classes



Examples

- SOA 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.

```
life = Recursion(interest=dict(i=0.04))
life.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)
print(1000 * mthly.temporary_annuity(35, t=30))
```

- SOA sample question 7.20: Calculate the policy value and modified reserve, where gross premiums are given by the equivalence principle, of a whole life insurance where mortality follows the Standard Ultimate Life Table.

```
life = SULT()
S = life.FPT_policy_value(35, t=1, b=1000)
policy = Policy(benefit=1000, initial_premium=.3, initial_policy=300,
                renewal_premium=.04, renewal_policy=30)
P = life.gross_premium(A=life.whole_life_insurance(35), **policy.premium_terms)
R = life.gross_policy_value(35, t=1, policy=policy.set(premium=P))
```

Resources

- Documentation of formulas, classes and methods: actuarialmath.pdf
- Executable Google Colab notebook: faml.ipynb
- Github repo: https://github.com/terence-lim/actuarialmath.git
- SOA FAM-L Sample Solutions: copy retrieved Aug 2022
- SOA FAM-L Sample Questions: copy retrieved Aug 2022
- Actuarial Mathematics for Life Contingent Risks (Dickson, Hardy and Waters), Institute and Faculty of Actuaries, published by Cambridge University Press

Contact me

Linkedin: https://www.linkedin.com/in/terencelim

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

CHAPTER

ONE

LIFE CONTINGENT RISKS

1.1 Interest rates

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

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

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

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

Doubling the force of interest:

- $i' \leftarrow 2i + i^2$
- $d' \leftarrow 2d - d^2$
- $v' \leftarrow v^2$
- $\delta' \leftarrow 2\delta$

Annuity certain:

- Due: $\ddot{a}_{\overline{n|}} = \frac{1-v^n}{d}$
- Immediate: $a_{\overline{n|}} = \frac{1-v^n}{i} = \ddot{a}_{\overline{n+1|}} - 1$
- Continuous: $\bar{a}_{\overline{n|}} = \frac{1-v^n}{\delta}$

1.2 Probability

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

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

Bernoulli (p) : $Y \in (a,\ b)$ w.p. $(p,\ 1-p) \Rightarrow$

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

Binomial $(N,\ p)$: Y is sum of N i.i.d. 0-1 Bernoulli(p) $\Rightarrow$

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

Mixture $(p,\ p_1,\ p_2)$: Y is Binomial $(p',\ N)$, where $p' \in (p_1,\ p_2)$ w.p. $(p,\ 1-p) \Rightarrow$

- $E[Y] = p\ N\ p_1 + (1-p)\ N\ p_2$
- $Var[Y] = E[Y^2] - E[Y]^2 = E[Var(Y \mid p') + E(Y \mid p')^2] - E[Y]^2$

Conditional Variance shortcut:

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

Portfolio Percentile (p): Y is sum of N i.i.d. r.v. each with mean μ and variance $\sigma^2 \Rightarrow$

- $Y \sim$ Normal with mean $E[Y] = N\mu$ and variance $Var[Y] = N\sigma^2$
- $Y_p = E[y] + z_p\ \sqrt{Var[Y]}$

1.3 Examples

```python
from actuarialmath.life import Life, Interest
import math

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)))
print()

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))
print()


print("Example: double the force of interest i=0.05")
i = 0.05
i2 = Interest.double_force(i=i)
d2 = Interest.double_force(d=i/(1+i))
print('i:', round(i2, 6), round(Interest(d=d2).i, 6))
print('d:', round(d2, 6), round(Interest(i=i2).d, 6))
print()

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

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

(continues on next page)

(continued from previous page)

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

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


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
```

```
z          0.842  1.036  1.282  1.645  1.960  2.326  2.576
Pr(Z<=z)     0.8   0.85    0.9   0.95  0.975   0.99  0.995
```

1.4 Documentation

```
print(Life.help())
print(Interest.help())
```

```
class Life: base class for Life Contingent Risks

    - interest (Dict) : interest rate to assume (key may be i, d, v or delta)

Methods:

 - variance(...)  Variance of weighted sum of two r.v.
        - a (float) : weight on first r.v.
        - b (float) : weight on other r.v.
        - var_a (float) : variance of first r.v.
        - var_b (float) : variance of other r.v.
        - cov_ab (float) : covariance of the r.v.'s

 - covariance(...)  Covariance of two r.v.
        - a (float) : expected value of first r.v.
        - b (float) : expected value of other r.v.
        - ab (float) : expected value of product of the two r.v.

 - bernoulli(...)  Mean or variance of bernoulli r.v. with values {a, b}
        - p (float) : probability of first value
        - a (float) : first value
        - b (float) : other value
        - variance (bool) : whether to return variance (True) or mean (False)

 - binomial(...)  Mean or variance of binomial r.v.
        - p (float) : probability of occurence
        - N (int) : number of trials
        - variance (bool) : whether to return variance (True) or mean (False)
```

(continues on next page)

(continued from previous page)

```
 - mixture(...)  Mean or variance of binomial mixture
        - p (float) : probability of selecting first r.v.
        - p1 (float) : probability of occurrence if first r.v.
        - p2 (float) : probability of occurrence if other r.v.
        - N (int) : number of trials
        - variance (bool) : whether to return variance (True) or mean (False)

 - conditional_variance(...)  Conditional variance formula
        - p (float) : probability of selecting first r.v.
        - p1 (float) : probability of occurence for first r.v.
        - p2 (float) : probability of occurence for other r.v.
        - N (int) : number of trials

 - portfolio_percentile(...)  Probability percentile of the sum of N iid r.v.'s
        - mean (float) : mean of each independent obsevation
        - variance (float) : variance of each independent observation
        - prob (float) : probability threshold
        - N (int) : number of observations to sum

 - solve(...)  Solve root of equation f(arg) = target
        - f (Callable) : output given an input value
        - target (float) : output value to target
        - guess (float or Tuple[float, float]) : initial guess, or range of guesses
        - args (tuple) : optional arguments required by function f

class Interest: interest rates conversion and math

    i, d, v, delta, i_m, d_m (float) : interest rate to be assumed
    v_t (Callable) : discount rate as a function of time
    m (int) : m'thly frequency, if i_m or d_m assumed

Methods:

 - annuity(...)  Compute value of the annuity certain factor
        t (int) : ending year
        m (int) : m'thly frequency of payments (0 for continuous payments)
        due (bool) : whether annuity due (True) or immediate (False)

 - mthly(...)  Convert to/from m'thly interest rates i, d <-> i_m, d_m
        m (int) : m'thly frequency
        i, d (float): annual-pay interest rate, to convert to m'thly, or
        i_, d_m (float): m'thly interest rate, to convert to annual pay

 - double_force(...)  Double the force of interest
        i, d, v, delta : original interest rate to double force of interest
```

CHAPTER

TWO

SURVIVAL

T_x is time-to-death, or future lifetime, of (x)

2.1 Survival function

$$S_x(t) = {}_tp_x = \text{Prob }(T_x > t) = \frac{S_0(x+t)}{S_0(x)} = 1 - F_x(t) = \int_t^\infty f_x(t)ds = e^{-\int_0^t \mu_{x+t} ds} = \frac{l_{x+t}}{l_x}$$

$$S_x(0) = 1,\ S_x(\infty) = 0$$

2.2 Mortality rate

$$_{u|t}q_x = \int_u^{u+t} {}_sp_x \mu_{x+s} ds = {}_up_x - {}_{u+t}p_x = \frac{l_{x+u} - l_{x+u+t}}{l_x}$$

$$_tp_x + {}_tq_x = 1$$

2.3 Force of mortality

$$\mu_{x+t} = \frac{f_x(t)}{S_x(t)} = \frac{-\frac{\partial}{\partial t}{}_tp_x}{{}_tp_x} = -\frac{\partial}{\partial t} \ln {}_tp_x$$

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

2.4 Lifetime distribution

$$F_x(t) = \int_0^t f_x(t)ds = 1 - S_x(t)$$

Lifetime density function: $f_x(t) = \frac{\partial}{\partial t} F_x(t) = \dfrac{f_0(x+t)}{S_0(x)} = {}_tp_x\ \mu_{x+t}$

2.5 Examples

```python
import math
from actuarialmath.survival import Survival

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

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

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

print("CAS41-F99:12: k = 41")
fun = (lambda k:
        Survival(l=lambda x,s: 100*(k - .5*(x+s))**(2/3)).mu_x(50))
print(int(Survival.solve(fun, target=1/48, guess=50)))
```

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

# SOA Question 2.6: (C) 13.3
13.340451278922776

SOA Question 2.7: (B) 0.1477
0.1477272727272727

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

2.6 Documentation

```python
print(Survival.help())
```

```
class Survival: basic survival and mortality functions

    - S (Callable) : survival function, or
    - f (Callable) : lifetime distribution, or
    - l (Callable) : lives table function, or
    - mu (Callable) : force of mortality
    - maxage (int) : maximum age
    - minage (int) : minimum age
```

(continues on next page)

(continued from previous page)

```
Methods:

 - l_x(...)  Number of lives age ([x]+s): l_[x]+s
        - x (int) : age of selection
        - s (int) : years after selection

 - p_x(...)  Probability that (x) lives t years: t_p_x
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : years survived

 - q_x(...)  Probability that (x) lives for u, but not for t+u: u|t_q_[x]+s
        - x (int) : age of selection
        - s (int) : years after selection
        - u (int) : survive u years, then...
        - t (int) : death within next t years

 - f_x(...)  lifetime density function
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : death at year t

 - mu_x(...)  Force of mortality of (x+t): mu_x+t
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : force of mortality at year t

 - survival_curve(...)  Construct curve of survival probabilities at integer ages
```

CHAPTER

THREE

FUTURE LIFETIME

3.1 Complete expectation of life

First moment: $\mathring{e}_x = \int_0^\infty t\ {}_tp_x\ \mu_{x+t} ds = \int_0^\infty\ {}_tp_x\ dt$

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

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

3.2 Curtate expectation of life

Curtate future lifetime: K_x is number of completed future years by (x) prior to death $= \lfloor T_x \rfloor$

First moment: $e_x = \sum_{k=0}^\infty k\ {}_{k|}q_x\ = \sum_{k=1}^\infty\ {}_kp_x\ dt$

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

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

3.3 Temporary expectation of life

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

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

3.4 Examples

```python
from actuarialmath.lifetime import Lifetime
import math

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

(continues on next page)

(continued from previous page)

```
print(life.e_x(106))
print()

print("SOA Question 2.4: (E) 8.2")
life = Lifetime(l=lambda x,s: 0. if (x+s) >= 100 else 1 - ((x+s)**2)/10000.)
print(life.e_x(75, t=10, curtate=False))
print()

print("SOA Question 2.8: (C) 0.938")
def fun(mu):  # Solve first for mu, given start and end proportions
    male = Lifetime(mu=lambda x,s: 1.5 * mu)
    female = Lifetime(mu=lambda x,s: mu)
    return (75 * female.p_x(0, t=20)) / (25 * male.p_x(0, t=20))
mu = Lifetime.solve(fun, target=85/15, guess=-math.log(0.94))
life = Lifetime(mu=lambda x,s: mu)
print(life.p_x(0, t=1))
print()
```

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

SOA Question 2.4: (E) 8.2
8.20952380952381

SOA Question 2.8: (C) 0.938
0.9383813306903798
```

3.5 Documentation

```
print(Lifetime.help())
```

```
class Lifetime: expected future lifetimes

Methods:

 - e_x(...)  Compute moments of expected future lifetime
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : limited at t years
        - curtate (bool) : whether curtate (True) or continuous (False) lifetime
        - moment (int) : compute first (1) or second (2) moment
```

CHAPTER

FOUR

FRACTIONAL AGES

4.1 Uniform distribution of deaths

$l_{x+r} = (1-r)\, l_x + r\, l_{x+1}$ (i.e. linear interpolation)

${}_r q_x = r\, q_x$

${}_r q_{x+s} = \dfrac{r\, q_x}{1 - s\, q_x}$

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

$f_x(r) = q_x$ (i.e. constant within age)

$\mathring{e}_x = q_x \dfrac{1}{2} + p_x (1 + \mathring{e}_{x+1})$

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

$\mathring{e}_x \approx e_x + 0.5$

4.2 Constant force of mortality

$l_{x+r} = (l_x)^{1-r} \cdot (l_{x+1})^r$ (i.e. exponential interpolation)

${}_r p_x = (p_x)^r$

$\mu_{x+r} = -\ln\, p_x$ (i.e. constant within age)

$f_x(r) = e^{-\mu t} \cdot \mu$

4.3 Examples

```
from actuarialmath.fractional import Fractional
```

4.4 Documentation

```
print(Fractional.help())
```

```
class Fractional: survival functions and expected lifetimes at fractional ages

    - udd (bool) : UDD (True) or constant force of mortality (False) between ages

Methods:

 - E_r(...)  Pure endowment through fractional age: t_E_[x]+s+r
        - x (int) : age of selection
        - s (int) : years after selection
        - r (float) : fractional year after selection
        - t (float) : limited at fractional year t

 - l_r(...)  Lives at fractional age: l_[x]+s+r
        - x (int) : age of selection
        - s (int) : years after selection
        - r (float) : fractional year after selection

 - p_r(...)  Survival from and through fractional age: t_p_[x]+s+r
        - x (int) : age of selection
        - s (int) : years after selection
        - r (float) : fractional year after selection
        - t (float) : fractional number of years survived

 - q_r(...)  Deferred mortality within fractional ages: u|t_q_[x]+s+r
        - x (int) : age of selection
        - s (int) : years after selection
        - r (float) : fractional year after selection
        - u (float) : fractional number of years survived, then
        - t (float) : death within next fractional years t

 - mu_r(...)  Force of mortality at fractional age: mu_[x]+s+r
        - x (int) : age of selection
        - s (int) : years after selection
        - r (float) : fractional year after selection

 - f_r(...)  lifetime density function at fractional age: f_[x]+s+r (t)
        - x (int) : age of selection
        - s (int) : years after selection
        - r (float) : fractional year after selection
        - t (float) : death at fractional year t

 - e_r(...)  Expectation of future lifetime through fractional age: e_[x]+s:t
        - x (int) : age of selection
        - s (int) : years after selection
        - t (float) : fractional year limit of expected future lifetime

 - e_curtate(...)  Convert between curtate and complete lifetime assuming UDD␣
↪within age
        - e (float) : value of continuous lifetime, or
        - e_curtate (float) : value of curtate lifetime
```

CHAPTER FIVE

INSURANCE

5.1 Pure endowment

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

5.2 Life insurance

Whole life insurance:

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

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

Term insurance:

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

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

Endowment insurance:

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

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

Deferred insurance:

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

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

5.3 Variances

Notationally: ${}^2\overline{A}_x$ and ${}^2A_x$ are $\overline{A}_x$ and A_x respectively at double the force of interest.

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

Pure Endowment:

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

Whole life insurance:

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

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

Term insurance:

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

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

Deferred insurance:

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

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

Endowment insurance:

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

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

5.4 Varying insurance

Increasing insurance:

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

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

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

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

Decreasing insurance:

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

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

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

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

5.5 Present value random variable Z

Expected present value of insurance benefits = EPV(Z)

Whole life insurance:

$Z = v^{K_x+1}$ (discrete) or v^{T_x} (continuous)

Term insurance:

$Z = 0$ when $K_x \geq t$ or $T_x > t$, else whole life

Deferred whole life insurance:

$Z = 0$ when $K_x < t$ or $T_x \leq t$, else whole life

Endowment insurance:

$Z = v^t$ when $K_x \geq t$ or $T_x > t$, else whole life

5.6 Examples

```python
import matplotlib.pyplot as plt
import numpy as np
import math
from actuarialmath.insurance import Insurance

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

print("SOA Question 4.18  (A) 81873 ")
life = Insurance(interest=dict(delta=0.05),
                 maxage=10,
                 f=lambda x,s,t: .1 if t < 2 else .4*t**(-2))
benefit = lambda x,t: 0 if t < 2 else 100000
prob = 0.9 - life.q_x(0, t=2)
x, y = life.survival_curve()
T = life.Z_t(0, prob=prob)
life.Z_plot(0, T=T, benefit=benefit, discrete=False, curve=(x,y))
print(life.Z_from_t(T) * benefit(0, T))
print()

print("SOA Question 4.10:  (D)")
life = Insurance(interest=dict(i=0.01), 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)
```

(continues on next page)

(continued from previous page)

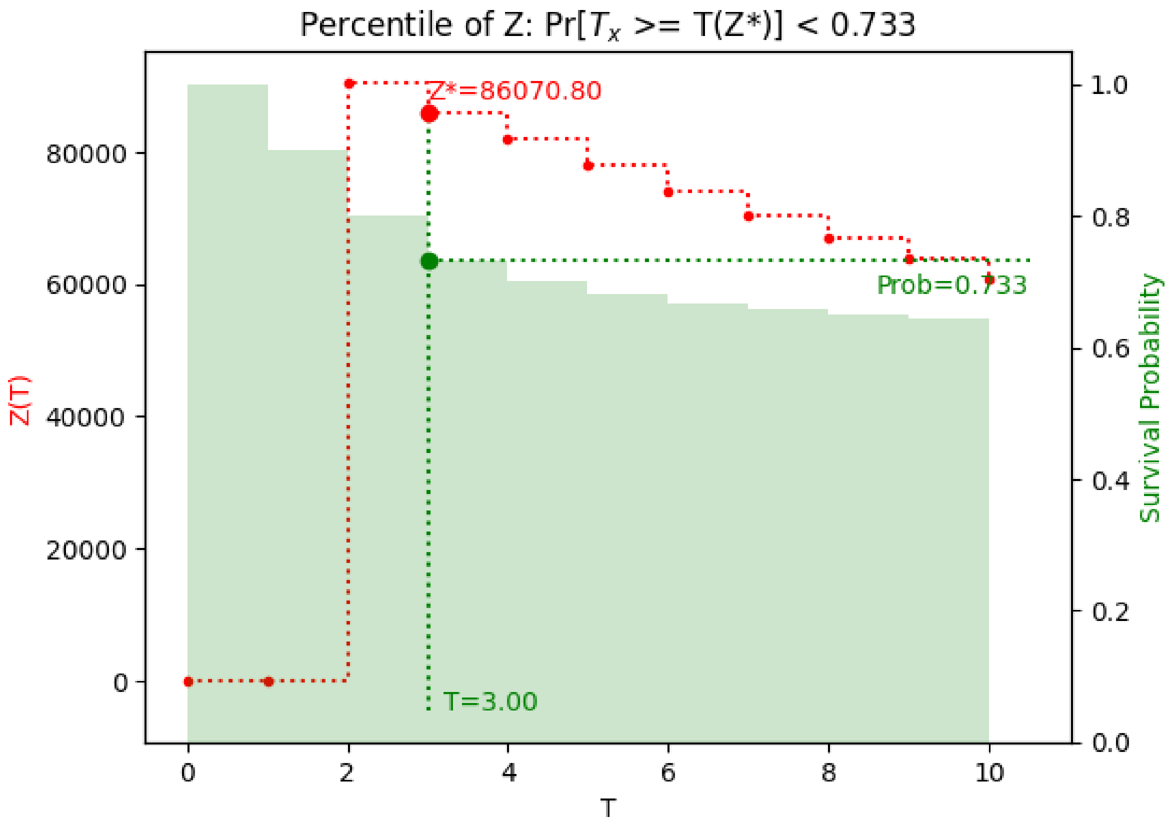
```
                                  - life.E_x(x+20, t=10) * A(x+30, t)))]
fig, ax = plt.subplots(3, 2)
ax = ax.ravel()
for i, b in enumerate([fun] + benefits):
    life.Z_plot(0, benefit=b, ax=ax[i], verbose=False, color=f"C{i+1}")
    ax[i].legend(["(" + "abcde"[i-1] + ")" if i else "Z"])
z = [sum(abs(b(0, t) - fun(0, t)) for t in range(40)) for b in benefits]
print("ABCDE"[np.argmin(z)])
print()

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

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), guess=[143400, 279300])
print(Insurance.insurance_variance(A2=A2, A1=A1, b=1000))
print()

print("SOA Question 4.15  (E) 0.0833 ")
life = Insurance(mu=lambda *x: 0.04,
                    interest=dict(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)
print(A2 - A1**2)
print()

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

# Example: plot Z vs T
life = Insurance(interest=dict(delta=0.06), mu=lambda *x: 0.04)
prob = 0.8
x = 0
discrete = False
t = life.Z_t(0, prob, discrete=discrete)
Z = life.Z_from_prob(x, prob=prob, discrete=discrete)
print(t, life.Z_to_t(Z))
print(Z, life.Z_from_t(t, discrete=discrete))
print(prob, life.Z_to_prob(x, Z=Z))
```

(continues on next page)

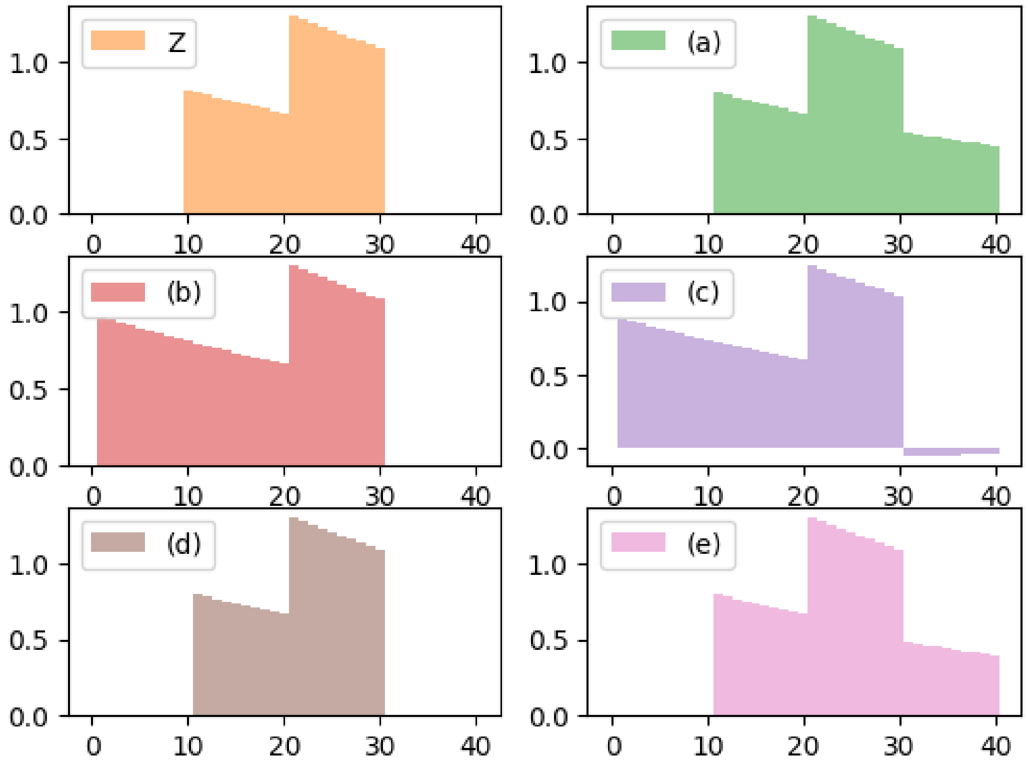
(continued from previous page)

```
life.Z_plot(0, T=t, discrete=discrete)

print("Other examples of usage")
life = Insurance(interest=dict(delta=0.06), mu=lambda *x: 0.04)
benefit = lambda x,t: math.exp(0.02 * t)
A1 = life.A_x(0, benefit=benefit)
A2 = life.A_x(0, moment=2, benefit=benefit)
print(A1, A2, A2 - A1**2)  # 0.0833

life = Insurance(interest=dict(delta=0.05), mu=lambda x,s: 0.03)
benefit = lambda x,t: math.exp(0.04 * t)
print(life.A_x(0, benefit=benefit))  #0.75
print(life.A_x(0, moment=2, benefit=benefit)) #0.60

life = Insurance(interest=dict(delta=0.08),
                    maxage=25,
                    S=lambda x,s,t: 1 - (0.02*t + 0.0008*(t**2)))
print(life.A_x(0)*10000)  #3647
print()
```

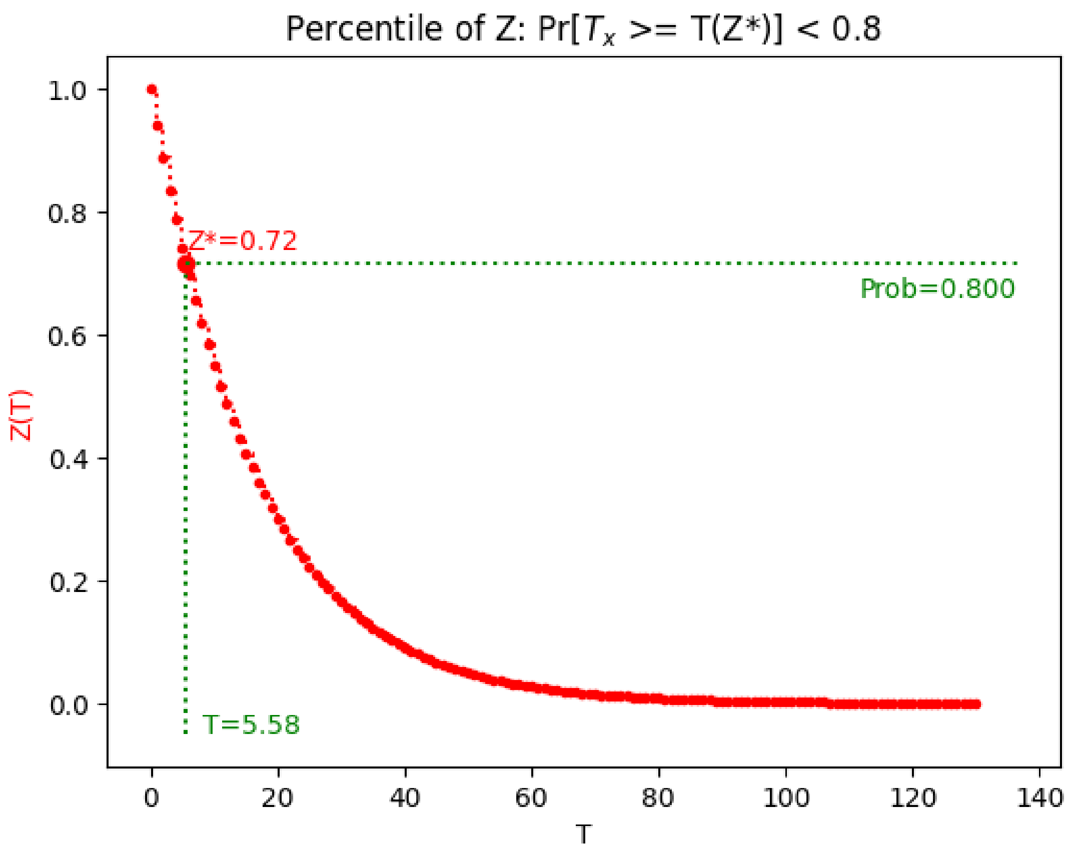
```
SOA Question 6.33:  (B) 0.13
0.10539922456186429
3884632.549746798
0.12828940905648634

SOA Question 4.18  (A) 81873
81873.07530779815

SOA Question 4.10:  (D)
D

SOA Question 4.12:  (C) 167
166.82999999999998

SOA Question 4.11:  (A) 143385
143384.99999999997

SOA Question 4.15  (E) 0.0833
0.08334849338238598

SOA Question 4.4  (A) 0.036
0.03567680106032681

5.578588782855243 5.000000000000001
0.7408182206817179 0.7155417527999328
0.8 0.8187307530779818
Other examples of usage
0.48998642116279045 0.32009235393201546 0.080005661008096
0.7421209265054034
0.5930972723997999
3503.833219537252
```



Percentile of Z: Pr[T_x >= T(Z*)] < 0.733
Z*=86070.80
Prob=0.733
T=3.00
Z(T)
Survival Probability
T



Z
(a)
(b)
(c)
(d)
(e)



5.7 Documentation

```
print(Insurance.help())
```

```
class Insurance: life insurance

Methods:

 - E_x(...)  Pure endowment: t_E_x
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of pure endowment
        - endowment (int) : amount of pure endowment
        - moment (int) : compute first or second moment

 - A_x(...)  Numerically compute EPV of insurance from basic survival functions
        - x (int) : age of selection
        - s (int) : years after selection
        - u (int) : year deferred
        - t (int) : term of insurance
        - benefit (Callable) : benefit as a function of age and year
        - endowment (int) : amount of endowment for endowment insurance
        - moment (int) : compute first or second moment
```

(continues on next page)

(continued from previous page)

```
        - discrete (bool) : benefit paid year-end (True) or moment of death (False)

- insurance_variance(...)  Compute variance of insurance given its two moments␣
↪and benefit
        - A2 (float) : second moment of insurance r.v.
        - A1 (float) : first moment of insurance r.v.
        - b (float) : benefit amount

- whole_life_insurance(...)  Whole life insurance: A_x
        - x (int) : age of selection
        - s (int) : years after selection
        - b (int) : amount of benefit
        - moment (int) : compute first or second moment
        - discrete (bool) : benefit paid year-end (True) or moment of death (False)

- term_insurance(...)  Term life insurance: A_x:t^1
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of insurance
        - b (int) : amount of benefit
        - moment (int) : compute first or second moment
        - discrete (bool) : benefit paid year-end (True) or moment of death (False)

- deferred_insurance(...)  Deferred insurance n|_A_x:t^1 = discounted term or␣
↪whole life
        - x (int) : age of selection
        - s (int) : years after selection
        - u (int) : year deferred
        - t (int) : term of insurance
        - b (int) : amount of benefit
        - moment (int) : compute first or second moment
        - discrete (bool) : benefit paid year-end (True) or moment of death (False)

- endowment_insurance(...)  Endowment insurance: A_x^1:t = term insurance + pure␣
↪endowment
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of insurance
        - b (int) : amount of benefit
        - endowment (int) : amount of endowment paid at end of term if survive
        - moment (int) : compute first or second moment
        - discrete (bool) : benefit paid year-end (True) or moment of death (False)

- increasing_insurance(...)  Increasing life insurance: (IA)_x
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of insurance
        - b (int) : amount of benefit in first year
        - discrete (bool) : benefit paid year-end (True) or moment of death (False)

- decreasing_insurance(...)  Decreasing life insurance: (DA)_x
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of insurance
        - b (int) : amount of benefit in first year
        - discrete (bool) : benefit paid year-end (True) or moment of death (False)
```

(continues on next page)

(continued from previous page)

```
- Z_t(...)  T_x given percentile of the r.v. Z: PV of WL or Term insurance
       - x (int) : age initially insured
       - prob (float) : desired probability threshold
       - discrete (bool) : benefit paid year-end (True) or moment of death (False)

- Z_from_t(...)  PV of insurance payment Z(t), given T_x (or K_x if discrete)
       - t (float) : year of death
       - discrete (bool) : benefit paid year-end (True) or moment of death (False)

- Z_to_t(...)  T_x s.t. PV of insurance payment is Z
       - Z (float) : Present value of benefit paid

- Z_from_prob(...)  Percentile of insurance PV r.v. Z, given probability
       - x (int) : age initially insured
       - prob (float) : threshold for probability of survival
       - discrete (bool) : benefit paid year-end (True) or moment of death (False)

- Z_to_prob(...)  Cumulative density of insurance PV r.v. Z, given percentile␣
↪value
       - x (int) : age initially insured
       - Z (float) : present value of benefit paid

- Z_x(...)  EPV of year t insurance death benefit
       - x (int) : age of selection
       - s (int) : years after selection
       - t (int) : term of insurance
       - discrete (bool) : benefit paid year-end (True) or moment of death (False)

- Z_plot(...)  Plot PV of insurance r.v. Z vs T
       - x (int) : age of selection
       - benefit (Callable) : benefit as a function of selection age and time
       - **kwargs : plotting options
```

CHAPTER

SIX

ANNUITIES

6.1 Life annuities

Whole life annuity:

$\overline{a}_x = \int_{t=0}^{\infty} v^t \ {}_t p_x \ dt$

$\ddot{a}_x = \sum_{k=0}^{\infty} v^k \ {}_{k|} p_x$

Temporary annuity:

$\overline{a}_{x:\overline{t|}} = \int_{s=0}^{t} v^s \ {}_s p_x \ ds = \overline{A}_x - \ {}_t E_x \ \overline{a}_{x+t}$

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

Deferred whole life annuity:

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

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

Certain and life annuity:

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

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

6.2 Insurance twin

Whole life and **Temporary Annuities** ONLY:

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

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

$\overline{a}_{x:\overline{t|}} = \dfrac{1 - \overline{A}_{x:\overline{t|}}}{\delta}$ (continuous endowment insurance twin)

$\ddot{a}_{x:\overline{t|}} = \dfrac{1 - A_{x:\overline{t|}}}{d}$ (discrete endowment insurance twin)

Double the force of interest:

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

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

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

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

6.3 Annuity Twin

Whole life and **Endowment Insurance** ONLY:

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

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

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

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

Double the force of interest:

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

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

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

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

6.4 Immediate annuities

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

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

6.5 Variances

Whole life annuity:

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

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

Temporary annuity:

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

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

6.6 Varying annuities

Increasing annuity:

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

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

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

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

Decreasing annuity:

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

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

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

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

6.7 Present value random variable Y

Expected present value of a life annuity = EPV(Y)

Whole life annuity:

$Y = \ddot{a}_{\overline{K_x+1|}}$ (discrete) or $\overline{a}_{\overline{T_x|}}$ (continuous)

Temporary insurance:

$Y = \ddot{a}_{\overline{t|}}$ (discrete) or $\overline{a}_{\overline{t|}}$ (continuous) when $K_x \geq t$ or $T_x > t$, else whole life

Certain and life annuity:

$Y = \ddot{a}_{\overline{n|}}$ (discrete) or $\overline{a}_{\overline{n|}}$ (continuous) when $K_x < n$ or $T_x \leq n$, else whole life

6.8 Examples

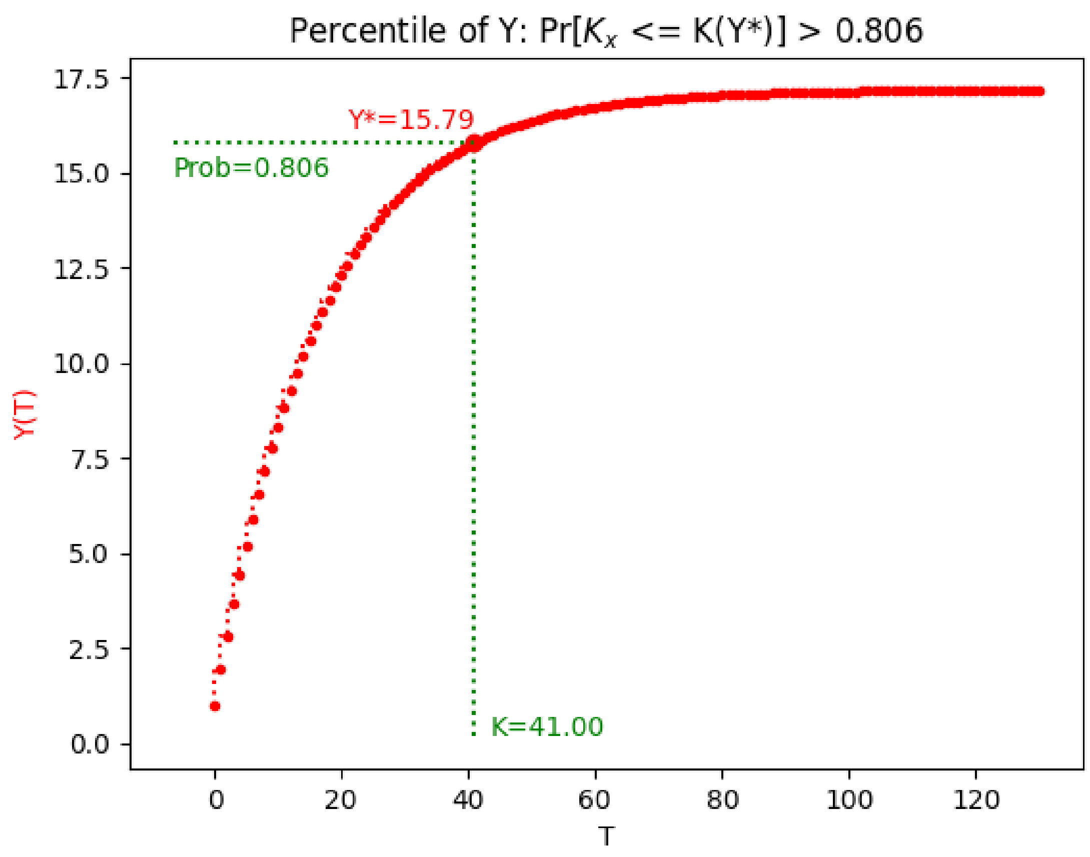
```python
from actuarialmath.annuity import Annuity
import matplotlib.pyplot as plt
print("SOA Question 5.6:  (D) 1200")
life = Annuity(interest=dict(i=0.05))
var = life.annuity_variance(A2=0.22, A1=0.45)
mean = life.annuity_twin(A=0.45)
print(life.portfolio_percentile(mean=mean, variance=var, prob=.95, N=100))
print()

print("Plot example")
life = Annuity(interest=dict(delta=0.06), mu=lambda *x: 0.04)
prob = 0.8
x = 0
discrete = True
t = life.Y_t(0, prob, discrete=discrete)
```

(continues on next page)

(continued from previous page)

```
Y = life.Y_from_prob(x, prob=prob, discrete=discrete)
print(t, life.Y_to_t(Y))
print(Y, life.Y_from_t(t, discrete=discrete))
print(prob, life.Y_to_prob(x, Y=Y))
life.Y_plot(0, T=t, discrete=discrete)
plt.show()

print("Other usage")
mu = 0.04
delta = 0.06
life = Annuity(interest=dict(delta=delta), mu=lambda *x: mu)
print(life.temporary_annuity(50, t=20, b=10000, discrete=False))
print(life.endowment_insurance(50, t=20, b=10000, discrete=False))
print(life.E_x(50, t=20))
print(life.whole_life_annuity(50, b=10000, discrete=False))
print(life.whole_life_annuity(70, b=10000, discrete=False))

mu = 0.07
delta = 0.02
life = Annuity(interest=dict(delta=delta), mu=lambda *x: mu)
print(life.whole_life_annuity(0, discrete=False) * 30)   # 333.33
print(life.temporary_annuity(0, t=10, discrete=False) * 30)  # 197.81
print(life.interest.annuity(5, m=0))  # 4.7581
print(life.deferred_annuity(0, u=5, discrete=False)) # 7.0848
print(life.certain_life_annuity(0, u=5, discrete=False))  # 11.842

mu = 0.02
delta = 0.05
life = Annuity(interest=dict(delta=delta), mu=lambda *x: mu)
print(life.decreasing_annuity(0, t=5, discrete=False))  # 6.94
```

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

Plot example
41 49.084762499581856
15.790040843594133 15.790040843594133
0.8 0.8596183508486661
```



```
Other usage
86466.4716763387
4812.011699419677
0.13533528323661273
100022.36417519346
100165.25014511104
333.3430094556871
197.81011341979988
4.758129098202016
7.085079777653729
11.843208875855744
6.94209306102519
```

6.9 Documentation

```
print(Annuity.help())
```

```
class Annuity: life annuities

Methods:

 - a_x(...)  Numerically compute EPV of annuities from survival functions
        - x (int) : age of selection
```

(continues on next page)

(continued from previous page)

```
       - s (int) : years after selection
       - u (int) : year deferred
       - t (int) : term of insurance
       - benefit (Callable) : benefit as a function of age and year
       - discrete (bool) : annuity due (True) or continuous (False)

- immediate_annuity(...)  Compute EPV of immediate life annuity
       - x (int) : age of selection
       - s (int) : years after selection
       - t (int) : term of insurance
       - b (int) : benefit amount
       - variance (bool) : return EPV (False) or variance (True)

- annuity_twin(...)  Returns annuity from its WL or Endowment Insurance twin"
       - A (float) : cost of insurance
       - discrete (bool) : discrete/annuity due (True) or continous (False)

- insurance_twin(...)  Returns WL or Endowment Insurance twin from annuity
       - a (float) : cost of annuity
       - discrete (bool) : discrete/annuity due (True) or continous (False)

- whole_life_annuity(...)  Whole life annuity: a_x
       - x (int) : age of selection
       - s (int) : years after selection
       - b (int) : annuity benefit amount
       - variance (bool): return EPV (True) or variance (False)
       - discrete (bool) : annuity due (True) or continuous (False)

- temporary_annuity(...)  Temporary life annuity: a_x:t
       - x (int) : age of selection
       - s (int) : years after selection
       - t (int) : term of annuity in years
       - b (int) : annuity benefit amount
       - variance (bool): return EPV (True) or variance (False)
       - discrete (bool) : annuity due (True) or continuous (False)

- deferred_annuity(...)  Deferred life annuity n|t_a_x =  n+t_a_x - n_a_x
       - x (int) : age of selection
       - s (int) : years after selection
       - u (int) : years deferred
       - t (int) : term of annuity in years
       - b (int) : annuity benefit amount
       - discrete (bool) : annuity due (True) or continuous (False)

- certain_life_annuity(...)  Certain and life annuity = certain + deferred
       - x (int) : age of selection
       - s (int) : years after selection
       - u (int) : years of certain annuity
       - t (int) : term of life annuity in years
       - b (int) : annuity benefit amount
       - discrete (bool) : annuity due (True) or continuous (False)

- increasing_annuity(...)  Increasing annuity
       - x (int) : age of selection
       - s (int) : years after selection
       - t (int) : term of life annuity in years
```

(continues on next page)

(continued from previous page)

```
       - b (int) : benefit amount at end of first year
       - discrete (bool) : annuity due (True) or continuous (False)

- decreasing_annuity(...)  Identity (Da)_x:n + (Ia)_x:n = (n+1) a_x:n temporary␣
↪annuity
       - x (int) : age of selection
       - s (int) : years after selection
       - t (int) : term of life annuity in years
       - b (int) : benefit amount at end of first year
       - discrete (bool) : annuity due (True) or continuous (False)

- Y_t(...)  T_x given percentile of the r.v. Y = PV of WL or Temporary Annuity
       - x (int) : age of selection
       - prob (float) : desired probability threshold
       - discrete (bool) : annuity due (True) or continuous (False)

- Y_from_t(...)  PV of insurance payment Y(t), given T_x (or K_x if discrete)
       - t (float): year of death
       - discrete (bool) : annuity due (True) or continuous (False)

- Y_from_prob(...)  Percentile of annuity PV r.v. Y, given probability
       - x (int) : age initially insured
       - prob (float) : desired probability threshold
       - discrete (bool) : annuity due (True) or continuous (False)

- Y_to_prob(...)  Cumulative density of insurance PV r.v. Y, given percentile␣
↪value
       - x (int) : age initially insured
       - Y (float) : present value of benefits paid

- Y_x(...)  EPV of t'th year's annuity benefit
       - x (int) : age initially insured
       - s (int) : years after selection
       - t (int) : year of death
       - discrete (bool) : annuity due (True) or continuous (False)

- Y_plot(...)  Plot PV of annuity r.v. Y vs T
       - x (int) : age initially insured
       - discrete (bool) : annuity due (True) or continuous (False)
       - **kwargs : plotting options
```

CHAPTER

SEVEN

PREMIUMS

7.1 Equivalence principle

Set premiums such that expected loss at issue equals zero:

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

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

7.2 Net premium

Is the premium is determined excluding expenses under the equivalence principle

- Whole life insurance: $P_x = \dfrac{A_x}{\ddot{a}_x}$ or $\dfrac{\overline{A}_x}{\overline{a}_x}$
- Term life insurance: $P^1_{x:\overline{t|}} = \dfrac{A^1_{x:\overline{t|}}}{\ddot{a}_{x:\overline{t|}}}$ or $\dfrac{\overline{A}^1_{x:\overline{t|}}}{\overline{a}_{x:\overline{t|}}}$
- Pure endowment: $P_{x:\overline{t|}}^{\ \ 1} = \dfrac{{}_tE_x}{\ddot{a}_{x:\overline{t|}}}$ or $\dfrac{{}_tE_x}{\overline{a}_{x:\overline{t|}}}$
- Endowment insurance: $P_{x:\overline{t|}} = \dfrac{A_{x:\overline{t|}}}{\ddot{a}_{x:\overline{t|}}}$ or $\dfrac{\overline{A}_{x:\overline{t|}}}{\overline{a}_{x:\overline{t|}}}$

Shortcuts for whole life and endowment insurance only:

$P = b\,(\dfrac{1}{\ddot{a}_x} - d) = b\,(\dfrac{dA_x}{1 - A_x})$ (fully discrete)

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

7.3 Gross premium

Accounts for expenses. If set under equivalence principle, then expected loss at issue equals zero:

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

Return of premiums paid without interest upon death:

- $EPV_0 = \sum_{k=0}^{t-1}~P(k+1)~v^{k+1}~_{k|}q_x = P \cdot (IA)^1_{x:\overline{t|}}$

Expenses:

- per policy and/or percent of premium initial expenses in year 1:

  e_i = initial_per_policy + initial_per_premium $\times$ gross_premium

- per policy and/or percent of premium renewal expenses in year 2+:

  e_r = renewal_per_policy + renewal_per_premium $\times$ gross_premium

- settlement expense paid with death benefit: E

  $b + E$ = death benefit + settlement expense = claim_costs

7.4 Examples

```python
from actuarialmath.premiums import Premiums
import numpy as np

print("SOA Question 6.29  (B) 20.5")
life = Premiums(interest=dict(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, guess=[20, 22]))
print()

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,
                         renewal_policy=200, initial_policy=200))
print()

print("SOA Question 6.16: (A) 2408.6")
life = Premiums(interest=dict(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)
print()
```

(continues on next page)

(continued from previous page)

```
print("SOA Question 6.20:  (B) 459")
life = Premiums(interest=dict(i=0.04),
                l=lambda x,s: dict(zip([75, 76, 77, 78],
                                    np.cumprod([1,.9,.88,.85]))).get(x+s, 0))
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, guess=100))
print()

print("Other usage")
life = Premiums(interest=dict(delta=0.06), mu=lambda x,s: 0.04)
print(life.net_premium(0))
```

```
SOA Question 6.29  (B) 20.5
20.480268314431726

SOA Question 6.2: (E) 3604
3604.229940320728

SOA Question 6.16: (A) 2408.6
0.3000139997200056 13.999720005599887 2408.575206281868

SOA Question 6.20:  (B) 459
458.83181728297285

Other usage
0.03692697915432344
```

7.5 Documentation

```
print(Premiums.help())
```

```
class Premiums: equivalence principle, net and gross premiums

Methods:

 - net_premium(...)  Net level premium for n-pay, u-deferred t-year term insurance
        - x (int) : age initially insured
        - s (int) : years after selection
        - u (int) : years of deferral
        - n (int) : number of years of premiums paid
        - t (int) : year of death
        - b (int) : benefit amount
        - endowment (int): endowment amount
        - return_premium (bool) : whether premiums without interest refunded at␣
 ↪death
        - annuity (bool) : whether benefit is insurance (False) or deferred annuity
        - discrete (bool) : discrete/annuity due (True) or continuous (False)
        - initial_cost (int) : EPV of any other expenses or benefits
```

(continues on next page)

(continued from previous page)

```
- gross_premium(...)  Gross premium by equivalence principle
       - A (float) : insurance factor
       - a (float) : annuity factor
       - IA (float) : increasing insurance factor for premiums returned w/o␣
↪interest
       - E (float) : pure endowment factor for endowment benefit
       - benefit (int) : insurance benefit amount
       - endowment (float) : endowment benefit amount
       - settlement_policy (float) : settlement expense per policy
       - initial_policy (float) : initial expense per policy
       - renewal_policy (float) : renewal expense per policy
       - initial_premium (float) : initial premium per $ of gross premium
       - renewal_premium (float) : renewal premium per $ of gross premium
       - discrete (bool) : annuity due (True) or continuous (False)

- insurance_equivalence(...)  Whole life or endowment insurance factor, given net␣
↪premium
       - premium (float) : level net premium amount
       - b (int) : benefit amount
       - discrete (bool) : discrete/annuity due (True) or continuous (False)

- annuity_equivalence(...)  Whole life or temporary annuity factor, given net␣
↪premium
       - premium (float) : level net premium amount
       - b (int) : benefit amount
       - discrete (bool) : discrete/annuity due (True) or continuous (False)

- premium_equivalence(...)  Premium given whole life or temporary/endowment␣
↪annuity/insurance
       - A (float) : insurance factor
       - a (float) : annuity factor
       - b (int) : insurance benefit amount
       - discrete (bool) : annuity due (True) or continuous (False)
```

CHAPTER

EIGHT

POLICY VALUES

8.1 Net policy value

${}_tV = EPV_t(\text{future benefits}) - EPV_t(\text{future premiums})$

- ${}_0V = E[{}_0L] = 0$ (assumes equivalence principle)
- ${}_nV = E[{}_nL] = 0$ for a n-year term insurance
- ${}_nV = E[{}_nL] =$ endowment benefit for a n-year endowment insurance

Shortcuts for whole life and endowment insurance:

${}_tV = E[{}_tL] = b[1 - \frac{\ddot{a}_{x+t}}{\ddot{a}_x}]$ or $b[\frac{A_{x+t} - A_x}{1 - A_x}]$ (fully discrete)

${}_tV = E[{}_tL] = b[1 - \frac{\overline{a}_{x+t}}{\overline{a}_x}]$ or $b[\frac{\overline{A}_{x+t} - \overline{A}_x}{1 - \overline{A}_x}]$ (fully continuous)

8.2 Gross policy value

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

8.3 Variance of future loss

Whole life and endowment insurance only:

- Net future loss*:

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

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

- Gross future loss*:

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

Shortcuts for variance of net future loss (net premiums under equivalence principle)*:

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

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

*For whole life insurance, remove the $\overline{:n|}$ and $\overline{:n-t|}$ notations.

8.4 Expense reserve

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

Generally:

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

8.5 Present value of loss random variable L

**Note: we have used the terms "expected future loss", "policy value" and "reserves" interchangeably and loosely.

For full discrete whole life or endowment insurance:

- Net future loss

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

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

- Gross future loss

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

8.6 Examples

```python
from actuarialmath.policyvalues import PolicyValues, Policy
from actuarialmath.sult import SULT
import matplotlib.pyplot as plt

print("SOA Question 6.24:  (E) 0.30")
life = PolicyValues(interest=dict(delta=0.07))
x, A1 = 0, 0.30   # Policy for first insurance
P = life.premium_equivalence(A=A1, discrete=False)  # Need its premium
policy = Policy(premium=P, discrete=False)
def fun(A2):  # Solve for A2, given Var(Loss)
    return life.gross_variance_loss(A1=A1, A2=A2, policy=policy)
A2 = life.solve(fun, target=0.18, guess=0.18)
print()

policy = Policy(premium=0.06, discrete=False) # Solve second insurance
variance = life.gross_variance_loss(A1=A1, A2=A2, policy=policy)
print(variance)
print()
```

(continues on next page)

(continued from previous page)

```
print("SOA Question 6.30:  (A) 900")
life = PolicyValues(interest=dict(i=0.04))
policy = Policy(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, policy=policy)
print(var)
print()

print("SOA Question 7.32:  (B) 1.4")
life = PolicyValues(interest=dict(i=0.06))
policy = Policy(benefit=1, premium=0.1)
def fun(A2):
    return life.gross_variance_loss(A1=0, A2=A2, policy=policy)
A2 = life.solve(fun, target=0.455, guess=0.455)
policy = Policy(benefit=2, premium=0.16)
var = life.gross_variance_loss(A1=0, A2=A2, policy=policy)
print(var)
print()

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

```
SOA Question 6.24:  (E) 0.30

0.30419999999999975

SOA Question 6.30:  (A) 900
908.141412994607

SOA Question 7.32:  (B) 1.4
1.3848168384380901

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

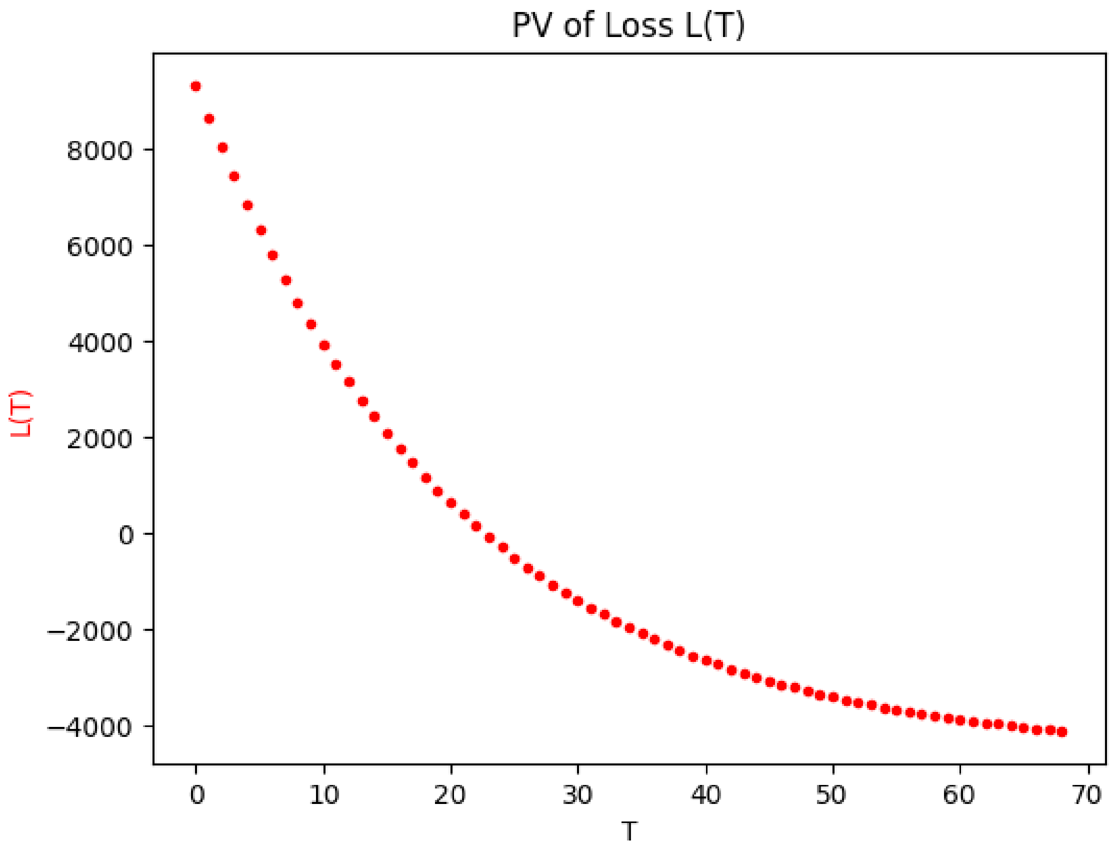
```
print("Plot Example -- SOA Question 6.6:  (B) 0.79")
life = SULT()
P = life.net_premium(62, b=10000)
policy = Policy(premium=1.03*P, renewal_policy=5,
                        initial_policy=5, initial_premium=0.05, benefit=10000)
L = life.gross_policy_value(62, policy=policy)
var = life.gross_policy_variance(62, policy=policy)
```

(continues on next page)

(continued from previous page)

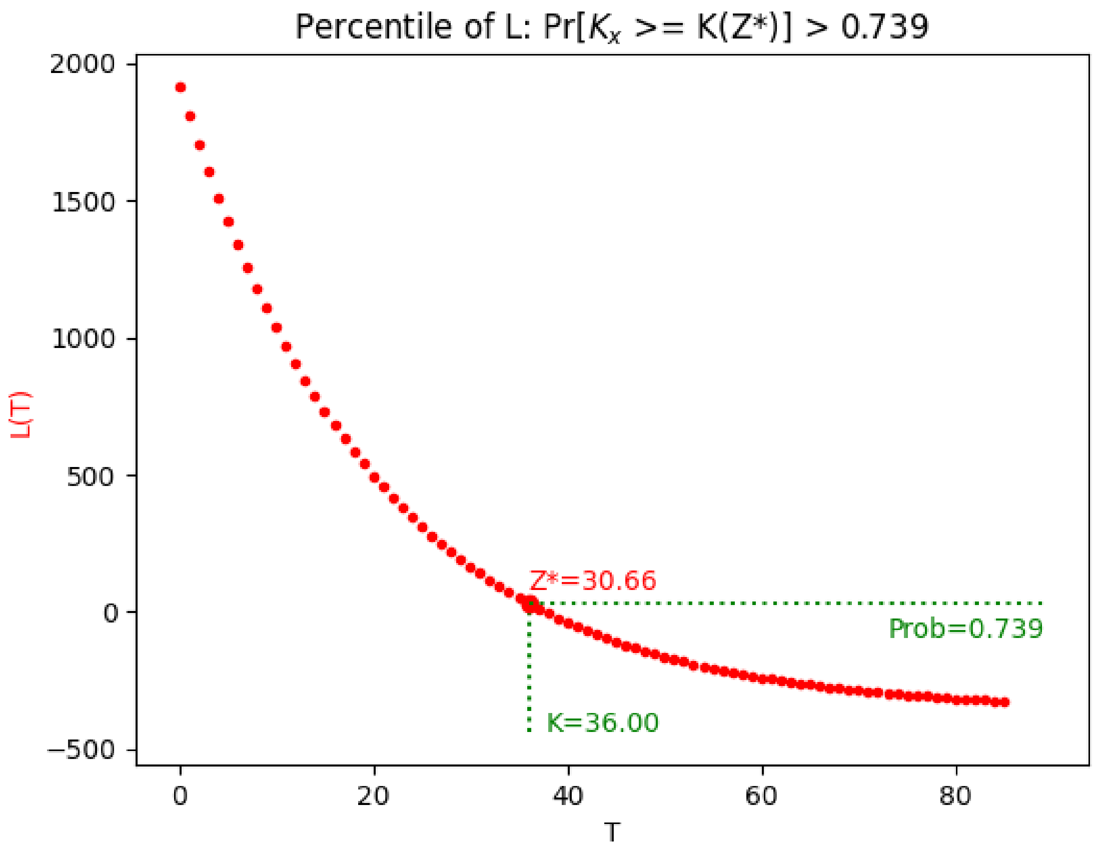
```
prob = life.portfolio_cdf(mean=L, variance=var, value=40000, N=600)
print(prob, 0.79)
life.L_plot(62, policy=policy)
print()
```

```
Plot Example -- SOA Question 6.6:  (B) 0.79
0.7914321142683509 0.79
```



```
print("Plot Example -- SOA QUestion 7.6:  (E) -25.4")
life = SULT()
P = life.net_premium(45, b=2000)
policy = Policy(benefit=2000, initial_premium=.25, renewal_premium=.05,
                initial_policy=2*1.5 + 30, renewal_policy=2*.5 + 10)
G = life.gross_premium(a=life.whole_life_annuity(45), **policy.premium_terms)
gross = life.gross_policy_value(45, t=10, policy=policy.set(premium=G))
net = life.net_policy_value(45, t=10, b=2000)
V = gross - net
print(V, -25.4)
T = life.L_to_t(G, policy=policy)
print(G)
life.L_plot(45, T=int(T), policy=policy)

plt.show()
```

```
Plot Example -- SOA QUestion 7.6:  (E) -25.4
-25.44920289521204 -25.4
31.161950196480408
```



8.7 Documentation

```
print(PolicyValues.help())
print(Policy.help())
```

```
class PolicyValues: net and gross future losses

Methods:

 - net_future_loss(...)  Shortcuts for WL or Endowment Insurance net loss
        - A (float) : insurance factor at age (x)
        - A1 (float) : insurance factor at t years after x
        - b (int) : benefit amount

 - net_variance_loss(...)  Shortcuts for variance of net loss of WL or Endowment␣
 ↪Insurance
        - A (float) : insurance factor at age (x)
        - A1 (float) : first moment of insurance factor at t years after x
```

(continues on next page)

(continued from previous page)

```
        - A2 (float) : insurance factor at double force of interest t years after x
        - b (int) : benefit amount

- net_policy_variance(...)  Variance of future loss for WL or Endowment Ins␣
↪assuming equivalence
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of life annuity in years
        - n (int) : number of years premiums paid
        - b (int) : benefit amount
        - endowment (float) : endowment amount
        - discrete (bool) : annuity due (True) or continuous (False)

- gross_future_loss(...)  Shortcut for WL or Endowment Insurance gross future loss
        - A (float) : insurance factor at age (x)
        - a (float) : annuity factor at age (x)
        - policy (Policy) : policy terms and expenses

- gross_policy_variance(...)  Variance of gross policy value for WL and Endowment␣
↪Insurance
        - x (int) : age initially insured
        - s (int) : years after selection
        - n (int) : number of years of premiums paid
        - t (int) : year of death
        - policy (Policy) : policy terms and expenses

- gross_policy_value(...)  Gross policy values for insurance: t_V = E[L_t]
        - x (int) : age initially insured
        - s (int) : years after selection
        - n (int) : number of years of premiums paid
        - t (int) : year of death
        - policy (Policy) : policy terms and expenses

- L_from_t(...)  PV of Loss L(t), given T_x (or K_x if discrete)
        - t (float) : fractional year of death
        - policy (Policy) : policy terms and expenses

- L_to_t(...)  T_x s.t. PV of loss is Z
        - L (float) : EPV of loss
        - policy (Policy) : policy terms and expenses

- L_from_prob(...)  Percentile of loss PV r.v. L, given probability
        - x (int) : age selected
        - prob (float) : desired probability threshold
        - policy (Policy) : policy terms and expenses

- L_to_prob(...)  Probability, given percentile of EPV of loss
        - x (int) : age selected
        - L (float) : EPV of loss
        - policy (Policy) : policy terms and expenses

- L_plot(...)  Plot loss r.v. L vs T
        - x (int) : age selected
        - policy (Policy) : policy terms and expenses
        - **kwargs : options for plotting
```

(continues on next page)

(continued from previous page)

```
class Policy: set policy expenses and benefit amounts

    - premium (float) : level premium amount
    - benefit (int) : insurance benefit amount
    - settlement_policy (float) : settlement expense per policy
    - endowment (float) : endowment benefit amount
    - initial_policy (float) : initial expense per policy
    - renewal_policy (float) : renewal expense per policy
    - initial_premium (float) : initial premium per $ of gross premium
    - renewal_premium (float) : renewal premium per $ of gross premium
    - discrete (bool) : annuity due (True) or continuous (False)
    - T (int) : term of insurance
    - discrete (bool) : annuity due (True) or continuous (False)

Methods:

 - premium_terms(...)  Return dict of terms required for calculating premiums

 - renewal_profit(...)  Renewal premium, less renewal per premium and policy␣
 ↪expenses

 - initial_cost(...)  Initial per premium and per policy expense, less premium

 - claims_cost(...)  Total claims costs = death benefit + settlement expense

 - policy_renewal(...)  New policy object with initial terms set to renewal, for t␣
 ↪> 0
```

CHAPTER

NINE

RESERVES

9.1 Recursion

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

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

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

9.2 Interim reserves

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

9.3 Modified reserves

Full Preliminary Term

- Initial premium: $\alpha = A^{1}_{x:\overline{1|}} = v\,q_x$
- Renewal premium: $\beta = \dfrac{A_{x+1}}{\ddot{a}_{x+1}}$

$${}_0V^{FPT} \;=\; {}_1V^{FPT} \;= 0$$
$${}_tV^{FPT} \text{ for } (x) \;=\; {}_{t-1}V^{FPT} \text{ for } (x+1)$$

9.4 Examples

```
import matplotlib.pyplot as plt
from actuarialmath.sult import SULT
from actuarialmath.reserves import Reserves
from actuarialmath.policyvalues import PolicyValues, Policy

print("SOA Question 7.31:  (E) 0.310")
x = 0
life = Reserves().set_reserves(T=3)
```

(continues on next page)

(continued from previous page)

```python
print(life._reserves)
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, guess=[.29, .31]) / 1000
print(P)
print()

print("SOA Question 7.13: (A) 180")
life = SULT()
V = life.FPT_policy_value(40, t=10, n=30, endowment=1000, b=1000)
print(V)
print()


print("Plot example: TODO from 6.12 -- this is temporary placeholder!")
life = PolicyValues(interest=dict(i=0.06))
a = 12
A = life.insurance_twin(a)
policy = Policy(benefit=1000, settlement_policy=20,
                initial_policy=10, initial_premium=0.75,
                renewal_policy=2, renewal_premium=0.1)
policy.premium = life.gross_premium(A=A, a=a, **policy.premium_terms)

life = Reserves(interest=dict(delta=0.06), mu=lambda x,s: 0.04)
life.set_reserves(T=100)
life.fill_reserves(x=0, policy=policy)
#life.V_plot()
```

```
SOA Question 7.31:  (E) 0.310
{'V': {0: 0, 3: 0}}
0.309966

SOA Question 7.13: (A) 180
180.1071785904076

Plot example: TODO from 6.12 -- this is temporary placeholder!
```

9.5 Documentation

```python
print(Reserves.help())
```

```
class Reserves: recursive, interim and modified
Methods:

 - set_reserves(...)  Setup the given reserves
       - T (int) : term of policy
       - V (dict) : reserve values
       - endowment (int) : endowment benefit amount
```

(continues on next page)

(continued from previous page)

```
- fill_reserves(...)  Fill in missing reserves
       - x (int) : age selected
       - reserve_benefit (bool) : whether benefit includes value of reserves
       - policy (Polity) : policy terms and expenses
       - max_iter (int) : number of loops to fill-in reserves table.

- V_plot(...)  Plot reserves over time

- t_V_forward(...)  Forward recursion (with optional reserve benefit)
       - x (int) : age selected
       - t (int) : year of reserve to solve
       - benefit (Callable) : benefit amount at t
       - premium (float) : amount of premium paid just after t-1
       - per_premium (float) : expense per $ premium
       - per_policy (float) : expense per policy
       - reserve_benefit (bool) : whether reserve value at t included in benefit

- t_V_backward(...)  Backward recursion (with optional reserve benefit)
       - x (int) : age selected
       - t (int) : year of reserve to solve
       - benefit (Callable) : benefit amount at t+1
       - premium (float) : amount of premium paid just after t
       - per_premium (float) : expense per $ premium
       - per_policy (float) : expense per policy
       - reserve_benefit (bool) : whether reserve value at t+1 included in benefit

- t_V(...)  Try to solve time-t Reserve by forward or backward recursion
       - x (int) : age selected
       - t (int) : year of reserve to solve
       - benefit (Callable) : benefit amount
       - premium (float) : amount of premium
       - per_premium (float) : expense per $ premium
       - per_policy (float) : expense per policy
       - reserve_benefit (bool) : whether reserve value included in benefit

- r_V_forward(...)  Forward recursion for interim reserves
       - x (int) : age of selection
       - s (int) : years after selection
       - r (float) : solve for interim reserve at fractional year x+s+r
       - benefit (int) : benefit amount in year x+s+1
       - premium (float) : premium amount just after year x+s

- r_V_backward(...)  Backward recursion for interim reserves
       - x (int) : age of selection
       - s (int) : years after selection
       - r (float) : solve for interim reserve at fractional year x+s+r
       - benefit (int) : benefit amount in year x+s+1

- FPT_premium(...)  Initial or renewal Full Preliminary Term premiums
       - x (int) : age of selection
       - s (int) : years after selection
       - n (int) : term of insurance
       - b (int) : benefit amount in year x+s+1
       - first (bool) : calculate year 1 (True) or year 2+ (False) FPT premium

- FPT_policy_value(...)  Compute Full Preliminary Term policy value at time t
```

(continues on next page)

(continued from previous page)

```
        - x (int) : age of selection
        - s (int) : years after selection
        - n (int) : term of insurance
        - t (int) : year of policy value to calculate
        - b (int) : benefit amount in year x+s+1
        - endowment (int) : endowment amount
        - discrete (bool) : fully discrete (True) or continuous (False) insurance
```

CHAPTER

TEN

MORTALITY LAWS

10.1 Uniform and Beta

Beta (ω, α):

$l_x \sim (\omega - x)^\alpha$

$\mu_x = \dfrac{\alpha}{\omega - x}$

${}_tp_x = (\dfrac{\omega - (x+t)}{\omega - x})^\alpha$

$\mathring{e}_x = \dfrac{\omega - x}{\alpha + 1}$

Uniform (ω): Beta with $\alpha = 1$

$l_x \sim \omega - x$

$\mu_x = \dfrac{1}{\omega - x}$

${}_tp_x = \dfrac{\omega - (x+t)}{\omega - x}$

$\mathring{e}_x = \dfrac{\omega - x}{2}$

$\mathring{e}_{x:\overline{n|}} = {}_np_x \, n + {}_nq_x \, \dfrac{n}{2}$

${}_nE_x = v^n \dfrac{\omega - (x+n)}{\omega - x}$

$\bar{A}_x = \dfrac{\bar{a}_{\overline{\omega - x|}}}{\omega - x}$

$\bar{A}^1_{x:\overline{n|}} = \dfrac{\bar{a}_{\overline{n|}}}{\omega - x}$

10.2 Gompertz and Makeham

Makeham's Law: $c > 1,\ B > 0,\ A \geq -B$

$\mu_x = A + Bc^x$

${}_tp_x = e^{\frac{Bc^x}{\ln c}(c^t - 1) - At}$

Gompertz's Law: Makeham's Law with $A = 0$

10.3 Examples

```
from actuarialmath.mortalitylaws import MortalityLaws, Uniform, Beta, Makeham,␣
↪Gompertz
print('Beta')
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)
print()

print('Uniform')
uniform = Uniform(80, interest=dict(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, interest=dict(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
print()

print('Gompertz/Makeham')
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
```

```
Beta
0.007188905547861446
50.0
13.333333333333332

Uniform
16.03290804858584
4.47503070125663
```

(continues on next page)

(continued from previous page)

```
0.989010989010989
27.692307692307693

0.7505031903214833
0.378867519462745
0.07552885288417432
0.8260320432056576
0.30333866657857067

Gompertz/Makeham
869.3908338193208
0.048389180223511644
0.0031580641631654026
```

10.4 Documentation

```
print(MortalityLaws.help())
print(Beta.help())
print(Uniform.help())
print(Makeham.help())
print(Gompertz.help())
```

```
class MortalityLaws: shortcuts for special mortality laws

Methods:

 - l_r(...)  Fractional lives given continuous mortality law: l_[x]+s+r
       - x (int) : age of selection
       - s (int) : years after selection
       - r (float) : fractional year after selection

 - p_r(...)  Fractional age survival probability given continuous mortality law
       - x (int) : age of selection
       - s (int) : years after selection
       - r (float) : fractional year after selection
       - t (float) : death within next t fractional years

 - q_r(...)  Fractional age deferred mortality given continuous mortality law
       - x (int) : age of selection
       - s (int) : years after selection
       - r (float) : fractional year after selection
       - u (float) : survive u fractional years, then...
       - t (float) : death within next t fractional years

 - mu_r(...)  Fractional age force of mortality given continuous mortality law
       - x (int) : age of selection
       - s (int) : years after selection
       - r (float) : fractional year after selection

 - f_r(...)  fractional age lifetime density given continuous mortality law
       - x (int) : age of selection
       - s (int) : years after selection
```

(continues on next page)

(continued from previous page)

```
        - r (float) : fractional year after selection
        - t (float) : mortality at fractional year t

 - e_r(...)  Fractional age future lifetime given continuous mortality law
        - x (int) : age of selection
        - s (int) : years after selection
        - r (float) : fractional year after selection
        - t (float) : death within next t fractional years

class Beta: beta distribution (is Uniform when alpha = 1)
    - omega (int) : maximum age
    - alpha (float) : alpha paramter of beta distribution




class Uniform: uniform distribution aka DeMoivre's Law
    - omega (int) : maximum age




class Makeham: includes element in force of mortality that does not depend on age
    - A, B, c (float) : parameters of Makeham distribution c > 1, B > 0, A >= -B




class Gompertz: as age increases so does force of mortality
    - B, c (float) : parameters of Gompertz distribution c > 1, B > 0
```

CHAPTER

ELEVEN

CONSTANT FORCE

11.1 Constant force of mortality

$_tp_x = e^{-\mu t}$

Future lifetime:

$\overset{\circ}{e}_x = \frac{1}{\mu}$

$\overset{\circ}{e}_{x:\overline{n|}} = \frac{1}{\mu}(1 - e^{-\mu n})$

Pure endowment:

$_nEx = e^{-(\mu+\delta)n}$

Insurance:

$\bar{A}_x = \frac{\mu}{\mu+\delta}$

$\bar{A}_{x:\overline{t|}} = \frac{\mu}{\mu+\delta}(1 - e^{-\mu t})$

Annuities:

$\bar{a}_x = \frac{1}{\mu+\delta}$

$\bar{a}_{x:\overline{t|}} = \frac{1}{\mu+\delta}(1 - e^{-\mu t})$

11.2 Examples

```
from actuarialmath.constantforce import ConstantForce
from scipy.stats import norm
import math

print("SOA Question 6.36:  (B) 500")
life = ConstantForce(mu=0.04, interest=dict(delta=0.08))
a = life.temporary_annuity(50, t=20, discrete=False)
A = life.term_insurance(50, t=20, discrete=False)
print(a,A)
def fun(R):
        return life.gross_premium(a=a, A=A, initial_premium=R/4500,
```

(continues on next page)

(continued from previous page)

```
                                   renewal_premium=R/4500, benefit=100000)
R = life.solve(fun, target=4500, guess=[400, 800])
print(R)
print()

print("SOA Question 6.31:  (D) 1330")
life = ConstantForce(mu=0.01, interest=dict(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)
print()

print("SOA Question 6.27:  (D) 10310")
life = ConstantForce(mu=0.03, interest=dict(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))
print(benefits, payments)
print(life.term_insurance(x, t=20), life.deferred_insurance(x, u=20))
P = benefits / payments
print(P)
print()


print("SOA Question 5.4:  (A) 213.7")
life = ConstantForce(mu=0.02, interest=dict(delta=0.01))
P = 10000 / life.certain_life_annuity(40, u=life.e_x(40, curtate=False),
                                        discrete=False)
print()


print("SOA Question 5.1: (A) 0.705")
life = ConstantForce(mu=0.01, interest=dict(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
print()


print("Other examples")
life = ConstantForce(mu=0.03, interest=dict(delta=0.04))
print(life.whole_life_annuity(20, discrete=False))  # 14.286
print(life.temporary_annuity(20, t=5, discrete=False))  # 4.219

life = ConstantForce(mu=0.04, interest=dict(delta=0.07))
#print(life.T_p(30, 0.7), 1000*life.certain.Z_t(30))  # 8.9169  535.7, , 0.7 ???

life = ConstantForce(mu=.03, interest=dict(delta=.04))
print(life.E_x(20, t=5))  # .7047
print(life.whole_life_insurance(20, discrete=False))  # .4286
print(life.term_insurance(20, t=5, discrete=False))  # .1266
print(life.endowment_insurance(20, t=5, discrete=False))  # .8313
print(life.deferred_insurance(20, u=5, discrete=False))  # .3020
```

(continues on next page)

(continued from previous page)

```
life1 = ConstantForce(mu=.04, interest=dict(delta=.02))
life2 = ConstantForce(mu=.05, interest=dict(delta=.02))
life3 = ConstantForce(mu=.05, interest=dict(delta=.03))

A1 = life1.term_insurance(0, t=5, discrete=False)
E1 = life1.E_x(0, t=5)
A2 = life2.term_insurance(5, t=7, discrete=False)
E2 = life2.E_x(5, t=7)
A3 = life3.whole_life_insurance(12, discrete=False)
print(A1, E1, A2, E2, A1 + E1 * (A2 + E2 * A3))


life = ConstantForce(mu=.04, interest=dict(delta=.06))
A1 = 10 * life.deferred_insurance(0, u=5, discrete=False)
A2 = 10 * 10 * life.deferred_insurance(0, u=5, moment=2, discrete=False)
E = 100 * A1
V = 100 * (A2 - A1**2)
print(A1, A2, E, V)  # 2.426, 11.233, 242.6, 534.8
print(E + norm.ppf(0.95) * math.sqrt(V)) # 281
```

```
SOA Question 6.36:  (B) 500
7.577350389254893 0.3030940155701957
500.0

SOA Question 6.31:  (D) 1330
1326.5406293909457

SOA Question 6.27:  (D) 10310
305783.51862973545 29.66002470618696
0.26992967028309356 0.053452469414929524
10309.617799001708

SOA Question 5.4:  (A) 213.7

SOA Question 5.1: (A) 0.705
0.7053680433746505

Other examples
14.28571428571429
4.21874157544695
0.7046880897187136
0.42857142857142866
0.1265622472634085
0.831250336982122
0.3020091813080202
0.17278785287885476 0.740818220681718 0.27669543272541713 0.6126263941844162 0.
↪6614218680727285
2.426122638850533 11.233224102930535 242.61226388505327 534.7153044187462
280.64771434478587
```

11.3 Documentation

```python
print(ConstantForce.help())
```

```
class ConstantForce: constant force of mortality exponential lifetime distribution
    - mu (float) : constant value of force of mortality

Methods:

 - e_x(...)  Expected lifetime E[T_x] is memoryless: does not depend on (x)
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : limited at year t
        - curtate (bool) : whether curtate (True) or continuous (False) lifetime
        - moment (int) : first (1) or second (2) moment

 - E_x(...)  Shortcut for pure endowment: does not depend on age x
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of pure endowment
        - endowment (int) : amount of pure endowment
        - moment (int) : compute first or second moment

 - whole_life_insurance(...)  Shortcut for APV of whole life: does not depend on␣
↪age x
        - x (int) : age of selection
        - s (int) : years after selection
        - b (int) : amount of benefit
        - moment (int) : compute first or second moment
        - discrete (bool) : benefit paid year-end (True) or moment of death (False)

 - temporary_annuity(...)  Shortcut for temporary life annuity: does not depend on␣
↪age x
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of annuity in years
        - b (int) : annuity benefit amount
        - variance (bool): return APV (True) or variance (False)
        - discrete (bool) : annuity due (True) or continuous (False)

 - term_insurance(...)  Shortcut for APV of term life: does not depend on age x
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of insurance
        - b (int) : amount of benefit
        - moment (int) : compute first or second moment
        - discrete (bool) : benefit paid year-end (True) or moment of death (False)

 - Z_t(...)  Shortcut for T_x (or K_x) given survival probability for insurance
        - x (int) : age selected
        - prob (float) : desired probability threshold
        - discrete (bool) : benefit paid year-end (True) or moment of death (False)

 - Y_t(...)  Shortcut for T_x (or K_x) given survival probability for annuity
        - x (int) : age selected
        - prob (float) : desired probability threshold
```

(continues on next page)

(continued from previous page)

```
        - discrete (bool) : continuous (False) or annuity due (True)
```

CHAPTER

TWELVE

LIFE TABLE

12.1 Examples

```
from actuarialmath.lifetable import LifeTable

print("SOA Question 6.53:  (D) 720")
x = 0
life = LifeTable(interest=dict(i=0.08), q={x:.1, x+1:.1, x+2:.1}).fill()
A = life.term_insurance(x, t=3)
P = life.gross_premium(a=1, A=A, benefit=2000, initial_premium=0.35)
print(A, P)
print(life.frame())
print()

print("SOA Question 6.41:  (B) 1417")
x = 0
life = LifeTable(interest=dict(i=0.05), q={x:.01, x+1:.02}).fill()
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())
print()


print("SOA Question 3.11:  (B) 0.03")
life = LifeTable(q={50//2: .02, 52//2: .04}, udd=True).fill()
print(life.q_r(50//2, t=2.5/2))
print(life.frame())
print()


print("SOA Question 3.5:  (E) 106")
l = [99999, 88888, 77777, 66666, 55555, 44444, 33333, 22222]
a = LifeTable(l={age:l for age,l in zip(range(60, 68), l)}, udd=True)\
    .q_r(60, u=3.4, t=2.5)
b = LifeTable(l={age:l for age,l in zip(range(60, 68), l)}, udd=False)\
    .q_r(60, u=3.4, t=2.5)
print(100000 * (a - b))
print()
```

(continues on next page)

(continued from previous page)

```python
print("SOA Question 3.14:  (C) 0.345")
life = LifeTable(l={90: 1000, 93: 825},
                 d={97: 72},
                 p={96: .2},
                 q={95: .4, 97: 1}, udd=True).fill()
print(life.q_r(90, u=93-90, t=95.5-93))
print(life.frame())
print()

print("Other usage examples")
l = [110, 100, 92, 74, 58, 38, 24, 10, 0]
table = LifeTable(l={age:l for age,l in zip(range(79, 88), l)},
                  interest=dict(i=0.06), maxage=87)
print(table.mu_x(80))
#    print(table.temporary_annuity(80, t=4, m=4, due=True)) # 2.7457
#    print(table.whole_life_annuity(80, m=4, due=True, woolhouse=True)) # 3.1778
#    print(table.whole_life_annuity(80, m=4, due=False, woolhouse=True)) # 2.9278
print(table.temporary_annuity(80, t=4)) # 2.7457
print(table.whole_life_annuity(80)) # 3.1778
print('*', table.whole_life_annuity(80, discrete=False)) # 2.9278

l = [100, 90, 70, 50, 40, 20, 0]
table = LifeTable(l={age:l for age,l in zip(range(70, 77), l)},
                  interest=dict(i=0.08), maxage=76)
print(table.A_x(70),
        table.A_x(70, moment=2))  # .75848, .58486
print(table.endowment_insurance(70, t=3)) # .81974
print('*', table.endowment_insurance(70, t=3, discrete=False)) # .81974

print(table.E_x(70, t=3)) # .39692
print('*', table.term_insurance(70, t=3, discrete=False)) # .43953
print('*', table.endowment_insurance(70, t=3, discrete=False)) # .83644
print(table.E_x(70, t=3, moment=2)) # .31503
print('*', table.term_insurance(70, t=3, moment=2, discrete=False)) # .38786
print('*', table.endowment_insurance(70, t=3, moment=2, discrete=False)) # .70294


l = [1000, 990, 975, 955, 925, 890, 840]
table = LifeTable(l={age:l for age,l in zip(range(70, 77), l)},
                  interest=dict(i=0.08), maxage=76)
print(table.increasing_annuity(70, t=4, discrete=True))
print(table.decreasing_annuity(71, t=5, discrete=False))

print('*', table.endowment_insurance(70, t=3, discrete=False)) # .7976

l = [100, 90, 70, 50, 40, 20, 0]
table = LifeTable(l={age:l for age,l in zip(range(70, 77), l)},
                  interest=dict(i=0.08), maxage=76)
print(1e6*table.whole_life_annuity(70, variance=True)) #1743784
```

```
SOA Question 6.53:  (D) 720
0.23405349794238678 720.1646090534978
          l        d    q    p
```

(continues on next page)

(continued from previous page)

```
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:  (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:  (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:  (E) 106
106.16575827938624

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

Other usage examples
0.08338160893905101
3.013501078071164
3.564334685633434
* 3.0554886512477943
0.7584803549199998 0.5848621157624223
0.8197429253670677
* 0.8364390776999064
0.3969161205100847
* 0.4395229571898218
* 0.8364390776999064
0.31508481344155215
* 0.3878582966930021
* 0.7029431101345542
8.373488543413096
6.7673201739834585
* 0.7976061232001961
1744071.8039800397
```

12.2 Documentation

```
print(LifeTable.help())
```

```
class LifeTable: life tables

    - minage (int) : minimum age
    - maxage (int) : maximum age
    - udd (bool) : Fractional age UDD (True) or constant force of mortality (False)
    - l (Dict[int, float]) : lives at start of year x, or
    - d (Dict[int, float]) : deaths in year x, or
    - p (Dict[int, float]) : probabilities that (x) survives one year, or
    - q (Dict[int, float]) : probabilities that (x) dies in one year

Methods:

 - fill(...)  Fill in missing lives and mortality (does not check consistency)
        - lives (int) : initial number of lives
        - max_iter (int) : number of iterations to fill
        - verbose (bool) : level of verbosity

 - __getitem__(...)  Return a column of the life table
        - col (str) : name of table column to return

 - __setitem__(...)  Sets a column of the life table
        col (str) : name of table column to set
        row (Dict[int, float]) : values to set in table column

 - frame(...)  Return life table values in a DataFrame

 - l_x(...)  Lookup l_x from life table
        - x (int) : age of selection
        - s (int) : years after selection

 - d_x(...)  Compute deaths as lives at x_t divided by lives at x
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : death within next t years

 - p_x(...)  t_p_x = lives beginning year x+t divided lives beginning year x
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : death within next t years

 - q_x(...)  Deferred mortality: u|t_q_x = (l[x+u] - l[x+u+t]) / l[x]
        - x (int) : age of selection
        - s (int) : years after selection
        - u (int) : survive u years, then...
        - t (int) : death within next t years

 - f_x(...)  lifetime density function
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : death at year t

 - mu_x(...)  Compute mu_x from p_x in life table
```

(continues on next page)

(continued from previous page)

```
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : death within next t years

- e_x(...)  Expected curtate lifetime from sum of lives in table
        - x (int) : age of selection
        - s (int) : years after selection
        - n (int) : future lifetime limited at n years

- E_x(...)  Pure Endowment from life table and interest rate
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : survives t years
        - moment (int) : return first (1) or second (2) moment or variance (-2)
```

CHAPTER THIRTEEN

SULT

13.1 Standard ultimate life table

From SOA's "Excel Workbook for FAM-L Tables":

- interest rate $i = 0.05$
- 100000 initial lives aged 20
- Makeham's Law with $A = 0.00022,\ B = 0.0000027,\ c = 1.124$

13.2 Pure endowment

$$
{}_tE_x = v^t \, \frac{l_{x+t}}{l_x}
$$

$$
{}^2_tE_x = v^{2t} \, \frac{l_{x+t}}{l_x} = v^t \, {}_tE_x
$$

13.3 Temporary Annuity

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

$$
{}^2A^1_{x:\overline{t|}} = {}^2A_x - \, {}^2_tE_x \, {}^2A_{x+t} = {}^2A_x - \, v^t \, {}_tE_x \, {}^2A_{x+t}
$$

13.4 Examples

```
from actuarialmath.sult import SULT
from actuarialmath.life import Interest
import math

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,
```

(continues on next page)

(continued from previous page)

```
                        initial_policy=100 + other_cost, renewal_policy=20)
print(a, P)
print()


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)
print()


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)
print()


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)
print()


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)
print()


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))
print()
```

(continues on next page)

(continued from previous page)

```
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)))
print()


print("SOA Question 5.3:  (C) 0.6239")
sult = SULT()
t = 10.5
print(t * sult.E_r(40, t=t))
print()


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))
print()


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))
print()


print("SOA Question 4.5:  (C) 35200")
sult = SULT()
print(100000 * Interest(delta=0.05).v_t(sult.Z_t(45, prob=.95)))
print()


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))
print()


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)
```

(continues on next page)

(continued from previous page)

```
print(sult.portfolio_percentile(mean=mean, variance=var, prob=.95))
print()


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))
print()

print("Other usage examples")
print(sult.temporary_annuity(80, t=10)*20000) # ~130770

E = sult.E_x(60, t=5)
print(E, E*sult.a_x(65, benefit=(lambda x,t: 1000 + .05*t)))

print(sult.whole_life_annuity(60))  # 14.9041
print(sult.whole_life_annuity(60, discrete=False))  # ~13.9041
print(sult.deferred_annuity(60, u=10))  #6.9485
print(sult.temporary_annuity(60, t=10))  # 7.9555
print(sult.temporary_annuity(60, t=15))  # 10.5282
print(sult.temporary_annuity(60, t=10))
print(sult.E_x(60, t=10))  #
print(sult.temporary_annuity(60, t=10, discrete=False))  # ~7.5341
print(sult.certain_life_annuity(60, u=10))  # 15.0563
print(sult.whole_life_annuity(60, variance=True, discrete=False)) # ~10.6182
print(sult.endowment_insurance(60, t=10)) # .62116
print(sult.whole_life_insurance(60, moment=2)) # .10834
print(sult.whole_life_insurance(70, moment=2)) # .21467
print(sult.endowment_insurance(60, t=10, moment=2)) # .38732
print(sult.temporary_annuity(60, t=10, variance=True)) #.6513


print(sult.p_x(70)) # 0.989587
print(math.log(sult.p_x(70, t=2)) / -2)  # 0.011103

A1 = sult.whole_life_insurance(20, discrete=False)
#print(A1, sult.whole_life(20))
A2 = sult.whole_life_insurance(50, discrete=False)
#print(A2, sult.whole_life(50))
A3 = sult.whole_life_insurance(70, discrete=False)
E2 = sult.E_x(20, t=30)
E3 = sult.E_x(20, t=50)
print(A1, E2, A2, E3, A3, 125*A1 + E2*175*A2 - E3*250*A3)  # 5,335

print(sult.whole_life_insurance(50)) # .18931
print(sult.term_insurance(50, t=20)) # .0402
print(sult.endowment_insurance(50, t=20))  # ~.31471
print(sult.E_x(50, t=20),
        sult.whole_life_insurance(70),
        sult.deferred_insurance(50, u=20)) # .14911

print(sult.whole_life_insurance(50, discrete=False)) # .19400
print(sult.term_insurance(50, t=20, discrete=False)) # .0412
print(sult.endowment_insurance(50, t=20, discrete=False))  #.38944
```

(continues on next page)

(continued from previous page)

```
print(sult.E_x(50, t=20),
        sult.whole_life_insurance(70, discrete=False),
        sult.deferred_insurance(50, u=20, discrete=False)) # .15281
print()

print("Standard Ultimate Life Table at i=0.05")
sult.frame()
```

```
SOA Question 6.52:  (D) 50.80
8.0750937741422 50.80135534704229

SOA Question 6.47:  (D) 66400
66384.13293704337

SOA Question 6.43:  (C) 170
171.22371939459944

SOA Question 6.39:  (A) 29
29.033866427845496

SOA Question 6.37:  (D) 820
4836.382819496279 1797.2773668474615 8.092602358383987
819.7190338249138

SOA Question 6.35:  (D) 530
534.4072234303344

SOA Question 5.8: (C) 0.92118
0.9211799771029529

SOA Question 5.3:  (C) 0.6239
6.23871918627528

SOA Question 4.17:  (A) 1126.7
1126.774772894844

SOA Question 4.14:  (E) 390000
389322.86778416135

SOA Question 4.5:  (C) 35200
36787.94411714415

SOA Question 3.9:  (E) 3850
3850.144345130047

SOA Question 3.8:  (B) 1505
1504.8328375406456

SOA Question 3.4:  (B) 815
815.0943255167722

Other usage examples
135770.41601330126
0.7668687235541889 10395.824343647037
14.904074300627297
```

(continues on next page)

(continued from previous page)

```
14.39854504493635
6.9485261567485095
7.955548143878787
10.52811141378872
7.955548143878787
0.5786434508971754
7.74293075434579
15.05634783239256
10.621710987844606
0.6211643741010102
0.10834081779190481
0.21466683367433795
0.38732012436971175
0.6504506203786906
0.9895866730368528
0.01110329636560811
0.05042882180359127 0.22808348858314714 0.1940084167264656 0.0794272723450692 0.
↪4388106987422976 5.333988893971686
0.18930786030072838
0.04020082061028696
0.3884385332061035
0.34823771259581654 0.4281760254481774 0.14910703969044142
0.1940084167264656
0.041197982733875926
0.38943569532969247
0.34823771259581654 0.4388106987422976 0.15281043399258967

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
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]
```

13.5 Documentation

```python
print(SULT.help())
```

```
class SULT: standard ultimate life table

    - interest (Dict) : interest rate assumed, may be i, d, v, delta
    - lives (int) : initial number of lives
    - minage (int) : minimum age
    - maxage (int) : maximum age
    - A, B, c (float) : parameters of Makeham distribution to generate SULT

Methods:

 - frame(...)  Displays SULT table in FAM-L exam format
```

CHAPTER

FOURTEEN

SELECT LIFE TABLE

14.1 Select and ultimate life table

14.2 Examples

```
from actuarialmath.sult import SULT
from actuarialmath.selectlife import Select

print("SOA Question 3.2:  (D) 14.7")
e_curtate = Select.e_curtate(e=15)
life = Select(l={65: [1000, None,],
                 66: [955, None]},
             e={65: [e_curtate, None]},
             d={65: [40, None,],
                 66: [45, None]}, udd=True).fill()
print(life.e_r(66))
print(life.frame('e'))
print()


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 = Select(q=q_, interest=dict(i=.04)).fill()
print(life.term_insurance(50, t=3))
print()


print("SOA Question 4.13:  (C) 350 ")
life = Select(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]}, interest=dict(i=.04)).fill()
print(life.deferred_insurance(65, t=2, u=2, b=2000))
print()


print("SOA Question 3.13:  (B) 1.6")
```

(continues on next page)

(continued from previous page)

```
life = Select(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]}).fill()
print(life.e_r(58, s=2))
print()


print("SOA Question 3.12: (C) 0.055 ")
life = Select(l={60: [10000, 9600, 8640, 7771],
                    61: [8654, 8135, 6996, 5737],
                    62: [7119, 6549, 5501, 4016],
                    63: [5760, 4954, 3765, 2410]}, udd=False).fill()
print(life.q_r(60, s=1, t=3.5) - life.q_r(61, s=0, t=3.5))

print()


print("SOA Question 3.7: (b) 16.4")
life = Select(q={50: [.0050, .0063, .0080],
                    51: [.0060, .0073, .0090],
                    52: [.0070, .0083, .0100],
                    53: [.0080, .0093, .0110]}).fill()
print(1000*life.q_r(50, s=0, r=0.4, t=2.5))
print()



print("SOA Question 3.6:  (D) 15.85")
life = Select(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]}).fill()
print(life.e_x(61))
print()


print("SOA Question 3.3:  (E) 1074")
life = Select(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))

print()


print("SOA Question 3.1:  (B) 117")
life = Select(l={60: [80000, 79000, 77000, 74000],
                    61: [78000, 76000, 73000, 70000],
                    62: [75000, 72000, 69000, 67000],
                    63: [71000, 68000, 66000, 65000]})
print(1000*life.q_r(60, s=0, r=0.75, t=3, u=2))
print()
```

```
SOA Question 3.2:  (D) 14.7
14.67801047120419
           0         1
e
65  14.50000  14.104167
66  14.17801  13.879121

SOA Question 4.16:  (D) .1116
0.1115661982248521

SOA Question 4.13:  (C) 350
351.0578236056159

SOA Question 3.13:  (B) 1.6
1.6003382187147688

SOA Question 3.12: (C) 0.055
0.05465655938591829

SOA Question 3.7: (b) 16.4
16.420207214428586

SOA Question 3.6:  (D) 15.85
5.846832

SOA Question 3.3:  (E) 1074
1073.684210526316

SOA Question 3.1:  (B) 117
116.7192429022082
```

```
print("Other usage examples")
life = Select(q={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]}).fill()
life.frame('l')
```

```
Other usage examples
```

```
                0             1             2             3
l
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
```

```
life.frame('q')
```

```
          0        1        2        3
q
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
```

```
print(life.p_x(21, 1, 4))   #0.99317
```

```
0.9931675400449915
```

14.3 Documentation

```
print(Select.help())
```

```
class Select: implement select and ultimate mortality life table

    - q (Dict[int, List[float]]) : probability [x]+s dies in one year
    - l (Dict[int, List[float]]) : number of lives aged [x]+s
    - d (Dict[int, List[float]]) : number of deaths of [x]+s
    - A (Dict[int, List[float]]) : whole life insurance, or
    - a (Dict[int, List[float]]) : whole life annuity, or
    - e (Dict[int, List[float]]) : expected future lifetime of [x]+s

Methods:

 - fill(...)  Fills in missing mortality values (does not check for consistency)
        lives (int) : initial number of lives
        max_iter (int) : number of iterations to fill
        verbose (bool) : level of verbosity

 - __getitem__(...)  Returns select and ultimate table values
        - col (str):  may be {'l', 'q', 'e', 'd', 'a', 'A'}

 - set_select(...)  Populate table column by year after selection
        - column (int) : column of select table to populate
        - select_age (bool): whether indexed by age selected (True) or actual
 ↪(False)
        - q (Dict[int, float]) : probabilities [x]+s dies in next year
        - l (Dict[int, float]) : number of lives aged [x]+s
        - A (Dict[int, float]) : whole life insurance at age [x]+s
        - a (Dict[int, float]) : whole life annuity at age [x]+s
        - e (Dict[int, float]) : expected future lifetime of life aged [x]+s

 - frame(...)  Returns select and ultimate table values as a DataFrame
        - col (str) : table to return

 - l_x(...)  Returns number of lives computed from select table
        - x (int) : age of selection
        - s (int) : years after selection

 - p_x(...)  t_p_[x]+s by chain rule: prod(1_p_[x]+s+y) for y in range(t)
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : survives t years

 - q_x(...)  t|u_q_[x]+s = [x]+s survives u years, does not survive next t
        - x (int) : age of selection
        - s (int) : years after selection
```

(continues on next page)

(continued from previous page)

```
       - u (int) : survives u years, then
       - t (int) : dies within next t years

- e_x(...)  Returns curtate expected life time computed from select table
       - x (int) : age of selection
       - s (int) : years after selection
       - t (int) : limit of expected future lifetime

- A_x(...)  Returns insurance value computed from select table
       - x (int) : age of selection
       - s (int) : years after selection

- a_x(...)  Returns annuity value computed from select table
       - x (int) : age of selection
       - s (int) : years after selection
```

CHAPTER

FIFTEEN

RECURSION

15.1 Chain rule

$_{t+n}p_x = \ _np_x \cdot \ _tp_{x+n}$

$_{t+n}E_x = \ _nE_x \cdot \ _tE_{x+n}$

15.2 Expected future lifetime

Complete expectation of life:

$\mathring{e}_x = \mathring{e}_{x:\overline{m|}} + \ _mp_x \ \mathring{e}_{x+m}$

- One-year recursion: $\mathring{e}_x = \mathring{e}_{x:\overline{1|}} + p_x \ \mathring{e}_{x+1}$
- Temporary expectation: $\mathring{e}_{x:\overline{m+n|}} = \mathring{e}_{x:\overline{m|}} + \ _mp_x \ \mathring{e}_{x+m:\overline{n|}}$

Curtate expectation of life:

$e_x = e_{x:\overline{m|}} + \ _mp_x \ e_{x+m}$

- One-year recursion: $e_x = p_x(1 + e_{x+1})$
- Temporary expectation: $e_{x:\overline{m+n|}} = e_{x:\overline{m|}} + \ _mp_x \ e_{x:\overline{n|}}$

15.3 Insurance

$$A_x = v \, q_x + v \, p_x \, A_{x+1} \ \Rightarrow \ A_{x+1} = \frac{A_x - v \, q_x}{v \, p_x}$$

$A^1_{x:\overline{t|}} = v \, q_x + v \, p_x \, A^1_{x+1:\overline{t-1|}}$

$A_{x:\overline{0|}} = b$

$A_{x:\overline{1|}} = q_x \, v \, b + p_x \, v \, b = v \, b$

$IA_x = v \, q_x + v \, p_x \, A_{x+1}$

$IA^1_{x:\overline{t|}} = v \, q_x + v \, p_x \, (A_{x+1} + IA^1_{x+1:\overline{t-1|}})$

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

15.4 Annuities

$$\ddot{a}_x = 1 + v\, p_x\, \ddot{a}_{x+1} \Rightarrow \ddot{a}_{x+1} = \frac{\ddot{a}_x - 1}{v\, p_x}$$

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

15.5 Examples

```python
from actuarialmath.recursion import Recursion
from actuarialmath.constantforce import ConstantForce
from actuarialmath.policyvalues import Policy

print("SOA Question 6.48:  (A) 3195")
life = Recursion(interest=dict(i=0.06), depth=5)
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)
print()

print("SOA Question 6.40: (C) 116 ")
# - standard formula discounts/accumulates by too much (i should be smaller)
x = 0
life = Recursion(interest=dict(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(interest=dict(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
print(P)
print()

print("SOA Question 6.17:  (A) -30000")
x = 0
life = ConstantForce(mu=0.1, interest=dict(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(interest=dict(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)
policy = Policy(premium=P * 2, benefit=100000, endowment=30000)
L = life1.gross_policy_value(x, t=0, n=2, policy=policy)
print(L)
print()
```

```
SOA Question 6.48:  (A) 3195
[ Pure Endowment: 5_E_0 ]
    pure endowment 5_E_0 = 5_p_0 * v^5
[ Pure Endowment: 5_E_0 ]
    pure endowment 5_E_0 = 5_p_0 * v^5
[ Term Insurance: A_5^1:3 ]
    backward: A_5 = qv + pvA_6
       backward: A_6 = qv + pvA_7
          endowment insurance - pure endowment = A_7^1:1
    pure endowment 1_E_7 = 1_p_7 * v^1
[ Term Insurance: A_5^1:3 ]
    pure endowment 1_E_7 = 1_p_7 * v^1
          endowment insurance - pure endowment = A_7^1:1
       backward: A_6 = qv + pvA_7
    backward: A_5 = qv + pvA_6
3195.1189176587473

SOA Question 6.40: (C) 116
[ Whole Life Annuity: a_0 ]
    backward: a_0 = 1 + E_0 a_1
    pure endowment 1_E_0 = 1_p_0 * v^1
7.2735849056603765 0.8000943396226414
[ Whole Life Insurance: A_1 ]
    forward: A_1 = (A_0/v - q) / p
          forward: A_1 = (A_0/v - q) / p
                forward: A_1 = (A_0/v - q) / p
                forward: A_1 = (A_0/v - q) / p
116.51945397474269

SOA Question 6.17:  (A) -30000
37251.49857703497 1.8378124241073728 20269.478042694187
[ Term Insurance: A_0^1:2 ]
    backward: A_0 = qv + pvA_1
       endowment insurance - pure endowment = A_1^1:1
    pure endowment 1_E_1 = 1_p_1 * v^1
[ Temporary Annuity: a_0:2 ]
    backward: a_0:2 = 1 + E_0 a_1:1
    pure endowment 1_E_0 = 1_p_0 * v^1
       1-year discrete annuity: a_x:1 = 1
[ Pure Endowment: 2_E_0 ]
    chain Rule: 2_E_0 = E_0 * 1_E_1
    pure endowment 1_E_1 = 1_p_1 * v^1
    pure endowment 1_E_0 = 1_p_0 * v^1
-30107.42633581125
```

15.6 Documentation

```
print(Recursion.help())
```

```
class Recursion: apply recursion and alternate formulas

    - depth (int) : maximum depth of recursions
    - verbose (bool) : level of verbosity
```

(continues on next page)

(continued from previous page)

```
Methods:

 - set_q(...)  Set mortality rate u|t_q_[x]+s given value
        - val (float) : value to set
        - x (int) : age of selection
        - s (int) : years after selection
        - u (int) : survive u years, then...
        - t (int) : death within next t years

 - set_p(...)  Set survival probability t_p_[x]+s given value
        - val (float) : value to set
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : survives next t years

 - set_e(...)  Set expected future lifetime e_[x]+s:t given value
        - val (float) : value to set
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : limit of expected future lifetime
        - curtate (bool) : curtate (True) or complete expectation (False)
        - moment (int) : first or second moment of expected future lifetime

 - set_E(...)  Set pure endowment t_E_[x]+s given value
        - val (float) : value to set
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : death within next t years
        - endowment (int) : endowment value
        - moment (int) : first or second moment of pure endowment

 - set_A(...)  Set insurance u|_A_[x]+s:t given value
        - val (float) : value to set
        - x (int) : age of selection
        - s (int) : years after selection
        - u (int) : defer u years
        - t (int) : term of insurance
        - endowment (int) : endowment amount
        - discrete (bool) : discrete (True) or continuous (False) insurance
        - moment (int) : first or second moment of insurance

 - set_IA(...)  Set increasing insurance IA_[x]+s:t given value
        - val (float) : value to set
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of increasing insurance
        - b (int) : benefit after year 1
        - discrete (bool) : discrete or continuous increasing insurance

 - set_DA(...)  Set decreasing insurance DA_[x]+s:t given value
        - val (float) : value to set
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of decreasing insurance
        - b (int) : benefit after year 1
```

(continues on next page)

(continued from previous page)

```
        - discrete (bool) : discrete or continuous decreasing insurance

 - set_a(...)  Set annuity u|_a_[x]+s:t given value
        - val (float) : value to set
        - x (int) : age of selection
        - s (int) : years after selection
        - u (int) : defer u years
        - t (int) : term of annuity
        - b (int) : benefit amount
        - discrete (bool) : whether annuity due (True) or continuous (False)
        - variance (bool) : whether first moment (False) or variance (True)
```

CHAPTER

SIXTEEN

M'THLY

16.1 1/m'thly insurance

$$K_x^{(m)} = \frac{1}{m}\lfloor mT_x \rfloor$$

$$A_x^{(m)} = \sum_{k=0}^{\infty} ~ v^{\frac{k+1}{m}} ~ _{\frac{k}{m}|\frac{1}{m}}q_x$$

$$A_{x:\overline{t|}}^{(m)} = \sum_{k=0}^{mt-1} ~ v^{\frac{k+1}{m}} ~ _{\frac{k}{m}|\frac{1}{m}}q_x$$

16.2 Annuity twin

$$A_x^{(m)} = 1 - d^{(m)} ~ \ddot{a}_x^{(m)} \iff \ddot{a}_x^{(m)} = \frac{1 - A_x^{(m)}}{d^{(m)}}$$

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

16.3 Immediate annuity

$$a_x^{(m)} = \ddot{a}_x^{(m)} - \frac{1}{m}$$

$$a_{x:\overline{t|}}^{(m)} = \ddot{a}_{x:\overline{t|}}^{(m)} - \frac{1}{m}(1 - ~_tE_x)$$

16.4 Examples

```
from actuarialmath.mthly import Mthly
from actuarialmath.premiums import Premiums
from actuarialmath.lifetable import LifeTable

print("SOA Question 6.4:  (E) 1893.9")
mthly = Mthly(m=12, life=Premiums(interest=dict(i=0.06)))
A1, A2 = 0.4075, 0.2105
mean = mthly.annuity_twin(A1)*15*12
```

(continues on next page)

(continued from previous page)

```python
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)
print()

print("SOA Question 4.2:  (D) 0.18")
life = LifeTable(q={0: .16, 1: .23}, interest=dict(i_m=.18, m=2),
                 udd=False).fill()
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].iloc[:, -1].sum())
print()
```

```
SOA Question 6.4:  (E) 1893.9
1893.912859650868

SOA Question 4.2:  (D) 0.18
               Z         p
m
1  275229.357798  0.083485
2  277754.397778  0.076515
3  277986.052822  0.102903
4  276285.832315  0.090297
0.17941813045022975
```

16.5 Documentation

```python
print(Mthly.help())
```

```
class 1/M'thly insurance and annuities

    - m (int) : number of payments per year
    - life (Premiums) : original survival and life contingent functions

Methods:

 - v_m(...)  Compute discount rate compounded over k m'thly periods
        - k (int) : number of m'thly periods to compound

 - p_m(...)  Compute survival probability over m'thly periods
        - x (int) : year of selection
        - s_m (int) : number of m'thly periods after selection
        - t_m (int) : survives number of m'thly periods

 - q_m(...)  Compute deferred mortality over m'thly periods
        - x (int) : year of selection
        - s_m (int) : number of m'thly periods after selection
        - u_m (int) : survive number of m'thly periods , then
        - t_m (int) : dies within number of m'thly periods
```

(continues on next page)

(continued from previous page)

```
- Z_m(...)  Return PV of insurance r.v. Z and probability by time as DataFrame
       - x (int) : year of selection
       - s (int) : years after selection
       - t (int) : year of death
       - benefit (Callable) : amount of benefit by year and age selected
       - moment (int) : return first or second moment
```

CHAPTER

SEVENTEEN

UDD M'THLY

17.1 Annuities

$$\alpha(m) = \frac{id}{i^{(m)}\, d^{(m)}}$$

$$\beta(m) = \frac{i - i^{(m)}}{i^{(m)}\, d^{(m)}}$$

$$\ddot{a}_x^{(m)} = \alpha(m)\, \ddot{a}_x - \beta(m)$$

$$\ddot{a}_{x:\overline{n|}}^{(m)} = \alpha(m)\, \ddot{a}_{x:\overline{n|}} - \beta(m)(1 - {}_tE_x)$$

$$_{u|}\ddot{a}_x^{(m)} = \alpha(m)\, {}_{u|}\ddot{a}_x - \beta(m)\, {}_uE_x$$

17.2 Insurance

Discrete insurance:

Whole life: $A_x^{(m)} = \dfrac{i}{i^{(m)}} A_x$

Temporary: $A_{x:\overline{t|}}^{1\ (m)} = \dfrac{i}{i^{(m)}} A_{x:\overline{t|}}^{1}$

Endowment: $A_{x:\overline{t|}}^{(m)} = \dfrac{i}{i^{(m)}} A_{x:\overline{t|}}^{1} + {}_tE_x$

Deferred: ${}_{u|}A_x^{(m)} = {}_uE_x\, \dfrac{i}{i^{(m)}} A_{x+u}$

Continuous insurance:

Whole life: $\overline{A}_x = \dfrac{i}{\delta} A_x$

Temporary: $\overline{A}_{x:\overline{t|}}^{1} = \dfrac{i}{\delta} A_{x:\overline{t|}}^{1}$

Endowment: $\overline{A}_{x:\overline{t|}} = \dfrac{i}{\delta} A_{x:\overline{t|}}^{1} + {}_tE_x$

Deferred: ${}_{u|}\overline{A}_x = {}_uE_x\, \dfrac{i}{\delta} A_{x+u}$

Double the force of interest:

$${}^2\overline{A}_x = \frac{i^2 - 2i}{2\delta}\, {}^2A_x$$

$$^2A_x^{(m)} = \frac{i^2 - 2i}{(i^{(m)})^2 - 2i^{(m)}}\ ^2A_x$$

17.3 Examples

```python
from actuarialmath.udd import UDD
from actuarialmath.sult import SULT
from actuarialmath.recursion import Recursion
from actuarialmath.policyvalues import Policy

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)
print(a)
A = UDD(m=0, life=sult).endowment_insurance(x+10, t=n-10)
print(A)
print(A*100000 - a*12*253)
policy = Policy(premium=253*12, endowment=100000, benefit=100000)
print(sult.gross_future_loss(A=A, a=a, policy=policy))
print()

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)
print()

print("SOA Question 6.38:  (B) 11.3")
x, n = 0, 10
life = Recursion(interest=dict(i=0.05))
life.set_A(0.192, x=x, t=n, endowment=1, discrete=False)
life.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(interest=dict(i=0.05))
    life.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, guess=a)  # discrete annuity
P = life.gross_premium(a=a, A=0.192, benefit=1000)
print(P)
print()

print("SOA Question 6.32:  (C) 550")
x = 0
life = Recursion(interest=dict(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)
```

(continues on next page)

(continued from previous page)

```
print()

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)
print()

print("Interest Functiona at i=0.05")
print("----------------------------")
print(UDD.frame())
print()
UDD.frame()
```

```
SOA Question 7.9:  (A) 38100
7.831075686716718
0.6187476755196442
38099.62176709247
38099.62176709246

SOA Question 6.49:  (C) 86
85.99177833261696

SOA Question 6.38:  (B) 11.3
[ Temporary Annuity: a_0:10 ]
    Annuity twin: a = (1 - A) / d
16.560714925944584
11.308644185253657

SOA Question 6.32:  (C) 550
0.5763261529803323 8.72530251348809
550.4356936711871

SOA Question 6.22:  (C) 102
102.40668704849178

Interest Functiona 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
```

```
       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
```

17.4 Documentation

```
print(UDD.help())
```

```
class UDD: 1/Mthly shortcuts with uniform distribution of deaths assumption

    - m (int) : number of payments per year
    - life (Fractional) : original fractional survival and mortality functions

Methods:

 - alpha(...)  1/Mthly UDD interest rate beta function value
       - m (int) : number of payments per year
       - i (float) : annual interest rate

 - beta(...)  1/Mthly UDD interest rate alpha function value
       - m (int) : number of payments per year
       - i (float) : annual interest rate

 - frame(...)  Display 1/mthly UDD interest function values
       - i (float) : annual interest rate
```

CHAPTER

EIGHTEEN

WOOLHOUSE M'THLY

18.1 Annuities

Whole life annuity:

$$\ddot{a}_x^{(m)} = \ddot{a}_x - \frac{m-1}{2m} - \frac{m^2-1}{12m^2}(\mu_x + \delta)$$

Temporary annuity:

$$\ddot{a}_{x:\overline{t|}}^{(m)} = \ddot{a}_x^{(m)} - {}_tE_x\ \ddot{a}_{x+t}^{(m)}$$

- Approximate $\mu_x \approx -\dfrac{1}{2}(\ln p_{x-1} + \ln p_x)$

18.2 Examples

```
from actuarialmath.woolhouse import Woolhouse
from actuarialmath.sult import SULT
from actuarialmath.recursion import Recursion
from actuarialmath.udd import UDD
from actuarialmath.policyvalues import Policy

print("SOA Question 7.7:  (D) 1110")
x = 0
life = Recursion(interest=dict(i=0.05)).set_A(0.4, x=x+10)
a = Woolhouse(m=12, life=life).whole_life_annuity(x+10)
print(a)
policy = Policy(premium=0, benefit=10000, renewal_policy=100)
V = life.gross_future_loss(A=0.4, policy=policy.policy_renewal)
print(V)
policy = Policy(premium=30*12, renewal_premium=0.05)
V1 = life.gross_future_loss(a=a, policy=policy.policy_renewal)
print(V, V1, V+V1)
print()


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)
```

(continues on next page)

(continued from previous page)

```python
payments = life.temporary_annuity(55, t=10)
print(benefits + expenses, payments)
def fun(P):
    return life.gross_future_loss(A=benefits + expenses, a=payments,
                                  policy=Policy(premium=P))
P = life.solve(fun, target=-800, guess=[12110, 12550])
print(P)
print()


print("SOA Question 6.15:  (B) 1.002")
life = Recursion(interest=dict(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))
print()

print("SOA Question 5.7:  (C) 17376.7")
life = Recursion(interest=dict(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))
print()
```

```
SOA Question 7.7:  (D) 1110
12.141666666666666
5260.0
5260.0 -4152.028174603174 1107.9718253968258

SOA Question 6.25:  (C) 12330
98042.52569470297 8.019169307712845
12325.781125438532

SOA Question 6.15:  (B) 1.002
1.0022973504113772

SOA Question 5.7:  (C) 17376.7
[ Pure Endowment: 30_E_35 ]
    pure endowment 30_E_35 = 30_p_35 * v^30
17.37671459632958
[ Pure Endowment: 30_E_35 ]
    pure endowment 30_E_35 = 30_p_35 * v^30
17376.71459632958
```

18.3 Documentation

```python
print(Woolhouse.help())
```

```
class Woolhouse: 1/m'thly shortcuts with woolhouse approximation
    - m (int) : number of payments per year
    - life (Fractional) : original fractional survival and mortality functions
    - three_term (bool) : whether to include (True) or ignore (False) third term
    - approximate_mu (Callable | bool) : function to approximate mu_x for third␣
↪term

Methods:

 - mu_x(...)  Approximates or computes mu_x for third term if not given
        - x (int) : age of selection
        - s (int) : years after selection

 - whole_life_insurance(...)  1/m'thly Woolhouse Whole life insurance: A_x
        - x (int) : age of selection
        - s (int) : years after selection
        - b (int) : amount of benefit
        - mu (float): optional mu_x for third woolhouse term

 - term_insurance(...)  1/m'thly Woolhouse Term insurance: A_x:t
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of insurance in years
        - b (int) : amount of benefit
        - mu (float): optional mu_x for third woolhouse term
        - mu1 (float): optional mu_x+t for third woolhouse term

 - endowment_insurance(...)  1/m'thly Woolhouse Term insurance: A_x:t
        - x (int) : age of selection
        - s (int) : years after selection
        - t (int) : term of insurance in years
        - b (int) : amount of benefit
        - endowment (int) : amount of endowment benefit
        - mu (float): optional mu_x for third woolhouse term

 - deferred_insurance(...)  1/m'thly Woolhouse Deferred insurance = discounted␣
↪term or whole life
        - x (int) : age of selection
        - s (int) : years after selection
        - u (int) : years deferred
        - t (int) : term of insurance in years
        - b (int) : amount of benefit
        - mu (float): optional mu_x for third woolhouse term
        - mu1 (float): optional mu_x+t for third woolhouse term

 - whole_life_annuity(...)  1/m'thly Woolhouse Whole life annuity: a_x
        - x (int) : age of selection
        - s (int) : years after selection
        - b (int) : annuity benefit amount
        - mu (float): optional mu_x for third woolhouse term

 - temporary_annuity(...)  1/m'thly Woolhouse Temporary life annuity: a_x
```

(continues on next page)

(continued from previous page)

```
            - x (int) : age of selection
            - s (int) : years after selection
            - t (int) : term of annuity in years
            - b (int) : annuity benefit amount
            - mu (float): optional mu_x for third woolhouse term
            - mu1 (float): optional mu_x+t for third woolhouse term

    - deferred_annuity(...)  1/m'thly Woolhouse Temporary life annuity: a_x
            - x (int) : age of selection
            - s (int) : years after selection
            - u (int) : years deferred
            - t (int) : term of annuity in years
            - b (int) : annuity benefit amount
            - mu (float): optional mu_x for third woolhouse term
            - mu1 (float): optional mu_x+t for third woolhouse term
```

CHAPTER

NINETEEN

ADJUST MORTALITY

19.1 Extra mortality risk

1. Add constant to force of mortality: $\mu_{x+t} + k \Rightarrow {}_tp_x \rightarrow {}_tp_x\, e^{-kt}$
2. Multiply force of mortality by constant: $\mu_{x+t} \cdot k \Rightarrow {}_tp_x \rightarrow ({}_tp_x)^k$
3. Mutiply mortality rate by a constant: $q_x \rightarrow q_x \cdot k$
4. Age rating: add years to age $\Rightarrow (x) \rightarrow (x + k)$

19.2 Examples

```
from actuarialmath.selectlife import Select
from actuarialmath.sult import SULT
from actuarialmath.adjustmortality import Adjust

print("SOA Question 5.5: (A) 1699.6")
life = SULT()
adjust = Adjust(life=life, extra=0.05, adjust=Adjust.ADD_FORCE)
select = Select(n=1)\
            .set_select(column=0, select_age=True, q=adjust.q)\
            .set_select(column=1, select_age=False, a=life['a']).fill()
print(100*select['a'][45][0])
print()

print("SOA Question 4.19:  (B) 59050")
life = SULT()
adjust = Adjust(life=life, extra=0.8, adjust=Adjust.MULTIPLY_RATE)
select = Select(n=1)\
            .set_select(column=0, select_age=True, q=adjust.q)\
            .set_select(column=1, select_age=False, q=life['q']).fill()
print(100000*select.whole_life_insurance(80, s=0))
print()

print("Other usage examples")
life = SULT()
adjust = Adjust(life=life, extra=0.05, adjust=Adjust.ADD_FORCE)
print(life.p_x(45), adjust.p_x(45))
```

```
SOA Question 5.5: (A) 1699.6
1699.6076593190103

SOA Question 4.19:  (B) 59050
59050.59973285648

Other usage examples
0.9992288829941123 0.9504959153149807
```

19.3 Documentation

```
print(Adjust.help())
```

```
class Adjust: adjusts mortality by extra risk

    - life (Survival) : original survival and mortality rates
    - adjust (int) : {ADD_FORCE, MULTIPLY_FORCE, ADD_AGE, MULTIPLY_RATE}
    - extra (float) : amount of extra risk to adjust by

Methods:

 - q_x(...)  Return q_[x]+s after adding age rating or multipliying mortality rate
       - x (int) : age of selection
       - s (int) : years after selection

 - p_x(...)  Return p_[x]+s after adding or multiplying force of mortality
       - x (int) : age of selection
       - s (int) : years after selection

 - q(...)  Adjusted mortality rates q_x, as dict keyed by age

 - p(...)  Adjusted survival probabilities p_x, as dict keyed by age
```

CHAPTER

TWENTY

FAM-L SOLUTIONS AND HINTS

```
"""Solutions and hints to SOA sample questions

Copyright 2022, Terence Lim

MIT License
"""
import math
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from actuarialmath.life import Life, Interest
from actuarialmath.survival import Survival
from actuarialmath.lifetime import Lifetime
from actuarialmath.insurance import Insurance
from actuarialmath.annuity import Annuity
from actuarialmath.premiums import Premiums
from actuarialmath.policyvalues import PolicyValues, Policy
from actuarialmath.reserves import Reserves
from actuarialmath.recursion import Recursion
from actuarialmath.lifetable import LifeTable
from actuarialmath.sult import SULT
from actuarialmath.selectlife import Select
from actuarialmath.constantforce import ConstantForce
from actuarialmath.adjustmortality import Adjust
from actuarialmath.mthly import Mthly
from actuarialmath.udd import UDD
from actuarialmath.woolhouse import Woolhouse
```

Helper function to autograde answers

- compares computed solutions to answer keys

```
from typing import Union
class Autograde:
    """Autograder to keep score of correct solutions"""
    def __init__(self, terminate: bool = False):
        self.score = {}
        self.terminate = terminate  # whether to halt on incorrect

    def __call__(self, solution: Union[float, str], answer: Union[float, str],
                 msg: str, rel_tol: float = 0.01):
        """Grade this question, and accumulate score"""
        section, question = str(msg).split('.')
```

(continues on next page)

(continued from previous page)

```
        msg = "SOA Question " + str(msg) + ":"
        if isinstance(solution, str) or isinstance(answer, str):
            correct = (solution == answer)
        else:
            correct = math.isclose(solution, answer, rel_tol=rel_tol)
        print(msg, '[', solution, ']', answer, '*' * 10 * (1-correct))
        if section not in self.score:
            self.score[section] = {}
        self.score[section][question] = correct
        if self.terminate:
            assert correct, msg

    def summary(self):
        """Return final score and by section"""
        score = {int(k): [len(v), sum(v.values())] for k, v in self.score.items()}
        out = pd.DataFrame.from_dict(score, orient='index',
                                     columns=['questions', 'correct'])
        out.loc[0] = out.sum(axis=0)
        return out.sort_index()

autograde = Autograde(terminate=False)
```

20.1 Tables

These tables are provided in the FAM-L exam (as of Aug 2022)

1. Interest Functions at i=0.05
2. Normal Distribution Table
3. Standard Ultimate Life Table

But you actually do not need any of them here!

```
print("Interest Functions at i=0.05")
UDD.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.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  
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]
```

20.2 2 Survival models

SOA Question 2.1: (B) 2.5

1. Derive function for μ from given survival function
2. Solve for ω given μ_{65}
3. Calculate e by summation

```
def fun(omega):  # Solve first for omega, given mu_65 = 1/180
    return Lifetime(l=lambda x,s: (1 - (x+s)/omega)**0.25).mu_x(65)
omega = int(Lifetime.solve(fun, target=1/180, guess=(106, 126)))
life = Lifetime(l=lambda x,s: (1 - (x+s)/omega)**0.25, maxage=omega)
e = life.e_x(106, curtate=True)
autograde(2.5, e, 2.1)
```

```
SOA Question 2.1: [ 2.5 ] 2.4786080555423604
```

SOA Question 2.2: (D) 400

1. Calculate survival probabilities for the two scenarios
2. 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))
autograde(400, std, 2.2)
```

```
SOA Question 2.2: [ 400 ] 396.5914603215815
```

SOA Question 2.3: (A) 0.0483

1. Derive formula for f given survival function

```
B, c = 0.00027, 1.1
life = Survival(S=lambda x,s,t: \
                (math.exp(-B * c**(x+s) * (c**t - 1)/math.log(c))))
f = life.f_x(x=50, t=10)
autograde(0.0483, f, 2.3)
```

```
SOA Question 2.3: [ 0.0483 ] 0.048327399045049846
```

SOA Question 2.4: (E) 8.2

1. Derive survival probability function ${}_tp_x$ given ${}_tq_0$
2. Compute $\overset{\circ}{e}$ by integration

```
life = Lifetime(l=lambda x,s: 0. if (x+s) >= 100 else 1 - ((x+s)**2)/10000.)
e = life.e_x(75, t=10, curtate=False)
autograde(8.2, e, 2.4)
```

```
SOA Question 2.4: [ 8.2 ] 8.20952380952381
```

SOA Question 2.5: (B) 37.1

1. Solve for e_{40} from limited lifetime formula
2. Compute e_{41} using backward recursion

```
life = Recursion().set_e(25, x=60, curtate=True)\
                  .set_q(0.2, x=40, t=20)\
                  .set_q(0.003, x=40)
def fun(e):  # solve e_40 from e_40:20 = e_40 - 20_p_40 e_60
    return life.set_e(e, x=40, curtate=True)\
               .e_x(x=40, t=20, curtate=True)
e40 = life.solve(fun, target=18, guess=[36, 41])
e41 = life.e_x(41, curtate=True)
autograde(37.1, e41, 2.5)
```

```
[ Lifetime: e_41 ]
    forward e_41 = e_41:1 + p_41 e_42
       shortcut 1-year curtate e_40:1
SOA Question 2.5: [ 37.1 ] 37.11434302908726
```

SOA Question 2.6: (C) 13.3

1. Derive force of mortality function μ from given survival function

```
life = Survival(l=lambda x,s: (1 - (x+s)/60)**(1/3))
mu = 1000*life.mu_x(35)
autograde(13.3, mu, 2.6)
```

```
SOA Question 2.6: [ 13.3 ] 13.340451278922776
```

SOA Question 2.7: (B) 0.1477

1. Calculate q given survival function

```
l = lambda x,s: (1-((x+s)/250) if (x+s)<40 else 1-((x+s)/100)**2)
q = Survival(l=l).q_x(30, t=20)
autograde(0.1477, q, 2.7)
```

```
SOA Question 2.7: [ 0.1477 ] 0.1477272727272727
```

SOA Question 2.8: (C) 0.94

1. Relate p_{male} and p_{female} through the common term μ
2. Solve for p_{female} given initial and surviving proportions

```
def fun(p):  # Solve first for mu, given start and end proportions
    mu = -math.log(p)
    male = Lifetime(mu=lambda x,s: 1.5 * mu)
    female = Lifetime(mu=lambda x,s: mu)
    return (75 * female.p_x(0, t=20)) / (25 * male.p_x(0, t=20))
p = Lifetime.solve(fun, target=85/15, guess=[0.89, 0.99])
autograde(0.94, p, 2.8)
```

```
SOA Question 2.8: [ 0.94 ] 0.9383813306903798
```

20.3 3 Life tables and selection

SOA Question 3.1: (B) 117

1. Constant force interpolation of lives at integer ages

```
life = Select(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)
autograde(117, q, 3.1)
```

```
SOA Question 3.1: [ 117 ] 116.7192429022082
```

SOA Question 3.2: (D) 14.7

1. UDD $\Rightarrow \mathring{e}_x = e_x + 0.5$

2. Fill select table using curtate expectations

```
e_curtate = Select.e_curtate(e=15)
life = Select(l={65: [1000, None,],
                 66: [955, None]},
              e={65: [e_curtate, None]},
              d={65: [40, None,],
                 66: [45, None]}, udd=True).fill()
e = life.e_r(66)
autograde(14.7, e, 3.2)
```

```
SOA Question 3.2: [ 14.7 ] 14.67801047120419
```

SOA Question 3.3: (E) 1074

1. UDD interpolation of lives at integer ages.

```
life = Select(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)
autograde(1074, q, 3.3)
```

```
SOA Question 3.3: [ 1074 ] 1073.684210526316
```

SOA Question 3.4: (B) 815

1. Portfolio percentile with N=4000 with mean and variance of number of survivors, N, 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)
autograde(815, pct, 3.4)
```

```
SOA Question 3.4: [ 815 ] 815.0943255167722
```

SOA Question 3.5: (E) 106

1. Compute mortality rates by interpolating lives at integer ages, using UDD linear and constant force exponential assumptions

```
l = [99999, 88888, 77777, 66666, 55555, 44444, 33333, 22222]
a = LifeTable(l={age:l for age,l in zip(range(60, 68), l)}, udd=True)\
    .q_r(60, u=3.4, t=2.5)
b = LifeTable(l={age:l for age,l in zip(range(60, 68), l)}, udd=False)\
    .q_r(60, u=3.4, t=2.5)
autograde(106, 100000 * (a - b), 3.5)
```

```
SOA Question 3.5: [ 106 ] 106.16575827938624
```

SOA Question 3.6: (D) 15.85

1. Recursion formulas for curtate expectation

```
e = Select(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]}).fill().e_x(61)
autograde(5.85, e, 3.6)
```

```
SOA Question 3.6: [ 5.85 ] 5.846832
```

SOA Question 3.7: (b) 16.4

- Deferred mortality formula with

1. Chain rule for survival probabilities, and
2. Interpolate between integer ages with constant force assumption

```
life = Select(q={50: [.0050, .0063, .0080],
                 51: [.0060, .0073, .0090],
                 52: [.0070, .0083, .0100],
                 53: [.0080, .0093, .0110]}).fill()
q = 1000 * life.q_r(50, s=0, r=0.4, t=2.5)
autograde(16.4, q, 3.7)
```

```
SOA Question 3.7: [ 16.4 ] 16.420207214428586
```

SOA Question 3.8: (B) 1505

1. Portfolio means and variances are 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)
autograde(1505, pct, 3.8)
```

```
SOA Question 3.8: [ 1505 ] 1504.8328375406456
```

SOA Question 3.9: (E) 3850

1. Portfolio means and variances are sum of 4000 independent members' means and variances (of survival)

```
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)
autograde(3850, pct, 3.9)
```

```
SOA Question 3.9: [ 3850 ] 3850.144345130047
```

SOA Question 3.10: (C) 0.86

1. Re-interpret the problem by reversing time: survival to year 6 is calculated in reverse as discounting by the same number of years.
2. Hence survival of annual entries to year 6 is calculated by discounting annuity by 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)
autograde(0.86, L / (L + R), "3.10")
```

```
SOA Question 3.10: [ 0.86 ] 0.8578442833761983
```

SOA Question 3.11: (B) 0.03

1. Calculate mortality rate by interpolating lives assuming UDD

```
life = LifeTable(q={50//2: .02, 52//2: .04}, udd=True).fill()
q = life.q_r(50//2, t=2.5/2)
autograde(0.03, q, 3.11)
```

```
SOA Question 3.11: [ 0.03 ] 0.0298
```

SOA Question 3.12: (C) 0.055

1. Compute survival probability by interpolating lives assuming constant force

```
life = Select(l={60: [10000, 9600, 8640, 7771],
                 61: [8654, 8135, 6996, 5737],
                 62: [7119, 6549, 5501, 4016],
                 63: [5760, 4954, 3765, 2410]}, udd=False).fill()
q = life.q_r(60, s=1, t=3.5) - life.q_r(61, s=0, t=3.5)
autograde(0.055, q, 3.12)
```

```
SOA Question 3.12: [ 0.055 ] 0.05465655938591829
```

SOA Question 3.13: (B) 1.6

1. Compute curtate expectations using recursion formulas
2. Convert to complete expectation assuming UDD

```
life = Select(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]}).fill()
e = life.e_r(58, s=2)
autograde(1.6, e, 3.13)
```

```
SOA Question 3.13: [ 1.6 ] 1.6003382187147688
```

SOA Question 3.14: (C) 0.345

1. Compute mortality by interpolating lives at integer ages assuming UDD

```
life = LifeTable(l={90: 1000, 93: 825},
                 d={97: 72},
                 p={96: .2},
                 q={95: .4, 97: 1}, udd=True).fill()
q = life.q_r(90, u=93-90, t=95.5 - 93)
autograde(0.345, q, 3.14)
```

```
SOA Question 3.14: [ 0.345 ] 0.345
```

20.4 4 Insurance benefits

SOA Question 4.1: (A) 0.27212

1. Solve EPV as sum of term and deferred insurance
2. Variance is difference of second moment and first moment squared

```
life = Recursion(interest=dict(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))
autograde(0.27212, std, 4.1)
```

```
SOA Question 4.1: [ 0.27212 ] 0.2721117749374753
```

SOA Question 4.2: (D) 0.18

1. Calculate Z(t) for each half-yearly t
2. Sum the deferred mortality probabilities for periods when PV > 277000

```
life = LifeTable(q={0: .16, 1: .23},
                    interest=dict(i_m=.18, m=2),
                    udd=False).fill()
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]['p'].sum()
autograde(0.18, p, 4.2)
```

```
SOA Question 4.2: [ 0.18 ] 0.17941813045022975
```

SOA Question 4.3: (D) 0.878

1. Solve q_{61} from endowment insurance EPV formula

2. Solve $A_{60:\overline{3}|}$ with new $i = 0.045$ as EPV of endowment insurance benefits.

```
life = Recursion(interest=dict(i=0.05)).set_q(0.01, x=60)
def fun(q):    # solve for q_61
    return life.set_q(q, x=61).endowment_insurance(60, t=3)
life.solve(fun, target=0.86545, guess=0.01)
A = life.set_interest(i=0.045).endowment_insurance(60, t=3)
autograde(0.878, A, "4.3")
```

```
[ Endowment Insurance: A_60:3 ]
    backward: A_60 = qv + pvA_61
       backward: A_61 = qv + pvA_62
SOA Question 4.3: [ 0.878 ] 0.8777667236003878
```

SOA Question 4.4 (A) 0.036

1. Integrate to find EPV of Z and Z^2
2. Variance is difference of second moment and first moment squared

```
life = Insurance(f=lambda *x: 0.025,
                    maxage=40+40,
                    interest=dict(v_t=lambda t: (1 + .2*t)**(-2)))
benefit = lambda x,t: 1 + .2 * t
A1 = life.A_x(40, benefit=benefit, discrete=False)
A2 = life.A_x(40, moment=2, benefit=benefit, discrete=False)
var = life.insurance_variance(A2=A2, A1=A1)
autograde(0.036, var, 4.4)
```

```
SOA Question 4.4: [ 0.036 ] 0.03567680106032681
```

SOA Question 4.5: (C) 35200

1. Find lifetime, with UDD interpolation between integer ages, that exceeds desired mortality rate
2. Compute PV of death benefit paid at that time.

```
sult = SULT(interest=dict(delta=0.05))
Z = 100000 * sult.Z_from_prob(45, 0.95)
autograde(35200, Z, 4.5)
```

```
SOA Question 4.5: [ 35200 ] 34993.774911115455
```

SOA Question 4.6: (B) 29.85

1. Calculate adjusted mortality rates
2. Compute term insurance as EPV of benefits

```
sult = SULT()
life = LifeTable(interest=dict(i=0.05),
                    q={70+k: .95**k * sult.q_x(70+k) for k in range(3)}).fill()
A = life.term_insurance(70, t=3, b=1000)
autograde(29.85, A, 4.6)
```

```
SOA Question 4.6: [ 29.85 ] 29.84835110355902
```

SOA Question 4.7: (B) 0.06

1. Variance of pure endowment Z can be related to first moment by bernoulli shortcut formula
2. Solve for i, since p is given.

```python
def fun(i):
    life = Recursion(interest=dict(i=i), verbose=False).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, guess=[0.058, 0.066])
autograde(0.06, i, 4.7)
```

```
SOA Question 4.7: [ 0.06 ] 0.06008023738770262
```

SOA Question 4.8 (C) 191

1. Use insurance recursion with special interest rate $i = 0.04$ in first year.

```python
v_t = lambda t: 1.04**(-t) if t < 1 else 1.04**(-1) * 1.05**(-t+1)
life = SULT(interest=dict(v_t=v_t))
A = life.whole_life_insurance(50, b=1000)
autograde(191, A, 4.8)
```

```
SOA Question 4.8: [ 191 ] 191.1281281882354
```

SOA Question 4.9: (D) 0.5

1. Use whole-life, endowment insurance and term insurance relationships.

```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().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, guess=[0.35, 0.55])
autograde(0.5, A, 4.9)
```

```
[ Pure Endowment: 15_E_35 ]
    endowment - term insurance = 15_E_35
SOA Question 4.9: [ 0.5 ] 0.5
```

SOA Question 4.10: (D)

1. Draw benefit diagrams

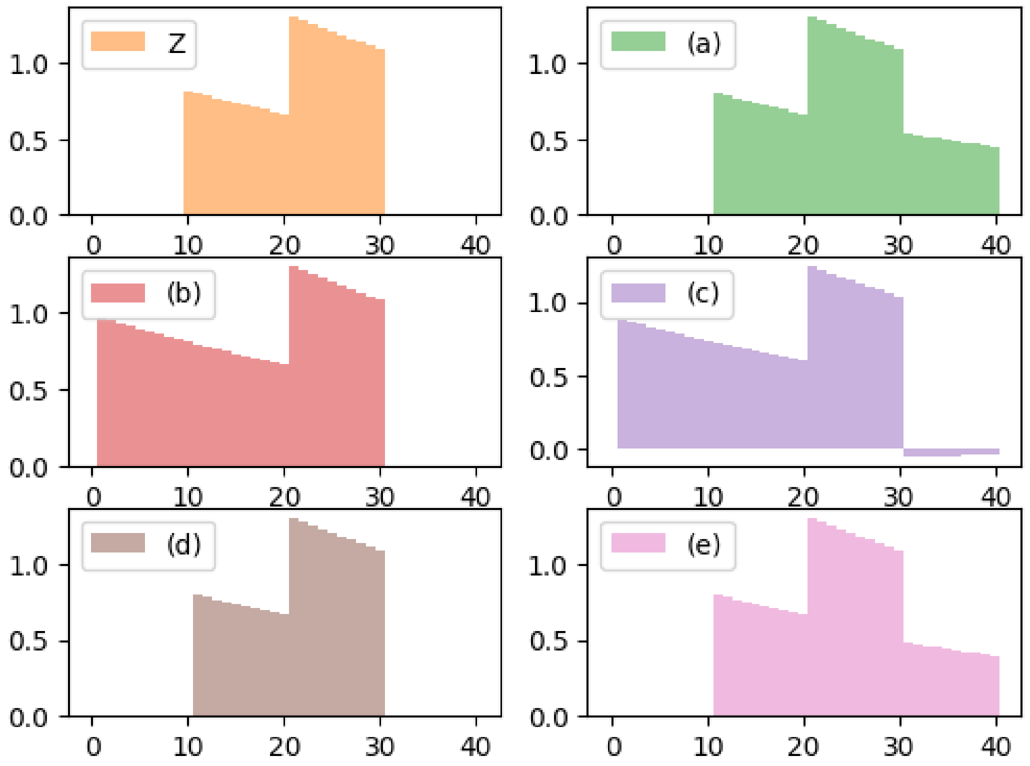
```python
life = Insurance(interest=dict(i=0.01), S=lambda x,s,t: 1, maxage=40)
def fun(x, t):
    if 10 <= t <= 20: return life.interest.v_t(t)
    elif 20 < t <= 30: return 2 * life.interest.v_t(t)
    else: return 0
def A(x, t):  # Z_x+k (t-k)
```

(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)
ax = ax.ravel()
for i, b in enumerate([fun] + benefits):
    life.Z_plot(0, benefit=b, ax=ax[i], verbose=False, color=f"C{i+1}")
    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]
autograde('D', "ABCDE"[np.argmin(z)], '4.10')
```

```
SOA Question 4.10: [ D ] D
```



SOA Question 4.11: (A) 143385

1. Recall that variance equals difference of second moment and first moment squared.
2. And endowment insurance = term insurance + pure endowment

```
A1 = 528/1000   # E[Z1]  term insurance
C1 = 0.209      # E[pure_endowment]
C2 = 0.136      # E[pure_endowment^2]
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), guess=[143400, 279300])
var = Insurance.insurance_variance(A2=A2, A1=A1, b=1000)
autograde(143385, var, 4.11)
```

```
SOA Question 4.11: [ 143385 ] 143384.99999999997
```

SOA Question 4.12: (C) 167

1. Since Z_1, Z_2 are non-overlapping, $E[Z_1\ Z_2] = 0$ for computing $Cov(Z_1, Z_2)$
2. whole life is sum of term and deferred, hence its variance is variance of components and 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)
autograde(167, var, 4.12)
```

```
SOA Question 4.12: [ 167 ] 166.82999999999998
```

SOA Question 4.13: (C) 350

1. Compute term insurance as EPV of benefits

```
life = Select(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]}, interest=dict(i=.04)).fill()
A = life.deferred_insurance(65, t=2, u=2, b=2000)
autograde(350, A, 4.13)
```

```
SOA Question 4.13: [ 350 ] 351.0578236056159
```

SOA Question 4.14: (E) 390000

1. Discounted (by interest rate $i = 0.05$) portfolio percentile of 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)
autograde(390000, F, 4.14)
```

```
SOA Question 4.14: [ 390000 ] 389322.86778416135
```

SOA Question 4.15 (E) 0.0833

1. Special benefit function has effect of reducing actuarial discount rate for constant force of mortality shortcut

```
life = Insurance(mu=lambda *x: 0.04, interest=dict(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)
autograde(0.0833, var, 4.15)
```

```
SOA Question 4.15: [ 0.0833 ] 0.08334849338238598
```

SOA Question 4.16: (D) 0.11

1. 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 = Select(q=q, interest=dict(i=.04)).fill()
A = life.term_insurance(50, t=3)
autograde(0.1116, A, 4.16)
```

```
SOA Question 4.16: [ 0.1116 ] 0.1115661982248521
```

SOA Question 4.17: (A) 1126.7

1. Find future lifetime with 50% survival probability
2. Compute EPV of special whole life as sum of term and deferred insurance with difference benefits before and after median lifetime.

```
sult = SULT()
median = sult.Z_t(48, prob=0.5, discrete=False)
benefit = lambda x,t: 5000 if t < median else 10000
A = sult.A_x(48, benefit=benefit)
autograde(1130, A, 4.17)
```

```
SOA Question 4.17: [ 1130 ] 1126.774772894844
```

SOA Question 4.18 (A) 81873

1. Find values of limits such that integral of lifetime density function equals required survival probability

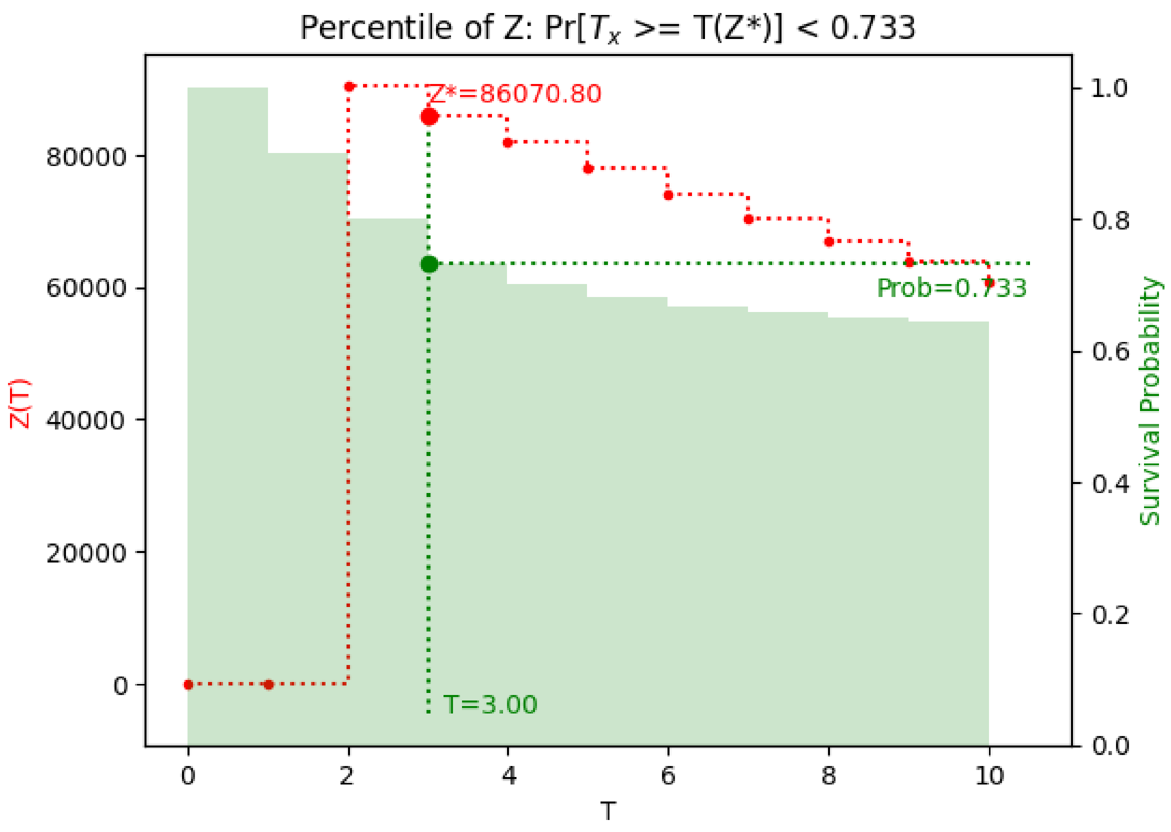
```
life = Insurance(interest=dict(delta=0.05),
                 maxage=10,
                 f=lambda x,s,t: 0.1 if t < 2 else 0.4*t**(-2))
benefit = lambda x,t: 0 if t < 2 else 100000
prob = 0.9 - life.q_x(0, t=2)
x, y = life.survival_curve()
```

(continues on next page)

(continued from previous page)

```
T = life.Z_t(0, prob=prob)
life.Z_plot(0, T=T, benefit=benefit, discrete=False, curve=(x,y))
Z = life.Z_from_t(T) * benefit(0, T)
autograde(81873, Z, 4.18)
```

```
SOA Question 4.18: [ 81873 ] 81873.07530779815
```



SOA Question 4.19: (B) 59050

1. Calculate adjusted mortality for the one-year select period
2. Compute whole life insurance using backward recursion formula

```
life = SULT()
adjust = Adjust(life=life, extra=0.8, adjust=Adjust.MULTIPLY_RATE)
select = Select(n=1)\
         .set_select(column=0, select_age=True, q=adjust.q)\
         .set_select(column=1, select_age=False, q=life['q']).fill()
A = 100000 * select.whole_life_insurance(80, s=0)
autograde(59050, A, 4.19)
```

```
SOA Question 4.19: [ 59050 ] 59050.59973285648
```

20.5 5 Annuities

SOA Question 5.1: (A) 0.705

1. Sum of annuity certain and deferred life annuity with constant force of mortality shortcut
2. Set equal to equation for PV annuity r.v. Y to infer lifetime
3. Compute survival probability from constant force of mortality function.

```
life = ConstantForce(mu=0.01, interest=dict(delta=0.06))
EY = life.certain_life_annuity(0, u=10, discrete=False)
a = life.p_x(0, t=life.Y_to_t(EY))
autograde(0.705, a, 5.1)  # 0.705
```

```
SOA Question 5.1: [ 0.705 ] 0.7053680433746505
```

SOA Question 5.2: (B) 9.64

1. Compute term life as difference of whole life and deferred insurance
2. Compute twin annuity-due, and adjust to an immediate annuity.

```
x, n = 0, 10
life = Recursion(interest=dict(i=0.05))
life.set_A(0.3, x).set_A(0.4, x+n).set_E(0.35, x, t=n)
a = life.immediate_annuity(x, t=n)
autograde(9.64, a, 5.2)
```

```
SOA Question 5.2: [ 9.64 ] 9.639999999999999
```

SOA Question 5.3: (C) 6.239

1. Required differential is the EPV of the benefit payment at the upper time limit.

```
sult = SULT()
t = 10.5
autograde(6.239, t * sult.E_r(40, t=t), 5.3)
```

```
SOA Question 5.3: [ 6.239 ] 6.23871918627528
```

SOA Question 5.4: (A) 213.7

1. Compute certain and life annuity factor as the sum of a certain annuity and a deferred life annuity.
2. Solve for amount of annual benefit that equals given EPV

```
life = ConstantForce(mu=0.02, interest=dict(delta=0.01))
P = 10000 / life.certain_life_annuity(40, u=life.e_x(40, curtate=False),
                                      discrete=False)
autograde(213.7, P, 5.4) # 213.7
```

```
SOA Question 5.4: [ 213.7 ] 213.74552118275955
```

SOA Question 5.5: (A) 1699.6

1. Adjust mortality rate for the extra risk
2. Compute annuity by backward recursion with the adjusted mortality rate or survival probability.

```
life = SULT()    # start with SULT life table
adjust = Adjust(life=life, extra=0.05, adjust=Adjust.ADD_FORCE)
select = Select(n=1)\
         .set_select(column=0, select_age=True, q=adjust.q)\
         .set_select(column=1, select_age=False, a=life['a']).fill()
a = 100 * select['a'][45][0]
autograde(1700, a, 5.5)
```

```
SOA Question 5.5: [ 1700 ] 1699.6076593190103
```

SOA Question 5.6: (D) 1200

1. Compute mean and variance of EPV of whole life annuity r.v. from whole life insurance twin and variance identities.
2. Portfolio percentile in the sum of $N = 100$ life annuity payments

```
life = Annuity(interest=dict(i=0.05))
var = life.annuity_variance(A2=0.22, A1=0.45)
mean = life.annuity_twin(A=0.45)
pct = life.portfolio_percentile(mean, var, prob=.95, N=100)
autograde(1200, pct, 5.6)
```

```
SOA Question 5.6: [ 1200 ] 1200.6946732201702
```

SOA Question 5.7: (C)

1. Compute endowment insurance from relationships of whole life, temporary and deferred insurances.
2. Compute value of temporary annuity from insurance twin
3. Apply Woolhouse approximation

```
life = Recursion(interest=dict(i=0.04))
life.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)
autograde(17376.7, a, 5.7)
```

```
[ Pure Endowment: 30_E_35 ]
    pure endowment 30_E_35 = 30_p_35 * v^30
SOA Question 5.7: [ 17376.7 ] 17376.71459632958
```

SOA Question 5.8: (C) 0.92118

1. Calculate EPV of certain and life annuity.
2. 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))
autograde(0.92118, p, 5.8)
```

```
SOA Question 5.8: [ 0.92118 ] 0.9211799771029529
```

SOA Question 5.9: (C) 0.015

1. Set EPV's expressed as forward recursions equal to each other, and solve for unknown constant k.

```
x, p = 0, 0.9  # set arbitrary p_x = 0.9
life1 = Recursion().set_a(21.854, x=x).set_p(p, x=x)
life2 = Recursion().set_a(22.167, x=x)
def fun(k):
    life2.set_p((1 + k) * p, x=x)
    return life1.whole_life_annuity(x+1) - life2.whole_life_annuity(x+1)
k = life2.solve(fun, target=0, guess=[0.005, 0.025])
autograde(0.015, k, 5.9)
```

```
SOA Question 5.9: [ 0.015 ] 0.015009110961925157
```

20.6 6 Premium Calculation

SOA Question 6.1: (D) 35.36

1. Calculate IA factor for return of premiums without interest
2. Solve for net premium such that EPV benefits = EPV premium

```
life = SULT(interest=dict(i=0.03))
P = life.net_premium(80, t=2, b=1000, return_premium=True)
autograde(35.36, P, 6.1)
```

```
SOA Question 6.1: [ 35.36 ] 35.35922286190033
```

SOA Question 6.2: (E) 3604

1. EPV return of premiums without interest = Premium $\times$ IA factor
2. 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)
autograde(3604, P, 6.2)
```

```
SOA Question 6.2: [ 3604 ] 3604.229940320728
```

SOA Question 6.3: (C) 0.390

1. Solve for lifetime t such that PV annuity certain = PV whole life annuity at age 65
2. 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)
autograde(0.39, q, 6.3)
```

```
SOA Question 6.3: [ 0.39 ] 0.39039071872030084
```

SOA Question 6.4: (E) 1890

```
mthly = Mthly(m=12, life=Reserves(interest=dict(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 = Reserves.portfolio_percentile(mean=mean, variance=var, prob=.9, N=200) / 200
autograde(1890, S, 6.4)
```

```
SOA Question 6.4: [ 1890 ] 1893.912859650868
```

SOA Question 6.5: (D) 33

```
life = SULT()
P = life.net_premium(30, b=1000)
gain = lambda k: (life.Y_x(30, t=k) * P - life.Z_x(30, t=k) * 1000)
k = min([k for k in range(20, 40) if gain(k) < 0])
autograde(33, k, 6.5)
```

```
SOA Question 6.5: [ 33 ] 33
```

SOA Question 6.6: (B) 0.79

```
life = SULT()
P = life.net_premium(62, b=10000)
policy = Policy(premium=1.03*P, renewal_policy=5,
                initial_policy=5, initial_premium=0.05, benefit=10000)
L = life.gross_policy_value(62, policy=policy)
var = life.gross_policy_variance(62, policy=policy)
prob = life.portfolio_cdf(mean=L, variance=var, value=40000, N=600)
autograde(.79, prob, 6.6)
```

```
SOA Question 6.6: [ 0.79 ] 0.7914321142683509
```

SOA Question 6.7: (C) 2880

```
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)
autograde(2880, G, 6.7)
```

```
SOA Question 6.7: [ 2880 ] 2880.2463991134578
```

SOA Question 6.8: (B) 9.5

```
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)
autograde(9.5, P, 6.8)
```

```
SOA Question 6.8: [ 9.5 ] 9.526003201821927
```

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)
autograde(647, P, 6.9)
```

```
SOA Question 6.9: [ 647 ] 646.8608151974504
```

SOA Question 6.10: (D) 0.91

```
x = 0
life = Recursion(interest=dict(i=0.06)).set_p(0.975, x=x)
a = 152.85/56.05  # solve a_x:3, given net premium and benefit APV

def fun(p):   # solve p_x+2, given a_x:3
    return life.set_p(p, x=x+1).temporary_annuity(x, t=3)
life.solve(fun, target=a, guess=0.975)

def fun(p):   # finally solve p_x+3, given A_x:3
    return life.set_p(p, x=x+2).term_insurance(x=x, t=3, b=1000)
p = life.solve(fun, target=152.85, guess=0.975)
autograde(0.91, p, "6.10")
```

```
SOA Question 6.10: [ 0.91 ] 0.90973829505257
```

SOA Question 6.11: (C) 0.041

```
life = Recursion(interest=dict(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
autograde(0.041, loss, 6.11)
```

```
[ Whole Life Insurance: A_50 ]
    backward: A_50 = qv + pvA_51
```

(continues on next page)

(continued from previous page)

```
[ Whole Life Insurance: A_50 ]
    backward: A_50 = qv + pvA_51
SOA Question 6.11: [ 0.041 ] 0.04069206883563675
```

SOA Question 6.12: (E) 88900

```
life = PolicyValues(interest=dict(i=0.06))
a = 12
A = life.insurance_twin(a)
policy = Policy(benefit=1000, settlement_policy=20,
                         initial_policy=10, initial_premium=0.75,
                         renewal_policy=2, renewal_premium=0.1)
policy.premium = life.gross_premium(A=A, a=a, **policy.premium_terms)
L = life.gross_variance_loss(A1=A, A2=0.14, policy=policy)
autograde(88900, L, 6.12)
```

```
SOA Question 6.12: [ 88900 ] 88862.59592874818
```

SOA Question 6.13: (D) -400

```
life = SULT(interest=dict(i=0.05))
A = life.whole_life_insurance(45)
policy = Policy(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, policy=policy.set(premium=P))
policy.premium = life.solve(fun, target=4953, guess=100)
L = life.gross_policy_value(45, policy=policy)
autograde(-400, L, 6.13)
```

```
SOA Question 6.13: [ -400 ] -400.94447599879277
```

SOA Question 6.14 (D) 1150

```
life = SULT(interest=dict(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)
autograde(1150, P, 6.14)
```

```
SOA Question 6.14: [ 1150 ] 1148.5800555155263
```

SOA Question 6.15: (B) 1.002

```
life = Recursion(interest=dict(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)
autograde(1.002, P, 6.15)
```

```
SOA Question 6.15: [ 1.002 ] 1.0022973504113772
```

SOA Question 6.16: (A) 2408.6

```
life = Premiums(interest=dict(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)
autograde(2410, p, 6.16)
```

```
SOA Question 6.16: [ 2410 ] 2408.575206281868
```

SOA Question 6.17: (A) -30000

```
x = 0
life = ConstantForce(mu=0.1, interest=dict(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(interest=dict(i=0.08))
life1.set_q(life.q_x(x, t=1) * 1.5, x=x, t=1)
life1.set_q(life.q_x(x+1, t=1) * 1.5, x=x+1, t=1)
policy = Policy(premium=P*2, benefit=100000, endowment=30000)
L = life1.gross_policy_value(x, t=0, n=2, policy=policy)
autograde(-30000, L, 6.17)
```

```
[ Term Insurance: A_0^1:2 ]
    backward: A_0 = qv + pvA_1
       endowment insurance - pure endowment = A_1^1:1
    pure endowment 1_E_1 = 1_p_1 * v^1
[ Temporary Annuity: a_0:2 ]
    backward: a_0:2 = 1 + E_0 a_1:1
    pure endowment 1_E_0 = 1_p_0 * v^1
       1-year discrete annuity: a_x:1 = 1
[ Pure Endowment: 2_E_0 ]
    chain Rule: 2_E_0 = E_0 * 1_E_1
    pure endowment 1_E_1 = 1_p_1 * v^1
    pure endowment 1_E_0 = 1_p_0 * v^1
SOA Question 6.17: [ -30000 ] -30107.42633581125
```

SOA Question 6.18: (D) 166400

```
life = SULT(interest=dict(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, guess=[162000, 168800])
autograde(166400, P, 6.18)
```

```
SOA Question 6.18: [ 166400 ] 166362.83871487685
```

SOA Question 6.19: (B) 0.033

```
life = SULT()
policy = Policy(initial_policy=.2, renewal_policy=.01)
a = life.whole_life_annuity(50)
A = life.whole_life_insurance(50)
policy.premium = life.gross_premium(A=A, a=a, **policy.premium_terms)
L = life.gross_policy_variance(50, policy=policy)
autograde(0.033, L, 6.19)
```

```
SOA Question 6.19: [ 0.033 ] 0.03283273381910885
```

SOA Question 6.20: (B) 459

```
life = LifeTable(interest=dict(i=0.04),
                    p={75: 0.9, 76: 0.88, 77: 0.85}).fill()
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, guess=[449, 489])
autograde(459, P, "6.20")
```

```
SOA Question 6.20: [ 459 ] 458.83181728297353
```

SOA Question 6.21: (C) 100

```
life = Recursion(interest=dict(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, guess=(80, 120))
autograde(100, P, 6.21)
```

```
SOA Question 6.21: [ 100 ] 100.85470085470084
```

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)
P = life.gross_premium(A=A, a=a, benefit=100000) / 12
autograde(102, P, 6.22)
```

```
SOA Question 6.22: [ 102 ] 102.40668704849178
```

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, guess=[30.3, 49.5])
autograde(44.7, P, 6.23)
```

```
SOA Question 6.23: [ 44.7 ] 44.70806635781144
```

SOA Question 6.24: (E) 0.30

```
life = PolicyValues(interest=dict(delta=0.07))
x, A1 = 0, 0.30   # Policy for first insurance
P = life.premium_equivalence(A=A1, discrete=False)  # Need its premium
policy = Policy(premium=P, discrete=False)
def fun(A2):  # Solve for A2, given Var(Loss)
    return life.gross_variance_loss(A1=A1, A2=A2, policy=policy)
A2 = life.solve(fun, target=0.18, guess=0.18)

policy = Policy(premium=0.06, discrete=False) # Solve second insurance
var = life.gross_variance_loss(A1=A1, A2=A2, policy=policy)
autograde(0.304, var, 6.24)
```

```
SOA Question 6.24: [ 0.304 ] 0.30419999999999975
```

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)
def fun(P):
    return life.gross_future_loss(A=benefits + expenses, a=payments,
                                  policy=Policy(premium=P))
P = life.solve(fun, target=-800, guess=[12110, 12550])
autograde(12330, P, 6.25)
```

```
SOA Question 6.25: [ 12330 ] 12325.781125438532
```

SOA Question 6.26 (D) 180

```
life = SULT(interest=dict(i=0.05))
def fun(P):
    return P - life.net_premium(90, b=1000, initial_cost=P)
P = life.solve(fun, target=0, guess=[150, 190])
autograde(180, P, 6.26)
```

```
SOA Question 6.26: [ 180 ] 180.03164891315885
```

SOA Question 6.27: (D) 10310

```
life = ConstantForce(mu=0.03, interest=dict(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
autograde(10310, P, 6.27)
```

```
SOA Question 6.27: [ 10310 ] 10309.617799001708
```

SOA Question 6.28 (B) 36

```
life = SULT(interest=dict(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)
autograde(36, P, 6.28)
```

```
SOA Question 6.28: [ 36 ] 35.72634219391481
```

SOA Question 6.29 (B) 20.5

```
life = Premiums(interest=dict(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, guess=[20, 22])
autograde(20.5, a, 6.29)
```

```
SOA Question 6.29: [ 20.5 ] 20.480268314431726
```

SOA Question 6.30: (A) 900

```
life = PolicyValues(interest=dict(i=0.04))
policy = Policy(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, policy=policy)
autograde(900, var, "6.30")
```

```
SOA Question 6.30: [ 900 ] 908.141412994607
```

SOA Question 6.31: (D) 1330

```
life = ConstantForce(mu=0.01, interest=dict(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)
autograde(1330, P, 6.31)
```

```
SOA Question 6.31: [ 1330 ] 1326.5406293909457
```

SOA Question 6.32: (C) 550

```
x = 0
life = Recursion(interest=dict(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
autograde(550, P, 6.32)
```

```
SOA Question 6.32: [ 550 ] 550.4356936711871
```

SOA Question 6.33: (B) 0.13

```
life = Insurance(mu=lambda x,t: 0.02*t, interest=dict(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)
autograde(0.13, p, 6.33, rel_tol=0.02)
```

```
SOA Question 6.33: [ 0.13 ] 0.12828940905648634
```

SOA Question 6.34: (A) 23300

```
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, guess=[23300, 23700])
autograde(23300, b, 6.34)
```

```
SOA Question 6.34: [ 23300 ] 23294.288659265632
```

SOA Question 6.35: (D) 530

```
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)
autograde(530, P, 6.35)
```

```
SOA Question 6.35: [ 530 ] 534.4072234303344
```

SOA Question 6.36: (B) 500

```
life = ConstantForce(mu=0.04, interest=dict(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, guess=[400, 800])
autograde(500, R, 6.36)
```

```
SOA Question 6.36: [ 500 ] 500.0
```

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)
P = (benefits + expenses) / a
autograde(820, P, 6.37)
```

```
SOA Question 6.37: [ 820 ] 819.7190338249138
```

SOA Question 6.38: (B) 11.3

```
x, n = 0, 10
life = Recursion(interest=dict(i=0.05))
life.set_A(0.192, x=x, t=n, endowment=1, discrete=False)
life.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(interest=dict(i=0.05), verbose=False)
    life.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, guess=a)   # discrete annuity
P = life.gross_premium(a=a, A=0.192, benefit=1000)
autograde(11.3, P, 6.38)
```

```
[ Temporary Annuity: a_0:10 ]
    Annuity twin: a = (1 - A) / d
SOA Question 6.38: [ 11.3 ] 11.308644185253657
```

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)
autograde(29, P, 6.39)
```

```
SOA Question 6.39: [ 29 ] 29.033866427845496
```

SOA Question 6.40: (C) 116

```
# - standard formula discounts/accumulates by too much (i should be smaller)
x = 0
life = Recursion(interest=dict(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(interest=dict(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
autograde(116, P, "6.40")
```

```
[ Whole Life Annuity: a_0 ]
    backward: a_0 = 1 + E_0 a_1
    pure endowment 1_E_0 = 1_p_0 * v^1
[ Whole Life Insurance: A_1 ]
    forward: A_1 = (A_0/v - q) / p
          forward: A_1 = (A_0/v - q) / p
                forward: A_1 = (A_0/v - q) / p
                forward: A_1 = (A_0/v - q) / p
SOA Question 6.40: [ 116 ] 116.51945397474269
```

SOA Question 6.41: (B) 1417

```
x = 0
life = LifeTable(interest=dict(i=0.05), q={x:.01, x+1:.02}).fill()
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
autograde(1417, P, 6.41)
```

```
SOA Question 6.41: [ 1417 ] 1416.9332301924137
```

SOA Question 6.42: (D) 0.113

```
x = 0
life = ConstantForce(interest=dict(delta=0.06), mu=0.06)
policy = Policy(discrete=True, premium=315.8,
                        T=3, endowment=1000, benefit=1000)
L = [life.L_from_t(t, policy=policy) 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 Question Solution incorrect treat endowment!
p = sum([q for (q, l) in zip (Q, L) if l > 0])
autograde(0.113, p, 6.42)
```

```
SOA Question 6.42: [ 0.113 ] 0.11307956328284252
```

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)
autograde(170, P, 6.43)
```

```
SOA Question 6.43: [ 170 ] 171.22371939459944
```

SOA Question 6.44: (D) 2.18

```
life = Recursion(interest=dict(i=0.05)).set_IA(0.15, x=50, t=10)
life.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)
autograde(2.2, P, 6.44)
```

```
SOA Question 6.44: [ 2.2 ] 2.183803457688809
```

SOA Question 6.45: (E) 690

```
life = SULT(udd=True)
policy = Policy(benefit=100000, premium=560, discrete=False)
p = life.L_from_prob(35, prob=0.75, policy=policy)
autograde(690, p, 6.45)
```

```
SOA Question 6.45: [ 690 ] 689.2659416264196
```

SOA Question 6.46: (E) 208

```
life = Recursion(interest=dict(i=0.05)).set_IA(0.51213, x=55, t=10)
life.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)
autograde(208, P, 6.46)
```

```
SOA Question 6.46: [ 208 ] 208.12282139036515
```

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)
autograde(66400, P, 6.47)
```

```
SOA Question 6.47: [ 66400 ] 66384.13293704337
```

SOA Question 6.48: (A) 3195 – example of deep insurance recursion

```
x = 0
life = Recursion(interest=dict(i=0.06), depth=5).set_p(.95, x=x, t=5)
life.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)
autograde(3195, P, 6.48)
```

```
[ Pure Endowment: 5_E_0 ]
    pure endowment 5_E_0 = 5_p_0 * v^5
[ Pure Endowment: 5_E_0 ]
    pure endowment 5_E_0 = 5_p_0 * v^5
[ Term Insurance: A_5^1:3 ]
    backward: A_5 = qv + pvA_6
       backward: A_6 = qv + pvA_7
          endowment insurance - pure endowment = A_7^1:1
    pure endowment 1_E_7 = 1_p_7 * v^1
[ Term Insurance: A_5^1:3 ]
    pure endowment 1_E_7 = 1_p_7 * v^1
          endowment insurance - pure endowment = A_7^1:1
       backward: A_6 = qv + pvA_7
    backward: A_5 = qv + pvA_6
SOA Question 6.48: [ 3195 ] 3195.1189176587473
```

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) / 12
autograde(86, P, 6.49)
```

```
SOA Question 6.49: [ 86 ] 85.99177833261696
```

SOA Question 6.50: (A) -47000

```
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
autograde(-47000, cash, "6.50")
```

```
SOA Question 6.50: [ -47000 ] -46948.2187697819
```

SOA Question 6.51: (D) 34700

```
life = Recursion()
life.set_DA(0.4891, x=62, t=10)
life.set_A(0.0910, x=62, t=10)
life.set_a(12.2758, x=62)
life.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)
autograde(34700, P, 6.51)
```

```
[ Increasing Insurance: IA_62:10 ]
    identity IA_62:10: (11)A - DA
SOA Question 6.51: [ 34700 ] 34687.207544453246
```

SOA Question 6.52: (D) 50.80 – hint: 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)
autograde(50.8, P, 6.52)
```

```
SOA Question 6.52: [ 50.8 ] 50.80135534704229
```

SOA Question 6.53: (D) 720

```
x = 0
life = LifeTable(interest=dict(i=0.08), q={x:.1, x+1:.1, x+2:.1}).fill()
A = life.term_insurance(x, t=3)
P = life.gross_premium(a=1, A=A, benefit=2000, initial_premium=0.35)
autograde(720, P, 6.53)
```

```
SOA Question 6.53: [ 720 ] 720.1646090534978
```

SOA Question 6.54: (A) 25440

```
life = SULT()
std = math.sqrt(life.net_policy_variance(45, b=200000))
autograde(25440, std, 6.54)
```

```
SOA Question 6.54: [ 25440 ] 25441.694847703857
```

20.7 7 Policy Values

SOA Question 7.1: (C) 11150

```
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)
autograde(11150, L, 7.1)
```

```
SOA Question 7.1: [ 11150 ] 11152.108749338717
```

SOA Question 7.2: (C) 1152

```
x = 0
life = Recursion(interest=dict(i=.1)).set_q(0.15, x=x).set_q(0.165, x=x+1)
life.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, guess=[1070, 1230])
autograde(1152, P, 7.2)
```

```
SOA Question 7.2: [ 1152 ] 1151.5151515151515
```

SOA Question 7.3: (E) 730

```
x = 0  # x=0 is (90) and interpret every 3 months as t=1 year
life = LifeTable(interest=dict(i=0.08/4),
                 l={0:1000, 1:898, 2:800, 3:706}).fill()
life.set_reserves(T=8, V={3: 753.72})
life.set_reserves(V={2: life.t_V_backward(x=0, t=2, premium=60*0.9,
                                           benefit=lambda t: 1000)})
V = life.t_V_backward(x=0, t=1, premium=0, benefit=lambda t: 1000)
autograde(730, V, 7.3)
```

```
SOA Question 7.3: [ 730 ] 729.998398765594
```

SOA Question 7.4: (B) -74 – 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
policy = Policy(benefit=1000, premium=1.02*P,
```

(continues on next page)

(continued from previous page)

```
                        renewal_policy=10, initial_policy=100)
V = life.gross_policy_value(x=40, t=1, policy=policy)
policy = Policy(benefit=4000, premium=0)
A = life.deferred_insurance(41, u=10)
V += life.gross_future_loss(A=A, a=0, policy=policy) # for deferred portion
autograde(-74, V, 7.4)
```

```
SOA Question 7.4: [ -74 ] -73.942155695248
```

SOA Question 7.5: (E) 1900

```
x = 0
life = Recursion(interest=dict(i=0.03), udd=True).set_q(0.04561, x=x+4)
life.set_reserves(T=3, V={4: 1405.08})
V = life.r_V_forward(x, s=4, r=0.5, benefit=10000, premium=647.46)
autograde(1900, V, 7.5)
```

```
SOA Question 7.5: [ 1900 ] 1901.766021537228
```

Answer 7.6: (E) -25.4

```
life = SULT()
P = life.net_premium(45, b=2000)
policy = Policy(benefit=2000, initial_premium=.25, renewal_premium=.05,
                        initial_policy=2*1.5 + 30, renewal_policy=2*.5 + 10)
G = life.gross_premium(a=life.whole_life_annuity(45), **policy.premium_terms)
gross = life.gross_policy_value(45, t=10, policy=policy.set(premium=G))
net = life.net_policy_value(45, t=10, b=2000)
V = gross - net
autograde(-25.4, V, 7.6)
```

```
SOA Question 7.6: [ -25.4 ] -25.44920289521204
```

SOA Question 7.7: (D) 1110

```
x = 0
life = Recursion(interest=dict(i=0.05)).set_A(0.4, x=x+10)
a = Woolhouse(m=12, life=life).whole_life_annuity(x+10)
policy = Policy(premium=0, benefit=10000, renewal_policy=100)
V = life.gross_future_loss(A=0.4, policy=policy.policy_renewal)
policy = Policy(premium=30*12, renewal_premium=0.05)
V += life.gross_future_loss(a=a, policy=policy.policy_renewal)
autograde(1110, V, 7.7)
```

```
SOA Question 7.7: [ 1110 ] 1107.9718253968258
```

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(interest=dict(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)
autograde(29.85, V, 7.8)
```

```
SOA Question 7.8: [ 29.85 ] 29.85469179271202
```

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)
policy = Policy(premium=253*12, endowment=100000, benefit=100000)
V = sult.gross_future_loss(A=A, a=a, policy=policy)
autograde(38100, V, 7.9)
```

```
SOA Question 7.9: [ 38100 ] 38099.62176709246
```

SOA Question 7.10: (C) -970

```
life = SULT()
G = 977.6
P = life.net_premium(45, b=100000)
policy = Policy(benefit=0, premium=G-P, renewal_policy=.02*G + 50)
V = life.gross_policy_value(45, t=5, policy=policy)
autograde(-970, V, "7.10")
```

```
SOA Question 7.10: [ -970 ] -971.8909301877826
```

SOA Question 7.11: (B) 1460

```
life = Recursion(interest=dict(i=0.05)).set_a(13.4205, x=55)
policy = Policy(benefit=10000)
def fun(P):
    return life.L_from_t(t=10, policy=policy.set(premium=P))
P = life.solve(fun, target=4450, guess=400)
V = life.gross_policy_value(45, t=10, policy=policy.set(premium=P))
autograde(1460, V, 7.11)
```

```
SOA Question 7.11: [ 1460 ] 1459.9818035330218
```

SOA Question 7.12: (E) 4.09

```
benefit = lambda k: 26 - k
x = 44
life = Recursion(interest=dict(i=0.04)).set_q(0.15, x=55)
life.set_reserves(T=25, endowment=1, V={11: 5.})
def fun(P):  # solve for net premium, from actuarialmath.actuarialmath.actuarialmath.
 ↪actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.
 ↪actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.
 ↪actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.
 ↪actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.
 ↪actuarialmath.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, guess=0.5)     # solved net premium
V = life.t_V(x, t=12, premium=P, benefit=benefit)   # recursion formula
autograde(4.09, V, 7.12)
```

```
SOA Question 7.12: [ 4.09 ] 4.089411764705883
```

Answer 7.13: (A) 180

```
life = SULT()
V = life.FPT_policy_value(40, t=10, n=30, endowment=1000, b=1000)
autograde(180, V, 7.13)
```

```
SOA Question 7.13: [ 180 ] 180.1071785904076
```

SOA Question 7.14: (A) 2200

```
x = 45
life = Recursion(interest=dict(i=0.05)).set_q(0.009, x=50)
life.set_reserves(T=10, V={5: 5500})
def fun(P):  # solve for net premium, from actuarialmath.actuarialmath.actuarialmath.
↪actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.
↪actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.
↪actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.
↪actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.actuarialmath.
↪actuarialmath.year 6 reserve
    return life.t_V(x=x, t=6, premium=P*0.96 - 50,
                    benefit=lambda t: 100000 + 200)
P = life.solve(fun, target=7100, guess=[2200, 2400])
autograde(2200, P, 7.14)
```

```
SOA Question 7.14: [ 2200 ] 2197.8174603174602
```

SOA Question 7.15: (E) 50.91

```
x = 0
life = Recursion(udd=True, interest=dict(i=0.05)).set_q(0.1, x=x+15)
life.set_reserves(T=3, V={16: 49.78})
V = life.r_V_backward(x, s=15, r=0.6, benefit=100)
autograde(50.91, V, 7.15)
```

```
SOA Question 7.15: [ 50.91 ] 50.91362826922369
```

SOA Question 7.16: (D) 380

```
life = Select(interest=dict(v=.95), A={86: [683/1000]},
                q={80+k: [.01*(k+1)] for k in range(6)}).fill()
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,
```

(continues on next page)

(continued from previous page)

```
                            policy=Policy(benefit=1000, premium=130))
autograde(380, V, 7.16)
```

```
SOA Question 7.16: [ 380 ] 381.6876905200001
```

SOA Question 7.17: (D) 1.018

```
x = 0
life = Recursion(interest=dict(v=math.sqrt(0.90703)))
life.set_q(0.02067, x=x+10)
life.set_A(0.52536, x=x+11)
life.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))
autograde(1.018, ratio, 7.17)
```

```
[ Whole Life Insurance: A_10 ]
    backward: A_10 = qv + pvA_11
[ Whole Life Insurance: A_10 ]
    backward: A_10 = qv + pvA_11
SOA Question 7.17: [ 1.018 ] 1.0182465434445054
```

SOA Question 7.18: (A) 17.1

```
x = 10
life = Recursion(interest=dict(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, guess=[17.1, 19.1])
autograde(17.1, a, 7.18)
```

```
SOA Question 7.18: [ 17.1 ] 17.07941929974385
```

SOA Question 7.19: (D) 720

```
life = SULT()
policy = Policy(benefit=100000, initial_policy=300, initial_premium=.5,
                        renewal_premium=.1)
P = life.gross_premium(A=life.whole_life_insurance(40),
                        **policy.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, policy=policy.set(premium=P).policy_renewal)
autograde(720, V, 7.19)
```

```
SOA Question 7.19: [ 720 ] 722.7510208759086
```

SOA Question 7.20: (E) -277.23

```
life = SULT()
S = life.FPT_policy_value(35, t=1, b=1000)  # is 0 for FPT at t=0,1
policy = Policy(benefit=1000, initial_premium=.3, initial_policy=300,
                renewal_premium=.04, renewal_policy=30)
P = life.gross_premium(A=life.whole_life_insurance(35),
                **policy.premium_terms)
R = life.gross_policy_value(35, t=1, policy=policy.set(premium=P))
autograde(-277.23, R - S, "7.20")
```

```
SOA Question 7.20: [ -277.23 ] -277.19303323929216
```

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),
                A=life.deferred_annuity(x, u=u), benefit=1000)
policy = Policy(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, policy=policy)
autograde(11866, V, 7.21)
```

```
SOA Question 7.21: [ 11866 ] 11866.30158100453
```

SOA Question 7.22: (C) 46.24

```
life = PolicyValues(interest=dict(i=0.06))
policy = Policy(benefit=8, premium=1.250)
def fun(A2):
    return life.gross_variance_loss(A1=0, A2=A2, policy=policy)
A2 = life.solve(fun, target=20.55, guess=20.55/8**2)
policy = Policy(benefit=12, premium=1.875)
var = life.gross_variance_loss(A1=0, A2=A2, policy=policy)
autograde(46.2, var, 7.22)
```

```
SOA Question 7.22: [ 46.2 ] 46.2375
```

SOA Question 7.23: (D) 233

```
life = Recursion(interest=dict(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, guess=[140, 260])
autograde(233, beta, 7.23)
```

```
[ Term Insurance: A_25^1:1 ]
    endowment insurance - pure endowment = A_25^1:1
    pure endowment 1_E_25 = 1_p_25 * v^1
SOA Question 7.23: [ 233 ] 232.64747466274176
```

SOA Question 7.24: (C) 680

```
life = SULT()
P = life.premium_equivalence(A=life.whole_life_insurance(50), b=1000000)
autograde(680, 11800 - P, 7.24)
```

```
SOA Question 7.24: [ 680 ] 680.291823645397
```

SOA Question 7.25: (B) 3947.37

```
life = Select(interest=dict(i=.04), 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)
autograde(3950, V, 7.25)
```

```
SOA Question 7.25: [ 3950 ] 3947.3684210526353
```

SOA Question 7.26: (D) 28540 – backward-forward reserve recursion

```
x = 0
life = Recursion(interest=dict(i=.05)).set_p(0.85, x=x).set_p(0.85, x=x+1)
life.set_reserves(T=2, endowment=50000)
benefit = lambda k: 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, guess=[27650, 28730])
autograde(28540, P, 7.26)
```

```
SOA Question 7.26: [ 28540 ] 28542.392566782808
```

SOA Question 7.27: (B) 213

```
x = 0
life = Recursion(interest=dict(i=0.03)).set_q(0.008, x=x)
life.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, guess=[200, 252])
autograde(213, G, 7.27)
```

```
SOA Question 7.27: [ 213 ] 212.970355987055
```

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
autograde(24.3, PW - P, 7.28)
```

```
SOA Question 7.28: [ 24.3 ] 24.334725400123975
```

SOA Question 7.29: (E) 2270

```
x = 0
life = Recursion(interest=dict(i=0.04)).set_a(14.8, x=x)\
                                       .set_a(11.4, x=x+10)
def fun(B):   # Solve for benefit B given net 10_V = 2290
    return life.net_policy_value(x, t=10, b=B)
B = life.solve(fun, target=2290, guess=2290*10)
policy = Policy(initial_policy=30, renewal_policy=5, benefit=B)
G = life.gross_premium(a=life.whole_life_annuity(x), **policy.premium_terms)
V = life.gross_policy_value(x, t=10, policy=policy.set(premium=G))
autograde(2270, V, 7.29)
```

```
SOA Question 7.29: [ 2270 ] 2270.743243243244
```

SOA Question 7.30: (E) 9035

```
policy = Policy(premium=0, benefit=10000)  # premiums=0 after t=10
L = SULT().gross_policy_value(35, policy=policy)
V = SULT(interest=dict(i=0)).gross_policy_value(35, policy=policy) # 10000
autograde(9035, V - L, "7.30")
```

```
SOA Question 7.30: [ 9035 ] 9034.654127845053
```

SOA Question 7.31: (E) 0.310

```
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=x, t=2, premium=G-P, benefit=lambda t: 0,
                           per_policy=5 + .08*G)
P = life.solve(fun, target=-23.64, guess=[.29, .31]) / 1000
autograde(0.310, P, 7.31)
```

```
SOA Question 7.31: [ 0.31 ] 0.309966
```

SOA Question 7.32: (B) 1.4

```
life = PolicyValues(interest=dict(i=0.06))
policy = Policy(benefit=1, premium=0.1)
def fun(A2):
    return life.gross_variance_loss(A1=0, A2=A2, policy=policy)
A2 = life.solve(fun, target=0.455, guess=0.455)
policy = Policy(benefit=2, premium=0.16)
var = life.gross_variance_loss(A1=0, A2=A2, policy=policy)
autograde(1.39, var, 7.32)
```

```
SOA Question 7.32: [ 1.39 ] 1.3848168384380901
```

Final Score

```
from datetime import datetime
print(datetime.now())

autograde.summary()
```

```
2022-12-01 18:32:04.263551
```

```
   questions  correct
0        136      136
2          8        8
3         14       14
4         19       19
5          9        9
6         54       54
7         32       32
```