Klugman, Panjer and Willmot on aggregate loss models

Klugman, Panjer and Willmot (2019), Loss Models: From Data to Decisions, fifth edition, Chapter 9

Stephen J. Mildenhall

2026-10-22 19:57 commit b7145b5+

Examples from the text Klugman et al. (2019), Loss Models: from data to decisions. The Loss models book is used as a text for several actuarial society exams and many college courses. KPW is shorthand for Loss Models.

1 Method of Moments Approximations, Examples 9.3 and 9.4

The observed mean (and standard deviation) of the number of claims and the individual losses over the past 10 months are 6.7 (2.3) and 179,247 (52,141), respectively. Determine the mean and standard deviation of aggregate claims per month.

```
from aggregate import qd, mv, MomentAggregator, round_bucket
import scipy.stats as ss
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

moms = MomentAggregator.agg_from_fs2(6.7, 2.3**2, 179247, 52141**2)
moms
```

```
ex      1.200955e+06
var     1.881802e+11
sd      4.337974e+05
cv      3.612104e-01
dtype: float64
```

Using normal and lognormal distributions as approximating distributions for aggregate claims, calculate the probability that claims will exceed 140% of expected costs.

```
fzn = ss.norm(loc=moms.ex, scale=moms.sd)
sigma = np.sqrt(np.log(moms.cv**2 + 1))
fzl = ss.lognorm(sigma, scale=moms.ex*np.exp(-sigma**2/2))
fzn.sf(1.4 * moms.ex), fzl.sf(1.4 * moms.ex)
```

```
(np.float64(0.1340631332467369), np.float64(0.1279965072394511))
```

Notes.

1. How to make the lognormal...

2 Group Dental Insurance, Examples 9.5, 9.6

Under a group dental insurance plan covering employees and their families, the premium for each married employee is the same regardless of the number of family members. The insurance company has compiled statistics showing that the annual cost of dental care per person for the benefits provided by the plan has the distribution (given in units of 25) on the left.

Furthermore, the distribution of the number of persons per insurance certificate (i.e. per employee) receiving dental care in any year has the distribution on the right.

Determine the mean and standard deviation of total payments per employee.

x	$f(x)$	n	$\Pr(N = n)$
1	0.150	0	0.05
2	0.200	1	0.10
3	0.250	2	0.15
4	0.125	3	0.20
5	0.075	4	0.25
6	0.050	5	0.15
7	0.050	6	0.06
8	0.050	7	0.03
9	0.025	8	0.01
10	0.025		

```
kpw_9_5 = build('agg KPW.95 '
                'dfreq  [0:8] [0.05, 0.1, 0.15, 0.2, 0.25, 0.15, 0.06, 0.03, 0.01] '
                'dsev [1:10] [0.15, 0.2, 0.25, 0.125, 0.075, 0.05, 0.05, 0.05, 0.025, 0.025]')

qd(kpw_9_5)
mv(kpw_9_5)
```

```
Aggregate object: KPW.95. Frequency distribution empirical. Severity
dhistogram, [1, 10], bounded; 10 atoms [1 2 3 ... 9 10] on [1, 10].
Updated with bucket size 1 and log2 = 7. Validation: not unreasonable.

      Mean     SD      CV     Skew P01 Median P99
X
Freq   3.4 1.7205 0.50602 0.063622
Sev    3.7 2.3152 0.62572   1.0074   1      3  10
Agg  12.58 7.6646 0.60927  0.52196   0     12  33
mean     = 12.58
variance = 58.7464
std dev  = 7.66462
```

The probability distributions are in the `density_df` dataframe.

```
with pd.option_context('display.max_rows', 360, 'display.max_columns', 10,
                       'display.width', 150,
                       'display.float_format', lambda x: f'{x:.5g}'):
    print(kpw_9_5.density_df.query('p > 0')[['p', 'F', 'S']])
```

```
              p        F          S
loss
0          0.05     0.05       0.95
1         0.015    0.065      0.935
2      0.023375 0.088375    0.91163
3      0.034675  0.12305    0.87695
4      0.032577  0.15563    0.84437
5      0.035786  0.19141    0.80859
```

6	0.039808	0.23122	0.76878
7	0.043562	0.27478	0.72522
8	0.047518	0.3223	0.6777
9	0.049034	0.37133	0.62867
10	0.051898	0.42323	0.57677
11	0.051379	0.47461	0.52539
12	0.051187	0.5258	0.4742
13	0.050305	0.5761	0.4239
14	0.048182	0.62429	0.37571
15	0.045759	0.67004	0.32996
16	0.042809	0.71285	0.28715
17	0.039378	0.75223	0.24777
18	0.035746	0.78798	0.21202
19	0.031968	0.81995	0.18005
20	0.028324	0.84827	0.15173
21	0.024788	0.87306	0.12694
22	0.021491	0.89455	0.10545
23	0.018458	0.91301	0.086993
24	0.015692	0.9287	0.0713
25	0.013234	0.94193	0.058066
26	0.011076	0.95301	0.04699
27	0.0092013	0.96221	0.037789
28	0.0075941	0.96981	0.030195
29	0.0062231	0.97603	0.023972
30	0.0050662	0.98109	0.018906
31	0.0040954	0.98519	0.01481
32	0.003288	0.98848	0.011522
33	0.0026221	0.9911	0.0089001
34	0.002076	0.99318	0.0068241
35	0.001632	0.99481	0.0051921
36	0.0012732	0.99608	0.0039189
37	0.00098531	0.99707	0.0029336
38	0.00075628	0.99782	0.0021773
39	0.00057544	0.9984	0.0016019
40	0.000434	0.99883	0.0011679
41	0.00032431	0.99916	0.00084357
42	0.00024008	0.9994	0.00060349
43	0.000176	0.99957	0.00042749
44	0.00012773	0.9997	0.00029976
45	9.1748e-05	0.99979	0.00020801
46	6.52e-05	0.99986	0.00014281
47	4.5831e-05	0.9999	9.6977e-05
48	3.1858e-05	0.99993	6.5119e-05
49	2.1894e-05	0.99996	4.3225e-05
50	1.4871e-05	0.99997	2.8353e-05
51	9.9806e-06	0.99998	1.8373e-05
52	6.6161e-06	0.99999	1.1757e-05
53	4.3305e-06	0.99999	7.4261e-06
54	2.7976e-06	1	4.6285e-06
55	1.7833e-06	1	2.8452e-06
56	1.1211e-06	1	1.7241e-06
57	6.9482e-07	1	1.0292e-06
58	4.2428e-07	1	6.0496e-07
59	2.5511e-07	1	3.4985e-07
60	1.5095e-07	1	1.989e-07
61	8.7827e-08	1	1.1107e-07
62	5.0209e-08	1	6.086e-08
63	2.8179e-08	1	3.2681e-08
64	1.5508e-08	1	1.7173e-08
65	8.3573e-09	1	8.8161e-09
66	4.4032e-09	1	4.4129e-09
67	2.2637e-09	1	2.1492e-09
68	1.1334e-09	1	1.0158e-09

```
69   5.5143e-10        1 4.6435e-10
70   2.6002e-10        1 2.0433e-10
71   1.1836e-10        1 8.5974e-11
72   5.1711e-11        1 3.4264e-11
73   2.1497e-11        1 1.2767e-11
74   8.3923e-12        1 4.3748e-12
75   3.0273e-12        1 1.3475e-12
76   9.8572e-13        1 3.6171e-13
77   2.8076e-13        1 8.0935e-14
78   6.7139e-14        1 1.3767e-14
79   1.2207e-14        1 1.5543e-15
80   1.5222e-15        1          0
```

Aggregate stop loss premiums can be computed as tail integrals of the survival function. Multiply by the units, 25.

```
(kpw_9_5.density_df.S[::-1].cumsum()[::-1] * 25)[:8]
```

```
loss
0.0    314.500000
1.0    290.750000
2.0    267.375000
3.0    244.584375
4.0    222.660625
5.0    201.551289
6.0    181.336613
7.0    162.117133
Name: S, dtype: float64
```

Exercise 9.19. An insurance portfolio produces N = 0, 1, 3 claims with probabilities 0.5, 0.4, 0.1. Individual claim amounts are 1 or 10 with probability 0.9, 0.1. Individual claim amounts and N are mutually independent. Calculate the probability that the ratio of aggregate claims to expected claims will exceed 3.0.

```
kpw_9_19 = build('agg KPW.9.19 dfreq [0 1 3] [.5 .4 .1] '
                 'dsev [1 10] [.9 .1]')
qd(kpw_9_19)
m = kpw_9_19.actual_m
print(f'mean        {m:.5g}\nprobability {kpw_9_19.sf(3 * m):.4g}')
```

```
Aggregate object: KPW.9.19. Frequency distribution empirical. Severity
dhistogram, [1, 10], bounded; atoms [1 10]. Updated with bucket size 1
and log2 = 5. Validation: not unreasonable.

      Mean     SD     CV   Skew P01 Median P99
X
Freq   0.7    0.9 1.2857 1.4486
Sev    1.9    2.7 1.4211 2.6667   1      1  10
Agg   1.33 2.8332 2.1302  3.414   0      0  12
mean        1.33
probability 0.0671
```

Exercise 9.23. An individual loss distribution is normal with mean = 100 and variance = 9. The distribution for the number of claims has outcomes 0, 1, 2, 3 with probabilities 0.5, 0.2, 0.2, 0.1. Determine the probability that aggregate claims exceed 100.

```
kpw_9_23 = build('agg KPW.9.23 dfreq [0:3] [1/2 1/5 1/5 1/10] '
                 'sev 3 * norm + 100')
qd(kpw_9_23)
qd(kpw_9_23.density_df.loc[90:110:64, ['p', 'F', 'S']])
```

```
Aggregate object: KPW.9.23. Frequency distribution empirical. Severity
norm, [0, inf), super-exponential right tail. Updated with bucket size
1/32 and log2 = 16. Validation: fails sev skew.

      Mean     SD     CV    Skew    P01 Median    P99
X
Freq   0.9  1.044   1.16  0.7276
Sev    100      3   0.03       0 93.031    100 106.97
Agg     90 104.44 1.1605 0.72937      0      0 306.66

               p       F       S
loss
90.0  3.2132e-06 0.50009 0.49991
92.0  2.3742e-05 0.50078 0.49922
94.0  0.00011248 0.50461 0.49539
96.0  0.00034169 0.51841 0.48159
98.0  0.00066552 0.55083 0.44917
100.0 0.00083113 0.60042 0.39958
102.0 0.00066552 0.64983 0.35017
104.0 0.00034169 0.68193 0.31807
106.0 0.00011248 0.69551 0.30449
108.0 2.3742e-05 0.69925 0.30075
110.0 3.2132e-06 0.69992 0.30008
```

Exercise 9.24. An employer self-insures a life insurance program with the following two characteristics:

1. Given that a claim has occurred, the claim amount is 2,000 with probability 0.4 and 3,000 with probability 0.6.
2. The number of claims has outcomes 0, 1, 2, 3, 4 with probabilities 1/16, 1/4, 3/8, 1/4, 1/16.

The employer purchases aggregate stop-loss coverage that limits the employer's annual claims cost to 5,000. The aggregate stop-loss coverage costs 1,472. Determine the employer's expected annual cost of the program, including the cost of stop-loss coverage.

```
kpw_9_24 = build('agg KPW.9.24 dfreq [0:4] [1/16 1/4 3/8 1/4 1/16] '
                 'dsev [2 3] [0.4 0.6] '
                 'aggregate net of inf xs 5')
qd(kpw_9_24)

net = kpw_9_24.summary_df.loc['Agg', 'Mean']
print(f'\ngross loss    {kpw_9_24.actual_m:.5g}\nretained loss {net:.5g}\n'
      f'premium       {net + 1.472:.5g}')
```

```
Aggregate object: KPW.9.24. Frequency distribution empirical. Severity
dhistogram, [2, 3], bounded; atoms [2 3]. Updated with bucket size 1 and
log2 = 4. Validation: reinsurance; subject not unreasonable.

       Mean     SD      CV     Skew P01 Median P99
X
Freq      2      1     0.5        0
Sev     2.6 0.4899 0.18842 -0.40825   2      3   3
Agg  4.0275 1.4754 0.36633  -1.4019   0      5   5

gross loss    5.2
retained loss 4.0275
premium       5.4995
```

Working in thousands.

Exercise 9.31. Medical and dental claims are assumed to be independent with compound Poisson distributions as follows:

- Medical claims 2 expected claims, amounts uniform (0, 1000)
- Dental claims 3 expected claims, amounts uniform (0, 200)

Let X be the amount of a given claim under a policy that covers both medical and dental claims. Determine E[(X − 100)+], the expected cost (in excess of 100) of any given claim.

```
kpw_9_31 = build('agg KPW.9.31 [2 3] claims '
                 'sev [1000 200] * uniform '
                 'occurrence ceded to inf xs 100 '
                 'poisson')
qd(kpw_9_31)
qd(kpw_9_31.reins_summary_df)
```

```
Aggregate object: KPW.9.31. Frequency distribution poisson. Mixed
severity with 2 components. Updated with bucket size 1/2 and log2 = 16.
Validation: reinsurance; subject not unreasonable.

      Mean     SD      CV    Skew P01 Median    P99
X
Freq     5 2.2361 0.44721 0.44721
Sev    177 258.59   1.461  1.4244   0     47    875
Agg    885 700.71 0.79177 0.95458   0  786.5 2952.5

                        EX  Est EX  Change EX      CV  Est CV  Change CV      Sk  Est Sk
stage view  component
occ   gross freq         5     NaN        NaN 0.44721     NaN        NaN 0.44721     NaN
            sev        260     260          0  1.0444  1.0444          0  1.3042  1.3042
            agg       1300    1300 -1.285e-12 0.64664 0.64664 -1.083e-11 0.85178 0.85178
      ceded freq         5       5          0 0.44721     NaN        NaN 0.44721     NaN
            sev        260     177   -0.31923  1.0444   1.461     0.3989  1.3042  1.4244
            agg       1300     885   -0.31923 0.64664 0.79177    0.22443 0.85178 0.95458
      net   freq         5       5          0 0.44721     NaN        NaN 0.44721     NaN
            sev        260      83   -0.68077  1.0444 0.35009   -0.66478  1.3042 -1.5092
            agg       1300     415   -0.68077 0.64664 0.47383   -0.26725 0.85178 0.48992
```

Could also compute impact of aggregate reinsurance structures.

Exercise 9.34. A compound Poisson distribution has 5 expected claim and claim amount distribution p(100) = 0.80, p(500) = 0.16, and p(1,000) = 0.04. Determine the probability that aggregate claims will be exactly 600.

```
kpw_9_34 = build('agg KPW.9.34 5 claims '
                 'dsev [1 5 10] [.8 .16 .04] '
                 'poisson')
qd(kpw_9_34)
print(f'{kpw_9_34.pmf(6):.6g}')
kpw_9_34.density_df.index = kpw_9_34.density_df.index.astype(int)
qd(kpw_9_34.density_df.query('p > 0.001')[['p', 'F', 'S']], accuracy=5)
```

```
Aggregate object: KPW.9.34. Frequency distribution poisson. Severity
dhistogram, [1, 10], bounded; atoms [1 5 10]. Updated with bucket size 1
and log2 = 8. Validation: not unreasonable.

      Mean     SD      CV    Skew P01 Median P99
X
Freq     5 2.2361 0.44721 0.44721
Sev      2 2.1909  1.0954  2.2822   1      1  10
```

```
Agg     10 6.6332 0.66332  1.0416   1      9  30
0.0598929

             p         F         S
loss
0    0.0067379 0.0067379   0.99326
1     0.026952   0.03369   0.96631
2     0.053904  0.087593   0.91241
3     0.071871   0.15946   0.84054
4     0.071871   0.23134   0.76866
5     0.062888   0.29422   0.70578
6     0.059893   0.35412   0.64588
7     0.065027   0.41914   0.58086
8     0.068449   0.48759   0.51241
9     0.062365   0.54996   0.45004
10    0.051448   0.60141   0.39859
11    0.045388   0.64679   0.35321
...        ...       ...       ...
23   0.0099885   0.95791  0.042093
24   0.0084688   0.96638  0.033624
25   0.0065328   0.97291  0.027091
26   0.0050132   0.97792  0.022078
27   0.0042014   0.98212  0.017877
28   0.0036975   0.98582  0.014179
29   0.0030698   0.98889  0.011109
30   0.0023339   0.99122 0.0087753
31   0.0017543   0.99298  0.007021
32   0.0014285   0.99441 0.0055925
33   0.0012267   0.99563 0.0043658
34   0.0010036   0.99664 0.0033621
```

Work in hundreds. Convert index to integer to improve display. Show all outcomes with probability greater than 0.001.

Exercise 9.35. Aggregate payments have a compound distribution. The frequency distribution is negative binomial with $r = 16$ and $\beta = 6$, and the severity distribution is uniform on the interval $(0, 8)$. Use the normal approximation to determine the premium such that the probability is 5% that aggregate payments will exceed the premium.

The negative binomial has mean $r\beta$ and variance $r\beta(1 + \beta)$. Therefore the gamma mixing variance equals $c = 1/r$ (since $r\beta(1 + \beta) = n(1 + cn)$.) Hence the mixing cv equals 0.25. The premium is the 95%ile of the aggregate distribution.

```
kpw_9_35 = build('agg KPW.9.35 96 claims '
                 'sev 8 * uniform '
                 'mixed gamma 0.25')
qd(kpw_9_35)
mv(kpw_9_35)
appx = kpw_9_35.approximate('all')
ans = {k: v.isf(0.05) for k, v in appx.items()}
ans['FFT'] = kpw_9_35.q(0.95)
qd(pd.DataFrame(ans.values(),
                index=pd.Index(ans.keys(), name='method'),
                columns=['premium']).sort_values('premium'),
  accuracy=4)
```

```
Aggregate object: KPW.9.35. Frequency distribution gamma. Severity
uniform, [0, 8], bounded. Updated with bucket size 1/16 and log2 = 16.
Validation: not unreasonable.

      Mean     SD      CV    Skew    P01 Median    P99
```

```
X
Freq    96 25.923 0.27003 0.50149
Sev      4 2.3095 0.57739       0 0.0625      4 7.9375
Agg    384 106.13 0.27639 0.50366 176.88 375.12 669.31
mean     = 384
variance = 11264
std dev  = 106.132

          premium
method
norm       558.57
slognorm   571.88
FFT        572.38
sgamma      572.4
gamma       573.6
lognorm    578.31
```

The `approximate` method returns a dictionary with key the method, for normal and shifted and unshifted gamma and lognormal.

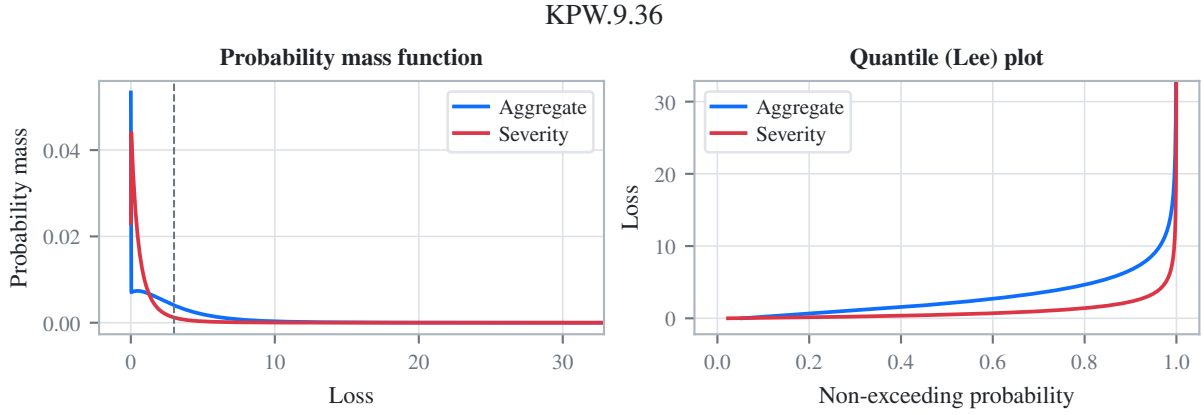
Exercise 9.36. The number of losses is Poisson with mean 3. Loss amounts have a Burr distribution with $\alpha = 3$, $\theta = 2$, and $\gamma = 1$. Determine the variance of aggregate losses.

A matter of converting parameterizations. This is the `scipy.stats` `burr12` distribution. The shape parameters are `c=gamma` and `d=alpha`. `theta` is a scale parameter.

```
kpw_9_36 = build('agg KPW.9.36 3 claims '
                 'sev 2 * burr12 1 3 '
                 'poisson')
qd(kpw_9_36)
mv(kpw_9_36)
kpw_9_36.plot();
```

```
Aggregate object: KPW.9.36. Frequency distribution poisson. Severity
burr12, [0, inf), power-law right tail. Updated with bucket size 1/32
and log2 = 16. Validation: fails sev cv, agg cv.

        Mean     SD      CV    Skew P01  Median    P99
X
Freq       3 1.7321 0.57735 0.57735
Sev  0.99994 1.7287  1.7288  21.773   0 0.53125 7.2812
Agg   2.9998 3.4591  1.1531  8.8756   0  2.0938 15.219
mean     = 3
variance = 12
std dev  = 3.4641
```



3 Compound Poisson, Example 9.9, 9.10

Policy A has a compound Poisson distribution with 2 expected claims and severity probabilities 0.6 on a payment of 1 and 0.4 on a payment of 2. Policy B has a compound Poisson distribution with 1 expected claim and probabilities 0.7 on a payment of 1 and 0.3 on a payment of 3.

Determine the probability that the total payment on the two policies will be 2.

Figure the weighted severity by hand.

```
kpw_9_9 = build('agg KPW.9.9 3 claims '
                'dsev [1 2 3] [1.9/3 .8/3 .3/3] '
                'poisson')
qd(kpw_9_9)
print(f'{kpw_9_9.pmf(2):.6g}')
kpw_9_9.density_df.index = kpw_9_9.density_df.index.astype(int)
bit = kpw_9_9.density_df.query('p > 0.001')[['p', 'F', 'S']]
bit['p*exp(3)'] = bit.p * np.exp(3)
qd(bit, accuracy=5)
```

```
Aggregate object: KPW.9.9. Frequency distribution poisson. Severity
dhistogram, [1, 3], bounded; atoms [1 2 3]. Updated with bucket size 1
and log2 = 6. Validation: not unreasonable.

       Mean      SD      CV    Skew P01 Median P99
X
Freq      3  1.7321 0.57735 0.57735
Sev  1.4667 0.66999 0.45681  1.1192   1      1   3
Agg     4.4  2.7928 0.63474 0.75284   0      4  12
0.129695

             p        F         S  p*exp(3)
loss
0     0.049787 0.049787   0.95021         1
1     0.094595  0.14438   0.85562       1.9
2       0.1297  0.27408   0.72592     2.605
3      0.14753  0.42161   0.57839    2.9632
4      0.14324  0.56484   0.43516     2.877
5      0.12498  0.68983   0.31017    2.5104
6     0.099904  0.78973   0.21027    2.0066
7     0.074101  0.86383   0.13617    1.4884
8     0.051641  0.91547  0.084527    1.0372
9     0.034066  0.94954  0.050462   0.68423
10    0.021404  0.97094  0.029058   0.42991
11    0.012877  0.98382   0.01618   0.25865
12   0.0074477  0.99127 0.0087326   0.14959
13   0.0041552  0.99542 0.0045774   0.08346
14   0.0022429  0.99767 0.0023345   0.04505
15   0.0011742  0.99884 0.0011603  0.023584
```

The last column answers Example 9.10.

Alternatively, use the `Portfolio` class.

```
p = build('port KPW.9.9.p '
          'agg A 2 claims dsev [1 2] [.6 .4] poisson '
           'agg B 1 claims dsev [1 3] [.7 .3] poisson')
qd(p)
```

```
Portfolio object: KPW.9.9.p. Contains 2 aggregate components. Updated
with bucket size 1 and log2 = 6. Validation: not unreasonable.

            Mean      SD      CV    Skew P01 Median P99
unit  X
A     Freq     2  1.4142 0.70711 0.70711
      Sev    1.4  0.4899 0.34993 0.40825   1      1   2
      Agg    2.8  2.0976 0.74915 0.82344   0      2   9
B     Freq     1       1       1       1
      Sev    1.6 0.91652 0.57282 0.87287   1      1   3
      Agg    1.6  1.8439  1.1524  1.4037   0      1   7
total Agg    4.4  2.7928 0.63474 0.75284   0      4  12
```

Exercise 9.39. For a compound distribution, frequency has a binomial distribution with parameters m = 3 and q = 0.4 and severity has an exponential distribution with a mean of 100. Calculate $\Pr(A \leq 300)$.

Assume 1.2 expected claims. Work in hundreds.

```
kpw_9_39 = build('agg KPW.9.39 1.2 claims '
                 'sev expon binomial 0.4')
qd(kpw_9_39)
print(f'probability = {kpw_9_39.cdf(3):.6g}')
```

```
Aggregate object: KPW.9.39. Frequency distribution binomial. Severity
expon, [0, inf), exponential right tail. Updated with bucket size 1/512
and log2 = 16. Validation: not unreasonable.

      Mean      SD      CV   Skew       P01  Median    P99
X
Freq   1.2 0.84853 0.70711 0.2357
Sev      1       1       1      2 0.0097656 0.69336 4.6055
Agg    1.2  1.3856  1.1547 1.7681         0 0.76172 6.0039
probability = 0.89413
```

Exercise 9.40. A company sells three policies. For policy A, all claim payments are 10,000 and a single policy has a Poisson number of claims with mean 0.01. For policy B, all claim payments are 20,000 and a single policy has a Poisson number of claims with mean 0.02. For policy C, all claim payments are 40,000 and a single policy has a Poisson number of claims with mean 0.03. All policies are independent. For the coming year, there are 5,000, 3,000, and 1,000 of policies A, B, and C, respectively. Calculate the expected total payment, the standard deviation of total payment, and the probability that total payments will exceed 30,000.

Must use a `Portfolio`. Work in thousands.

```
kpw_9_40 = build('port kpw_9_40\n'
                 '\tagg A 50 claims dsev [10] poisson\n'
                 '\tagg B 60 claims dsev [20] poisson\n'
                 '\tagg C 30 claims dsev [40] poisson\n')
qd(kpw_9_40)
qd(pd.Series({'expected payment': kpw_9_40.actual_m,
              'sd payment': kpw_9_40.actual_sd,
              'Pr > 3000': kpw_9_40.sf(3000)}).to_frame('value'),
              accuracy=5)
```

```
Portfolio object: kpw_9_40. Contains 3 aggregate components. Updated
with bucket size 10 and log2 = 9. Validation: not unreasonable.

            Mean     SD       CV    Skew  P01 Median  P99
unit  X
```

```
A     Freq    50 7.0711  0.14142 0.14142
      Sev     10      0        0           10     10   10
      Agg    500 70.711  0.14142 0.14142  340    500  670
B     Freq    60  7.746   0.1291  0.1291
      Sev     20      0        0           20     20   20
      Agg   1200 154.92   0.1291  0.1291  860   1200 1580
C     Freq    30 5.4772  0.18257 0.18257
      Sev     40      0        0           40     40   40
      Agg   1200 219.09  0.18257 0.18257  720   1200 1720
total Agg   2900 277.49 0.095686 0.11466 2280   2890 3570

                   value
expected payment    2900
sd payment        277.49
Pr > 3000        0.34658
```

4 Zero-Modified Binomial, Example 9.11

A compound distribution has a zero-modified binomial distribution with $m = 3$, $q = 0.3$, and $p_0^M = 0.4$. Individual payments are 0, 50, and 150, with probabilities 0.3, 0.5, and 0.2, respectively. Use the recursive formula to determine the probability distribution of S.

Solution. `aggregate` parameterizes the binomial by the per-trial probability `q` and the *mean*, so $m = 3$ trials at $q = 0.3$ is entered as $mq = 0.9$ claims. That figure is the **un-modified** base mean: `zm 0.4` then reweights the base binomial, exactly as in the $(a, b, 1)$ construction of §6.6, and the realized $\mathsf{E}(N)$ follows from it rather than being pinned to 0.9.

```
kpw_9_11 = build('agg KPW.9.11 0.9 claims '
                 'dsev [0 50 150] [.3 .5 .2] '
                 'binomial 0.3 zm 0.4')
qd(kpw_9_11)
```

```
Aggregate object: KPW.9.11. Frequency distribution binomial. Severity
dhistogram, [0, 150], bounded; atoms [0 50 150]. Updated with bucket
size 50 and log2 = 5. Validation: not unreasonable.

      Mean      SD      CV    Skew P01 Median P99
X
Freq   0.9 0.89452 0.99392 0.68232
Sev     55  52.202 0.94912 0.85413   0     50 150
Agg   49.5  69.807  1.4102   1.618   0      0 300
```

The count distribution is the reweighted binomial, with $p_k^M = \frac{1-p_0^M}{1-p_0}\, p_k$ for $k \geq 1$ and $p_0 = 0.7^3 = 0.343$:

```
qd(kpw_9_11.frequency.freq_df.head(4), accuracy=6)
```

```
         p     po_p
n
0      0.4  0.40657
1  0.35221  0.36591
2  0.19852  0.16466
3 0.046234 0.049398
```

and the aggregate is exact: the severity lattice is a multiple of 50, so the FFT reproduces the recursion to machine precision:

```
qd(kpw_9_11.density_df.query('p_total > 1e-12').p_total.to_frame(), accuracy=6)
```

```
          p_total
loss
0.0        0.5248
50.0      0.24207
100.0    0.060448
150.0     0.10307
200.0    0.048545
250.0   0.0074819
300.0   0.0099703
350.0   0.0029886
400.0  0.00017941
450.0  0.00039496
500.0  4.8045e-05
600.0   4.925e-06
700.0   4.574e-08
800.0  8.0664e-09
900.0  8.3044e-10
1000.0 8.2033e-11
1100.0 1.2971e-11
1200.0 1.7511e-12
```

5 Extended Truncated Negative Binomial, Example 9.12

The number of claims has a Poisson–ETNB distribution with Poisson parameter $\lambda = 2$ and ETNB parameters $\beta = 3$ and $r = 0.2$. The claim size distribution has probabilities 0.3, 0.5, and 0.2 at 0, 10, and 20, respectively. Determine the total claims distribution recursively.

> **To do**
>
> Needs a **compound** frequency with a zero-truncated secondary, a Poisson stopped sum of ZT negative binomials. Zero truncation itself is available since 1.0.0a152 (see Example 9.11), and here $r = 0.2 > 0$, so this is an ordinary ZT negative binomial rather than the genuinely *extended* $-1 < r < 0$ case. What is missing is the frequency-compounding slot: `pascal` is a Poisson stopped sum of an *un-truncated* negative binomial, and DecL has no way to zero-truncate the secondary distribution of a compound frequency. Tracked separately from `[ZT-ZM-Frequency-Fix]`.

Exercise 9.45. For a compound Poisson distribution, has 6 expected claims and individual losses take values 1, 2, 4 with equal probabilities. Determine the distribution of the aggregate.

```
kpw_9_45 = build('agg KPW.9.45 6 claims '
                 'dsev [1 2 4] poisson')
qd(kpw_9_45)
qd(kpw_9_45.density_df.query('p > 0.001')[['p', 'F', 'S']], accuracy=5)
```

```
Aggregate object: KPW.9.45. Frequency distribution poisson. Severity
dhistogram, [1, 4], bounded; atoms [1 2 4]. Updated with bucket size 1
and log2 = 7. Validation: not unreasonable.

       Mean     SD      CV    Skew P01 Median P99
X
Freq      6 2.4495 0.40825 0.40825
```

```
Sev  2.3333 1.2472 0.53452  0.3818   1      2   4
Agg      14 6.4807 0.46291 0.53639   2     13  31

             p         F         S
loss
0.0  0.0024788 0.0024788   0.99752
1.0  0.0049575 0.0074363   0.99256
2.0   0.009915  0.017351   0.98265
3.0    0.01322  0.030571   0.96943
4.0   0.021483  0.052054   0.94795
5.0   0.027101  0.079155   0.92085
6.0   0.036575   0.11573   0.88427
7.0   0.041045   0.15678   0.84322
8.0   0.050031   0.20681   0.79319
9.0    0.05345   0.26026   0.73974
10.0  0.059963   0.32022   0.67978
11.0   0.06019   0.38041   0.61959
...        ...       ...       ...
24.0  0.017373   0.93502  0.064978
25.0  0.014016   0.94904  0.050962
26.0  0.011474   0.96051  0.039488
27.0 0.0090615   0.96957  0.030427
28.0 0.0072502   0.97682  0.023176
29.0 0.0056163   0.98244   0.01756
30.0 0.0044008   0.98684  0.013159
31.0 0.0033471   0.99019 0.0098123
32.0 0.0025718   0.99276 0.0072405
33.0 0.0019231   0.99468 0.0053174
34.0 0.0014512   0.99613 0.0038662
35.0 0.0010677    0.9972 0.0027985
```

Exercise 9.47. Aggregate claims are compound Poisson with 2 expected claims and severity outcomes 1, 2 with probability 1/4 and 3/4. For a premium of 6, an insurer covers aggregate claims and agrees to pay a dividend (a refund of premium) equal to the excess, if any, of 75% of the premium over 100% of the claims. Determine the excess of premium over expected claims and dividends.

```
kpw_9_47 = build('agg KPW.9.47 2 claims '
                 'dsev [1 2] [1/4 3/4] poisson')
qd(kpw_9_47)

bit = kpw_9_47.density_df.query('p > 0')[['p', 'F', 'S']]
bit['dividend'] = np.maximum(0.75 * 6 - bit.index, 0)
qd(bit.head(10), accuracy=4)
exp_div = (bit.dividend * bit.p).sum()
print(f'prem      = {6:.5g}\n'
      f'exp loss  = {kpw_9_47.actual_m:.5g}\n'
      f'dividend  = {exp_div:.5g}\n'
      f'excess    = {6 - kpw_9_47.actual_m - exp_div:.5g}')
```

```
Aggregate object: KPW.9.47. Frequency distribution poisson. Severity
dhistogram, [1, 2], bounded; atoms [1 2]. Updated with bucket size 1 and
log2 = 5. Validation: not unreasonable.

      Mean      SD      CV    Skew P01 Median P99
X
Freq     2  1.4142 0.70711 0.70711
Sev   1.75 0.43301 0.24744 -1.1547   1      2   2
Agg    3.5  2.5495 0.72843 0.75429   0      3  11

            p       F        S  dividend
loss
0.0   0.13534 0.13534  0.86466       4.5
```

```
1.0  0.067668   0.203    0.797       3.5
2.0   0.21992 0.42292  0.57708       2.5
3.0   0.10432 0.52724  0.47276       1.5
4.0   0.17798 0.70522  0.29478       0.5
5.0  0.080391 0.78561  0.21439         0
6.0  0.095689  0.8813   0.1187         0
7.0  0.041288 0.92259 0.077408         0
8.0  0.038464 0.96106 0.038945         0
9.0    0.0159 0.97696 0.023045         0
prem      = 6
exp loss  = 3.5
dividend  = 1.6411
excess    = 0.85888
```

Exercise 9.57, 9.58. Aggregate losses have a compound Poisson claim distribution with 3 expected claims and individual claim amount distribution p(1) = 0.4, p(2) = 0.3, p(3) = 0.2, and p(4) = 0.1. Determine the probability that aggregate losses do not exceed 3.

Repeat the Exercise with a negative binomial frequency distribution with r = 6 and $\beta = 0.5$.

```
kpw_9_57 = build('agg KPW.9.57 3 claims '
                 'dsev [1:4] [.4 .3 .2 .1] poisson')
qd(kpw_9_57)
kpw_9_58 = build('agg KPW.9.58 3 claims '
                 'dsev [1:4] [.4 .3 .2 .1] mixed gamma 6**-0.5')
qd(kpw_9_58)

bit = pd.concat((kpw_9_57.density_df[['p', 'F', 'S']],
                 kpw_9_58.density_df[['p', 'F', 'S']]),
                keys=('Po', 'NB'), axis=1)
qd(bit.head(16), accuracy=5)
```

```
Aggregate object: KPW.9.57. Frequency distribution poisson. Severity
dhistogram, [1, 4], bounded; atoms [1 2 3 4]. Updated with bucket size 1
and log2 = 6. Validation: not unreasonable.

      Mean     SD      CV    Skew P01 Median P99
X
Freq     3 1.7321 0.57735 0.57735
Sev      2      1     0.5     0.6   1      2   4
Agg      6  3.873  0.6455 0.75394   0      5  17
Aggregate object: KPW.9.58. Frequency distribution gamma. Severity
dhistogram, [1, 4], bounded; atoms [1 2 3 4]. Updated with bucket size 1
and log2 = 7. Validation: not unreasonable.

      Mean     SD      CV    Skew P01 Median P99
X
Freq     3 2.1213 0.70711 0.94281
Sev      2      1     0.5     0.6   1      2   4
Agg      6 4.5826 0.76376  1.0474   0      5  20

            Po                         NB
             p        F        S        p        F        S
loss
0.0   0.049787 0.049787  0.95021 0.087791 0.087791  0.91221
1.0   0.059744  0.10953  0.89047 0.070233  0.15802  0.84198
2.0   0.080655  0.19019  0.80981  0.08545  0.24348  0.75652
3.0   0.097981  0.28817  0.71183 0.095933  0.33941  0.66059
4.0    0.10751  0.39568  0.60432 0.098486  0.43789  0.56211
5.0    0.10445  0.50013  0.49987 0.089535  0.52743  0.47257
6.0   0.098668   0.5988   0.4012 0.082877  0.61031  0.38969
7.0   0.088215  0.68701  0.31299 0.073657  0.68396  0.31604
```

```
8.0    0.07506  0.76207  0.23793 0.063257  0.74722  0.25278
9.0   0.061311  0.82338  0.17662  0.05302  0.80024  0.19976
10.0  0.048587  0.87197  0.12803 0.043819  0.84406  0.15594
11.0  0.037239  0.90921  0.09079 0.035507  0.87957  0.12043
12.0  0.027715  0.93692 0.063075 0.028316  0.90789 0.092115
13.0  0.020101  0.95703 0.042974 0.022288  0.93017 0.069827
14.0  0.014239  0.97127 0.028735 0.017338  0.94751 0.052489
15.0 0.0098562  0.98112 0.018879 0.013334  0.96085 0.039155
```

Exercise 9.59. A policy covers physical damage incurred by the trucks in a company's fleet. The number of losses in a year has a Poisson distribution with expectation 5. The amount of a single loss has a gamma distribution with shape 0.5 and scale 2,500. The insurance contract pays a maximum annual benefit of 20,000. Determine the probability that the maximum benefit will be paid. Use a span of 100 and the method of rounding.

```
kpw_9_59 = build('agg KPW.9.59 5 claims '
                 'sev 2500 * gamma 0.5 '
                 'poisson')
qd(kpw_9_59)
print(f'pr(loss >= 20000) = {kpw_9_59.sf(20000):.6g}')
```

```
Aggregate object: KPW.9.59. Frequency distribution poisson. Severity
gamma, [0, inf), exponential right tail. Updated with bucket size 4 and
log2 = 16. Validation: not unreasonable.

       Mean     SD      CV    Skew P01 Median   P99
X
Freq      5 2.2361 0.44721 0.44721
Sev    1250 1767.8  1.4142  2.8284   0    568  8292
Agg  6249.9 4841.2  0.7746   1.291  12   5196 21720
pr(loss >= 20000) = 0.0159346
```

Repeated at the requested span of 100.

```
kpw_9_59.update(log2=10, bs=100)
print(f'pr(loss >= 20000) = {kpw_9_59.sf(20000):.6g}')
```

```
pr(loss >= 20000) = 0.0157042

V:\worktrees\aggregate_REFACTOR\src\aggregate\_bucket_window.py:1430: InfeasibleGridWarning:
KPW.9.59: severity 'gamma' cannot be reproduced on this grid. The body needs bs <= 8.20968
(0.2102% of the mean is supplied below the first half bucket, and is placed at 0); the mean is not
complete until the grid reaches 28,193, so matching it to 0.0001 needs log2 = 12, and log2 = 10
was used. A thick unlimited severity has this problem at any bucket size, because its mean is
furnished far above its median and a finer bs only shortens the reach. Add an occurrence limit, or
accept the reported moment errors. Further occurrences this session are suppressed; each object
records its own verdict in validation.
  warn_infeasible_grid(agg)
```

Exercise 9.60. An individual has purchased health insurance, for which they pay 10 for each physician visit and 5 for each prescription. The probability that a payment will be 10 is 0.25, and the probability that it will be 5 is 0.75. The total number of payments per year has the Poisson–Poisson (Neyman Type A) distribution with primary mean 10 and secondary mean 4. Determine the probability that total payments in one year will exceed 400. Compare your answer to a normal approximation.

```
kpw_9_60 = build('agg KPW.9.60 40 claims '
                 'dsev [5 10] [3/4 1/4] '
                 'neyman 4')
qd(kpw_9_60)

fz = kpw_9_60.approximate('norm')
print(f'FFT            {kpw_9_60.sf(400):.5g}\n'
      f'Normal approx  {fz.sf(400):.5g}')
```

```
Aggregate object: KPW.9.60. Frequency distribution neyman. Severity
dhistogram, [5, 10], bounded; atoms [5 10]. Updated with bucket size 5
and log2 = 9. Validation: not unreasonable.

      Mean     SD      CV    Skew P01 Median P99
X
Freq    40 14.142 0.35355 0.41012
Sev   6.25 2.1651 0.34641  1.1547   5      5  10
Agg    250 89.443 0.35777 0.42101  70    245 485
FFT            0.054616
Normal approx  0.046766
```

6 Poisson Pareto, Example 9.14

The number of ground-up losses is Poisson distributed with mean 3. The individual loss distribution is Pareto with shape parameter $\alpha = 4$ and scale parameter 10. An individual ordinary deductible of 6, coinsurance of 75%, and an individual loss limit of 24 (before application of the deductible and coinsurance) are all applied. Determine the mean, variance, and distribution of aggregate payments.

The covered layer is 18 xs 6, in which the insured pays 25% because of the coinsurance clause. The severity is unconditional.

```
kpw_9_14 = build('agg KPW.9.14 3 claims '
                 '18 xs 6 '
                 'sev 10 * pareto 4 - 10 ! '
                 'occurrence net of 25% po inf xs 0 '
                 'poisson')
qd(kpw_9_14)
print(f'variance = {kpw_9_14.summary_df.loc["Agg", ["Mean", "CV"]].prod()**2:.6g}\ncomputed with
bs=1/{1/kpw_9_14.bs:.0f} and log2={kpw_9_14.log2}')
qd(kpw_9_14.density_df.loc[[0, 1, 2, 3], ['p', 'F', 'S']])
kpw_9_14.plot();
```

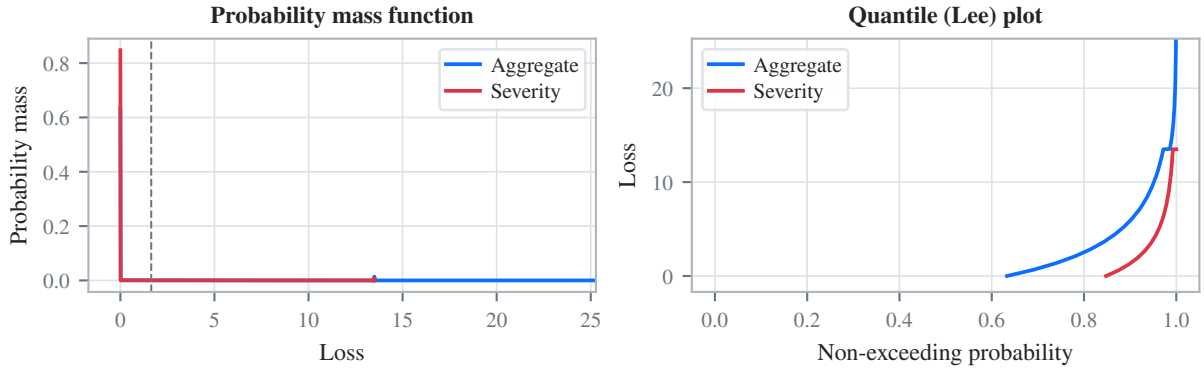
```
Aggregate object: KPW.9.14. Frequency distribution poisson. Severity
pareto, [0, 18], bounded; 18 xs 6. Updated with bucket size 1/128 and
log2 = 16. Validation: reinsurance; subject not unreasonable.

        Mean     SD      CV    Skew P01 Median    P99
X
Freq       3 1.7321 0.57735 0.57735
Sev  0.54674 1.9199  3.5115  4.6513   0      0 11.719
Agg   1.6402 3.4575   2.108  2.8396   0      0 14.648
variance = 11.9546
computed with bs=1/128 and log2=16

              p       F       S
loss
0.0     0.63312 0.63312 0.36688
1.0  0.00051313 0.71515 0.28485
```

```
2.0  0.00038181 0.77526 0.22474
3.0  0.00033695 0.82028 0.17972
```

KPW.9.14



Under reinsurance `summary_df` reports the view the program requested, net here, as `Mean` / `SD` / `CV` / `Skew` on the `Agg` row; the gross figures it started from are the `('agg', 'subject', 'agg')` row of `reins_summary_df`. The print statement computes net variance as the square of the product of the estimated mean and cv. The spikes on the density corresponds to the possibility of only limit claims.

Exercise 9.63. A ground-up model of individual losses has a gamma distribution with shape parameter 2 and scale 100. The number of losses has a negative binomial distribution with $r = 2$ and $\beta = 1.5$. An ordinary deductible of 50 and a loss limit of 175 (before imposition of the deductible) are applied to each individual loss.

- Determine the mean and variance of the aggregate payments on a per-loss basis.
- Determine the distribution of the number of payments.
- Determine the cumulative distribution function of the amount of a payment, given that a payment is made.
- Discretize the severity distribution using the method of rounding and a span of 40.
- Calculate the discretized distribution of aggregate payments up to a discretized amount paid of 120.

Negative binomial $c = 1/2$ and hence mixing cv $\sqrt{c}$, and the mean equals $r\beta/(1+\beta) = 1.4$. The cover is 125 xs 50. The severity is unconditional. First, the default calculation using `bs=1/64`.

```
kpw_9_63 = build('agg KPW.9.63 1.4 claims '
                 '125 xs 50 '
                 'sev 100 * gamma 2 ! '
                 'mixed gamma 2**-0.5')
qd(kpw_9_63)
mv(kpw_9_63)
qd(kpw_9_63.density_df.loc[:400:40*64, ['p', 'F', 'S']].join(
    kpw_9_63.sev_density_df[['p_sev', 'F_sev', 'S_sev']]),
    accuracy=5)
```

```
Aggregate object: KPW.9.63. Frequency distribution gamma. Severity
gamma, [0, 125], bounded; 125 xs 50. Updated with bucket size 1/32 and
log2 = 16. Validation: not unreasonable.

       Mean     SD      CV     Skew P01 Median    P99
X
Freq    1.4 1.5427  1.1019   1.5557
Sev  86.467 46.698 0.54006 -0.74977   0 117.84    125
Agg  121.05 144.39  1.1927   1.6385   0 100.84 614.38
```

```
mean     = 121.054
variance = 20847.33
std dev  = 144.386

               p       F        S      p_sev    F_sev   S_sev
loss
0.0      0.37325 0.37325  0.62675   0.090251 0.090251 0.90975
80.0  4.2253e-05 0.47185  0.52815 0.00011072  0.37323 0.62677
160.0  3.327e-05 0.72045  0.27955          0        1       0
240.0 3.0211e-05 0.80373  0.19627          0        1       0
320.0 1.7591e-05  0.9016 0.098397          0        1       0
400.0 9.0878e-06 0.94991 0.050095          0        1       0
```

Next, calculations performed with the requested broader `bs=40`.

```
kpw_9_63.update(log2=8, bs=40)
qd(kpw_9_63)
qd(kpw_9_63.density_df.loc[:400, ['p', 'F', 'S']].join(
    kpw_9_63.sev_density_df[['p_sev', 'F_sev', 'S_sev']]),
    accuracy=5)
```

```
Aggregate object: KPW.9.63. Frequency distribution gamma. Severity
gamma, [0, 125], bounded; 125 xs 50. Updated with bucket size 40 and
log2 = 8. Validation: fails sev mean, agg mean.

       Mean     SD     CV     Skew P01 Median P99
X
Freq    1.4 1.5427 1.1019   1.5557
Sev  84.042 45.887  0.546 -0.81723   0    120 120
Agg  117.66 140.56 1.1947    1.636   0    120 600

             p       F        S   p_sev   F_sev   S_sev
loss
0.0    0.39509 0.39509  0.60491  0.1558  0.1558  0.8442
40.0   0.05047 0.44556  0.55444 0.14517 0.30097 0.69903
80.0  0.053928 0.49949  0.50051  0.1412 0.44217 0.55783
120.0  0.20376 0.70325  0.29675 0.55783       1       0
160.0 0.042972 0.74622  0.25378       0       1       0
200.0 0.042194 0.78841  0.21159       0       1       0
240.0 0.081719 0.87013  0.12987       0       1       0
280.0 0.024403 0.89453  0.10547       0       1       0
320.0  0.02234 0.91687 0.083127       0       1       0
360.0 0.030181 0.94705 0.052946       0       1       0
400.0 0.011572 0.95863 0.041374       0       1       0
```

The apparent difference in the severity distribution is caused by the rounding method. In the first case F(40) is almost exact whereas in the second it is actually F(60).

7 Group Life Individual Risk Model, Example 9.15, 9.18

Consider a group life insurance contract with an accidental death benefit. Assume that for all members the probability of death in the next year is 0.01 and that 30% of deaths are accidental. For 50 employees, the benefit for an ordinary death is 50,000 and for an accidental death it is 100,000. For the remaining 25 employees, the benefits are 75,000 and 150,000, respectively. Develop an individual risk model and determine its mean and variance.

The `Portfolio` solution, working in thousands.

```
kpw_9_15p = build('port KPW.9.15.p '
                  'agg A 0.5 claims '
                      'dsev [50 100] [0.7 0.3] '
                      'binomial 0.01 '
                  'agg B 0.25 claims '
                      'dsev [75 150] [0.7 0.3] '
                      'binomial 0.01 ')
qd(kpw_9_15p)
mv(kpw_9_15p)
```

```
Portfolio object: KPW.9.15.p. Contains 2 aggregate components. Updated
with bucket size 50 and log2 = 6. Validation: fails sev cv, agg cv.

             Mean      SD      CV     Skew P01 Median P99
unit  X
A     Freq    0.5 0.70356  1.4071   1.3929
      Sev      65  22.913 0.35251  0.87287  50     50 100
      Agg    32.5  48.517  1.4928   1.6562   0      0 200
B     Freq   0.25 0.49749    1.99   1.9699
      Sev    97.5  40.234 0.41265 0.090686  50    100 150
      Agg  24.375  52.512  2.1543   2.3721   0      0 200
total Agg  56.875  71.494   1.257   1.4575   0     50 300
mean     = 56.875
variance = 5001.984
std dev  = 70.7247
```

The `density_df` dataframe contains the exact aggregate distribution, which is not easy to compute by other means. KPW says (emphasis added)

With regard to calculating the probabilities, there are at least three options. One is to do an **exact calculation**, which involves **numerous convolutions** and **almost always requires more excessive computing time**. Recursive formulas have been developed, but they are cumbersome and are not presented here. For one such method, see De Pril [27]. One alternative is a parametric approximation as discussed for the collective risk model. Another alternative is to replace the individual risk model with a similar collective risk model and then do the calculations with that model. These two approaches are presented here.

The following solution attempts to commute convolution through the mixture. This works for a compound Poisson. However, the sum of binomials is not binomial, and so the frequencies can't be independent binomial. They can be independent Poisson because it is additive.

```
kpw_9_15w = build('agg KPW.9.15.w '
                  '0.75 claims '
                  'dsev [50 75 100 150] '
                  '[0.35/0.75, 0.175/0.75, 0.15/0.75, 0.075/0.75] '
                  'binomial 0.01 ')
qd(kpw_9_15w)
mv(kpw_9_15w)
```

```
Aggregate object: KPW.9.15.w. Frequency distribution binomial. Severity
dhistogram, [50, 150], bounded; atoms [50 75 100 150]. Updated with
bucket size 25 and log2 = 6. Validation: not unreasonable.

       Mean      SD      CV   Skew P01 Median P99
X
Freq   0.75 0.86168  1.1489 1.1373
Sev  75.833  31.281 0.41249 1.1623  50     75 150
Agg  56.875  70.737  1.2437 1.4389   0     50 300
mean     = 56.875
```

```
variance = 5003.745
std dev  = 70.7372
```

The compound Poisson approximation matches the mean but its variance is slightly off.

```
kpw_9_15cp = build('agg KPW.9.15.cp '
                   '0.75 claims '
                   'dsev [50 75 100 150] '
                   '[0.35/0.75, 0.175/0.75, 0.15/0.75, 0.075/0.75] '
                   'poisson ')
qd(kpw_9_15cp)
mv(kpw_9_15cp)
```

```
Aggregate object: KPW.9.15.cp. Frequency distribution poisson. Severity
dhistogram, [50, 150], bounded; atoms [50 75 100 150]. Updated with
bucket size 25 and log2 = 6. Validation: not unreasonable.

       Mean      SD      CV   Skew P01 Median P99
X
Freq   0.75 0.86603  1.1547 1.1547
Sev  75.833  31.281 0.41249 1.1623  50     75 150
Agg  56.875  71.041  1.2491 1.4523   0     50 300
mean     = 56.875
variance = 5046.875
std dev  = 71.0414
```

Comparing probabilities shows that all three distributions are very close.

```
bit = pd.concat((kpw_9_15p.density_df.loc[:400:128, ['p_total']].query('p_total > 1e-10'),
                 kpw_9_15cp.density_df.loc[:400, ['p_total']].query('p_total > 0'),
                 kpw_9_15w.density_df.loc[:400, ['p_total']].query('p_total > 0'),
                ),
                keys=('exact', 'compound Po', 'wrong'), axis=1).rename(columns={'p_total': 'p'})
bit = bit.droplevel(1, axis=1)
bit.index.name = 'loss'
qd(bit, accuracy=5)
```

```
        exact  compound Po      wrong
loss
0.0   0.47059      0.47237    0.47059
50.0      NaN      0.16533    0.16637
75.0      NaN     0.082664   0.083185
100.0     NaN     0.099787    0.10032
125.0     NaN     0.028932   0.029017
150.0     NaN     0.070835   0.071104
175.0     NaN     0.017463   0.017428
200.0     NaN     0.024881   0.024823
225.0     NaN     0.011552   0.011478
250.0     NaN     0.011399     0.0113
275.0     NaN    0.0040587  0.0039712
300.0     NaN    0.0050707   0.004986
325.0     NaN    0.0018166  0.0017595
350.0     NaN     0.001694  0.0016392
375.0     NaN   0.00077117 0.00073805
400.0     NaN   0.00064447 0.00061581
```

8 Group Life Individual Risk Model, Example 9.16, 9.17

A small manufacturing business has a group life insurance contract on its 14 permanent employees. The actuary for the insurer has selected a mortality table to represent the mortality of the group. Each employee is insured for the amount of his or her salary rounded up to the next 1,000. The group's data are shown in the next table.

Employee	Age	Sex	Benefit	q
1	20	M	15,000	0.00149
2	23	M	16,000	0.00142
3	27	M	20,000	0.00128
4	30	M	28,000	0.00122
5	31	M	31,000	0.00123
6	46	M	18,000	0.00353
7	47	M	26,000	0.00394
8	49	M	24,000	0.00484
9	64	M	60,000	0.02182
10	17	F	14,000	0.00050
11	22	F	17,000	0.00050
12	26	F	19,000	0.00054
13	37	F	30,000	0.00103
14	55	F	55,000	0.00479

If the insurer adds a 45% relative loading to the net (pure) premium, what are the chances that it will lose money in a given year? Use the normal and lognormal approximations.

In order to make the answer self-contained, the code below includes the data munging to re-create the table, pasted from a pdf.

```
data = '''1
20
M
15,000
0.00149
2
23
M
16,000
0.00142
3
27
M
20,000
0.00128
4
30
M
28,000
0.00122
5
31
M
31,000
0.00123
6
46
M
18,000
```

```
0.00353
7
47
M
26,000
0.00394
8
49
M
24,000
0.00484
9
64
M
60,000
0.02182
10
17
F
14,000
0.00050
11
22
F
17,000
0.00050
12
26
F
19,000
0.00054
13
37
F
30,000
0.00103
14
55
F
55,000
0.00479'''
sdata = data.split('\n')
df = pd.DataFrame(zip(*[iter(sdata)]*5),
                  columns=['Employee', 'Age', 'Sex', 'Benefit', 'q'])
df.Benefit = df.Benefit.str.replace(',','').astype(float)
df.q = df.q.astype(float)
df = df.set_index('Employee')
qd(df)
print(f'expected claim count = {df.q.sum():.6g}')
```

```
          Age Sex  Benefit       q
Employee
1          20   M    15000 0.00149
2          23   M    16000 0.00142
3          27   M    20000 0.00128
4          30   M    28000 0.00122
5          31   M    31000 0.00123
6          46   M    18000 0.00353
7          47   M    26000 0.00394
8          49   M    24000 0.00484
9          64   M    60000 0.02182
10         17   F    14000  0.0005
```

```
11        22   F    17000  0.0005
12        26   F    19000 0.00054
13        37   F    30000 0.00103
14        55   F    55000 0.00479
expected claim count = 0.04813
```

Here are the FFT-exact, and various approximations to the required probability. Working in thousands. The `dsev` clauses enter the fixed benefit amount for each employee. Note the outsize impact of employee 9.

```
from aggregate import Portfolio
a = [build(f'agg ee.{i} {r.q} claims '
           f'dsev [{r.Benefit / 1000}] '
           f'bernoulli')
         for i, r in df.iterrows()]

kpw_9_16p = Portfolio('KPW.9.16p', a)
kpw_9_16p.update(log2=8, bs=1, remove_fuzz=True)
qd(kpw_9_16p)
mv(kpw_9_16p)
appx = kpw_9_16p.approximate('all')
premium = 1.45 * kpw_9_16p.actual_m
ans = {k: v.sf(premium) for k, v in appx.items()}
ans['FFT'] = kpw_9_16p.sf(premium)
qd(pd.DataFrame(ans.values(),
                index=pd.Index(ans.keys(), name='method'),
                columns=['premium']).sort_values('premium'),
  accuracy=5)
```

```
Portfolio object: KPW.9.16p. Contains 14 aggregate components. Updated
with bucket size 1 and log2 = 8. Validation: not unreasonable.

              Mean       SD     CV   Skew P01 Median P99
unit  X
ee.1  Freq 0.00149 0.038572 25.887 25.848
      Sev       15        0      0         15     15  15
      Agg  0.02235  0.57858 25.887 25.848   0      0   0
ee.2  Freq 0.00142 0.037656 26.518 26.481
      Sev       16        0      0         16     16  16
      Agg  0.02272   0.6025 26.518 26.481   0      0   0
ee.3  Freq 0.00128 0.035754 27.933 27.897
      Sev       20        0      0         20     20  20
      Agg   0.0256  0.71508 27.933 27.897   0      0   0
ee.4  Freq 0.00122 0.034907 28.612 28.577
      Sev       28        0      0         28     28  28
      Agg  0.03416   0.9774 28.612 28.577   0      0   0
...            ...      ...    ...    ...  ..    ...  ..
ee.11 Sev       17        0      0         17     17  17
      Agg   0.0085  0.38004  44.71 44.688   0      0   0
ee.12 Freq 0.00054 0.023232 43.022 42.998
      Sev       19        0      0         19     19  19
      Agg  0.01026   0.4414 43.022 42.998   0      0   0
ee.13 Freq 0.00103 0.032077 31.143 31.111
      Sev       30        0      0         30     30  30
      Agg   0.0309  0.96231 31.143 31.111   0      0   0
ee.14 Freq 0.00479 0.069044 14.414 14.345
      Sev       55        0      0         55     55  55
      Agg  0.26345   3.7974 14.414 14.345   0      0   0
total Agg   2.0544   10.126 4.9289 5.2673   0      0  60
mean     = 2.05441
variance = 102.5336
std dev  = 10.1259
```

```
          premium
method
FFT      0.047261
gamma    0.091498
lognorm   0.13449
sgamma    0.18346
slognorm  0.28099
norm      0.46363
```

Here is a sample from the distribution and the mean-matched compound Poisson (for Exercise 9.18). The latter `dsev` clause works because all the benefit amounts are different. The temporary variable `sev` creates the severity curve. The log pmf graph reflects the irregular benefit amounts. Compare the cdf under `comp Po` with Table 9.17.

```
sev = df[['Benefit', 'q']]
sev.q = sev.q / sev.q.sum()
sev = sev.sort_values('Benefit')
kpw_9_16cp = build('agg kpw_9_16.po '
                   f'{df.q.sum()} claims '
                   f'dsev {sev.Benefit.values /  1000} {sev.q.values} '
                   'poisson', bs=1, log2=10)
qd(kpw_9_16cp)
bit = pd.concat((kpw_9_16p.density_df.query('p_total > 0')[['p_total', 'F', 'S']],
                 kpw_9_16cp.density_df.query('p_total > 0')[['p_total', 'F', 'S']]),
                keys=['exact', 'comp Po'], axis=1)
bit.index = [f'{i:.0f}' for i in bit.index]
bit.index.name = 'loss'
with pd.option_context('display.max_rows', 360, 'display.max_columns', 10,
                       'display.width', 150, 'display.float_format', lambda x: f'{x:.7g}'):
    print(bit)
fig, axs = plt.subplots(1,2, figsize=(3.5*2, 2.45), constrained_layout=True, squeeze=True)
ax0, ax1 = axs.flat
bit[('exact', 'p_total')].plot(marker='.', lw=.25, logy=True, ax=ax0, label='Portfolio');
bit[('comp Po', 'p_total')].plot(marker='.', markerfacecolor='None', lw=.25, logy=True, ax=ax0,
label='compound Po');
(1-bit[('exact', 'p_total')].cumsum()).plot(ax=ax1);
(1-bit[('comp Po', 'p_total')].cumsum()).plot(ax=ax1);
ax0.legend();
ax0.set(ylabel='log pmf');
ax1.set(ylabel='survival function');
```

```
Aggregate object: kpw_9_16.po. Frequency distribution poisson. Severity
dhistogram, [14, 60], bounded; 14 atoms [14 15 16 ... 55 60] on [14,
60]. Updated with bucket size 1 and log2 = 10. Validation: not
unreasonable.

        Mean      SD      CV     Skew P01 Median P99
X
Freq 0.04813 0.21939  4.5582   4.5582
Sev   42.685  18.604 0.43586 -0.28928  14     55  60
Agg   2.0544  10.215  4.9723   5.4286   0      0  60
            exact                              comp Po
          p_total         F            S      p_total         F            S
loss
0        0.952739  0.952739   0.04726095    0.9530099 0.9530099   0.04699011
14   0.0004766078 0.9532157   0.04678434 0.0004765049 0.9534864   0.04651361
15      0.0014217 0.9546374   0.04536264  0.001419985 0.9549064   0.04509362
16    0.001354813 0.9559922   0.04400783  0.001353274 0.9562597   0.04374035
17   0.0004766078 0.9564688   0.04353122 0.0004765049 0.9567362   0.04326384
18    0.003375083 0.9598439   0.04015614  0.003364125 0.9601003   0.03989972
19   0.0005147571 0.9603586   0.03964138 0.0005146252 0.9606149   0.03938509
```

20	0.001221069	0.9615797	0.03842031	0.001219853	0.9618348	0.03816524
24	0.004633684	0.9662134	0.03378663	0.004612568	0.9664473	0.03355267
26	0.00376864	0.969982	0.03001799	0.003754859	0.9702022	0.02979781
28	0.001163761	0.9711458	0.02885423	0.001162791	0.971365	0.02863502
29	7.112054e-07	0.9711465	0.02885352	7.099922e-07	0.9713657	0.02863431
30	0.0009830108	0.9721295	0.02787051	0.0009833345	0.972349	0.02765098
31	0.001175572	0.9733051	0.02669493	0.001174457	0.9735235	0.02647652
32	2.399591e-06	0.9733075	0.02669253	3.352879e-06	0.9735268	0.02647317
33	5.971631e-06	0.9733134	0.02668656	5.946495e-06	0.9735328	0.02646722
34	6.178405e-06	0.9733196	0.02668038	6.272902e-06	0.9735391	0.02646095
35	4.242488e-06	0.9733239	0.02667614	4.23041e-06	0.9735433	0.02645672
36	1.993891e-06	0.9733259	0.02667415	7.927184e-06	0.9735512	0.02644879
37	2.434369e-06	0.9733283	0.02667171	2.426553e-06	0.9735536	0.02644637
38	6.643644e-06	0.9733349	0.02666507	6.751312e-06	0.9735604	0.02643961
39	7.574225e-06	0.9733425	0.02665749	7.531446e-06	0.9735679	0.02643208
40	8.474451e-06	0.973351	0.02664902	9.207982e-06	0.9735771	0.02642287
41	7.941654e-06	0.9733589	0.02664108	7.901023e-06	0.973585	0.02641497
42	2.23561e-05	0.9733813	0.02661872	2.219562e-05	0.9736072	0.02639278
43	6.125396e-06	0.9733874	0.0266126	6.100774e-06	0.9736133	0.02638668
44	2.143545e-05	0.9734088	0.02659116	2.130123e-05	0.9736346	0.02636538
45	4.672158e-06	0.9734135	0.02658649	4.659237e-06	0.9736393	0.02636072
46	1.210077e-05	0.9734256	0.02657439	1.205366e-05	0.9736513	0.02634866
47	2.791514e-06	0.9734284	0.0265716	2.78804e-06	0.9736541	0.02634587
48	5.56219e-06	0.973434	0.02656603	1.670997e-05	0.9736708	0.02632916
49	4.696495e-06	0.9734387	0.02656134	4.6784e-06	0.9736755	0.02632449
50	2.022647e-05	0.9734589	0.02654111	2.007543e-05	0.9736956	0.02630441
51	1.510358e-06	0.9734604	0.0265396	1.516371e-06	0.9736971	0.02630289
52	5.666162e-06	0.9734661	0.02653394	1.304127e-05	0.9737101	0.02628985
53	1.370555e-08	0.9734661	0.02653392	1.686769e-08	0.9737102	0.02628984
54	9.392317e-06	0.9734755	0.02652453	9.356468e-06	0.9737195	0.02628048
55	0.004591308	0.9780668	0.02193322	0.004570612	0.9782901	0.02170987
56	3.900383e-06	0.9780707	0.02192932	4.60947e-06	0.9782947	0.02170526
57	4.682325e-06	0.9780754	0.02192464	4.659748e-06	0.9782994	0.0217006
58	1.240279e-06	0.9780766	0.0219234	1.246296e-06	0.9783006	0.02169935
59	1.480938e-06	0.9780781	0.02192192	1.478042e-06	0.9783021	0.02169787
60	0.02125253	0.9993306	0.000669383	0.02079525	0.9990974	0.0009026226
61	1.248838e-06	0.9993319	0.0006681341	1.247493e-06	0.9990986	0.0009013751
62	4.172979e-08	0.9993319	0.0006680924	7.93117e-07	0.9990994	0.000900582
63	2.713599e-08	0.9993319	0.0006680653	4.479884e-08	0.9990995	0.0009005372
64	4.244129e-08	0.999332	0.0006680228	6.96877e-08	0.9990995	0.0009004675
65	4.491981e-08	0.999332	0.0006679779	5.030504e-08	0.9990996	0.0009004172
66	4.146533e-08	0.9993321	0.0006679364	9.361181e-08	0.9990997	0.0009003236
67	2.339189e-08	0.9993321	0.000667913	4.760197e-08	0.9990997	0.000900276
68	8.513895e-08	0.9993322	0.0006678279	1.101497e-07	0.9990998	0.0009001659
69	2.329551e-06	0.9993345	0.0006654983	2.321713e-06	0.9991022	0.0008978441
70	6.910878e-06	0.9993414	0.0006585875	6.89662e-06	0.9991091	0.0008909475
71	6.544798e-06	0.999348	0.0006520427	6.512085e-06	0.9991156	0.0008844354
72	2.350356e-06	0.9993503	0.0006496923	2.366974e-06	0.9991179	0.0008820685
73	1.628085e-05	0.9993666	0.0006334115	1.615066e-05	0.9991341	0.0008659178
74	1.314419e-05	0.9993797	0.0006202673	1.294404e-05	0.999147	0.0008529738
75	3.761971e-05	0.9994174	0.0005826475	3.685732e-05	0.9991839	0.0008161165
76	3.023577e-05	0.9994476	0.0005524118	2.959418e-05	0.9992135	0.0007865223
77	1.064505e-05	0.9994582	0.0005417667	1.041214e-05	0.9992239	0.0007761101
78	7.531246e-05	0.9995335	0.0004664543	7.34545e-05	0.9992973	0.0007026556
79	3.379094e-05	0.9995673	0.0004326633	3.334395e-05	0.9993307	0.0006693117
80	2.725768e-05	0.9995946	0.0004054057	2.665243e-05	0.9993573	0.0006426592
81	1.816289e-05	0.9996128	0.0003872428	1.801028e-05	0.9993754	0.000624649
82	5.911614e-09	0.9996128	0.0003872369	1.74019e-08	0.9993754	0.0006246316
83	5.608384e-06	0.9996184	0.0003816285	5.586048e-06	0.999381	0.0006190455
84	0.0001033707	0.9997217	0.0002782578	0.0001006574	0.9994816	0.0005183881
85	4.743e-06	0.9997265	0.0002735148	4.72191e-06	0.9994863	0.0005136662
86	8.972427e-05	0.9998162	0.0001837905	8.756326e-05	0.9995739	0.0004261029
87	1.649612e-08	0.9998162	0.000183774	2.194197e-08	0.9995739	0.000426081

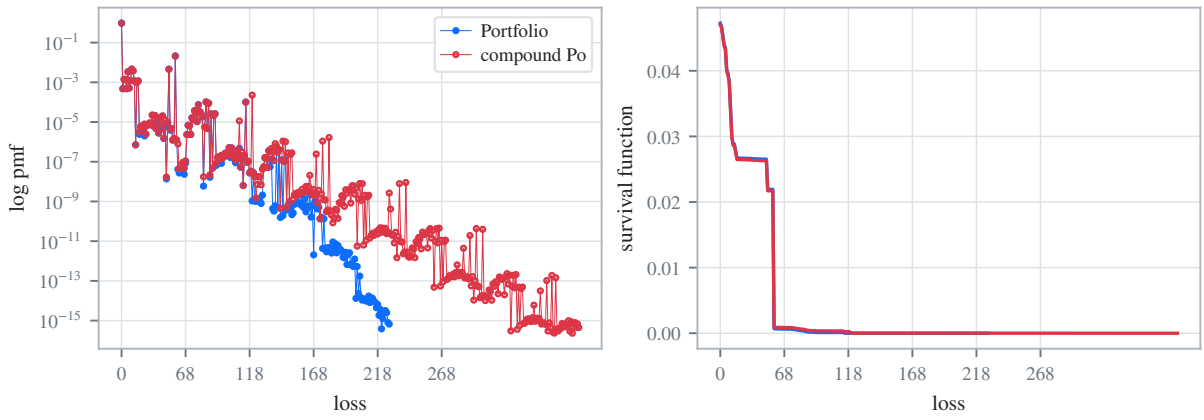
88	2.598867e-05	0.9998422	0.0001577853	2.540443e-05	0.9995993	0.0004006766
89	4.722525e-08	0.9998423	0.0001577381	4.726622e-08	0.9995994	0.0004006293
90	2.194831e-05	0.9998642	0.0001357898	2.147808e-05	0.9996208	0.0003791512
91	2.623287e-05	0.9998904	0.0001095569	2.56655e-05	0.9996465	0.0003534857
92	6.536066e-08	0.9998905	0.0001094916	8.594597e-08	0.9996466	0.0003533998
93	1.65287e-07	0.9998907	0.0001093263	1.62614e-07	0.9996468	0.0003532372
94	1.743665e-07	0.9998908	0.0001091519	1.733535e-07	0.9996469	0.0003530638
95	1.355125e-07	0.999891	0.0001090164	1.365987e-07	0.9996471	0.0003529272
96	8.286137e-08	0.9998911	0.0001089335	2.111788e-07	0.9996473	0.000352716
97	1.619798e-07	0.9998912	0.0001087716	1.594421e-07	0.9996474	0.0003525566
98	1.778121e-07	0.9998914	0.0001085938	1.768561e-07	0.9996476	0.0003523797
99	2.722478e-07	0.9998917	0.0001083215	2.665397e-07	0.9996479	0.0003521132
100	2.116102e-07	0.9998919	0.0001081099	2.234979e-07	0.9996481	0.0003518897
101	2.354722e-07	0.9998921	0.0001078744	2.302657e-07	0.9996483	0.0003516594
102	5.121685e-07	0.9998926	0.0001073623	4.978643e-07	0.9996488	0.0003511616
103	1.63469e-07	0.9998928	0.0001071988	2.132647e-07	0.9996491	0.0003509483
104	5.007765e-07	0.9998933	0.000106698	4.873585e-07	0.9996495	0.0003504609
105	2.016058e-07	0.9998935	0.0001064964	1.979245e-07	0.9996497	0.000350263
106	2.772016e-07	0.9998938	0.0001062192	2.703786e-07	0.99965	0.0003499926
107	8.955276e-08	0.9998939	0.0001061296	1.233815e-07	0.9996501	0.0003498692
108	1.241644e-07	0.999894	0.0001060055	3.6475e-07	0.9996505	0.0003495045
109	1.499989e-07	0.9998941	0.0001058555	1.469601e-07	0.9996506	0.0003493575
110	4.787335e-07	0.9998946	0.0001053768	1.139834e-05	0.999662	0.0003379592
111	5.24875e-08	0.9998947	0.0001053243	5.520826e-08	0.9996621	0.000337904
112	1.489302e-07	0.9998948	0.0001051753	3.069069e-07	0.9996624	0.0003375971
113	6.282775e-09	0.9998948	0.0001051691	6.359186e-09	0.9996624	0.0003375907
114	2.166401e-07	0.999895	0.0001049524	2.112529e-07	0.9996626	0.0003373795
115	0.0001024173	0.9999975	2.53513e-06	9.973353e-05	0.9997624	0.0002376459
116	9.301591e-08	0.9999976	2.442114e-06	1.065637e-07	0.9997625	0.0002375394
117	1.046485e-07	0.9999977	2.337466e-06	1.054805e-07	0.9997626	0.0002374339
118	2.779748e-08	0.9999977	2.309668e-06	2.74138e-08	0.9997626	0.0002374065
119	3.323942e-08	0.9999977	2.276429e-06	3.258744e-08	0.9997626	0.0002373739
120	1.07834e-09	0.9999977	2.27535e-06	0.0002268827	0.9999895	1.049118e-05
121	2.805718e-08	0.9999978	2.247293e-06	2.767002e-08	0.9999895	1.046351e-05
122	1.04354e-09	0.9999978	2.24625e-06	1.753529e-08	0.9999896	1.044597e-05
123	1.015218e-09	0.9999978	2.245234e-06	1.506102e-09	0.9999896	1.044447e-05
124	1.11823e-09	0.9999978	2.244116e-06	7.176094e-09	0.9999896	1.043729e-05
125	1.330164e-09	0.9999978	2.242786e-06	1.78429e-08	0.9999896	1.041945e-05
126	1.040647e-09	0.9999978	2.241745e-06	1.771138e-08	0.9999896	1.040174e-05
127	7.934845e-10	0.9999978	2.240952e-06	6.910456e-09	0.9999896	1.039483e-05
128	2.074415e-09	0.9999978	2.238878e-06	4.117227e-08	0.9999896	1.035365e-05
129	5.213364e-08	0.9999978	2.186744e-06	5.695507e-08	0.9999897	1.02967e-05
130	1.542995e-07	0.999998	2.032444e-06	1.646233e-07	0.9999899	1.013208e-05
131	1.46062e-07	0.9999981	1.886382e-06	1.424089e-07	0.99999	9.989666e-06
132	5.249366e-08	0.9999982	1.833889e-06	5.171857e-08	0.9999901	9.937948e-06
133	3.632949e-07	0.9999985	1.470594e-06	3.526438e-07	0.9999904	9.585304e-06
134	5.60785e-08	0.9999986	1.414515e-06	2.220186e-07	0.9999906	9.363285e-06
135	1.318434e-07	0.9999987	1.282672e-06	4.663595e-07	0.9999911	8.896926e-06
136	4.358886e-10	0.9999987	1.282236e-06	3.667832e-07	0.9999915	8.530142e-06
137	3.292504e-10	0.9999987	1.281907e-06	1.138415e-07	0.9999916	8.416301e-06
138	6.005527e-10	0.9999987	1.281306e-06	8.153441e-07	0.9999924	7.600957e-06
139	4.976509e-07	0.9999992	7.836552e-07	6.051008e-07	0.999993	6.995856e-06
140	4.940901e-10	0.9999992	7.831612e-07	3.024997e-07	0.9999933	6.693356e-06
141	4.051553e-07	0.9999996	3.780058e-07	4.064893e-07	0.9999937	6.286867e-06
142	1.556782e-10	0.9999996	3.778501e-07	4.463796e-10	0.9999937	6.286421e-06
143	1.251058e-07	0.9999997	2.527444e-07	1.219742e-07	0.9999938	6.164447e-06
144	1.937367e-10	0.9999997	2.525506e-07	1.098375e-06	0.9999949	5.066072e-06
145	1.058015e-07	0.9999999	1.467492e-07	1.030876e-07	0.999995	4.962984e-06
146	1.262168e-07	1	2.053237e-08	1.016858e-06	0.9999961	3.946126e-06
147	3.685373e-10	1	2.016383e-08	5.121882e-10	0.9999961	3.945614e-06
148	6.46479e-10	1	1.951735e-08	2.77595e-07	0.9999963	3.668019e-06
149	6.999917e-10	1	1.881736e-08	9.506666e-10	0.9999963	3.667068e-06
150	4.588675e-10	1	1.835849e-08	2.346692e-07	0.9999966	3.432399e-06

```
151  2.170583e-10         1 1.814144e-08 2.805268e-07 0.9999968 3.151872e-06
152  2.643346e-10         1  1.78771e-08  1.33265e-09 0.9999968  3.15054e-06
153   7.16224e-10         1 1.716088e-08 2.204159e-09 0.9999969 3.148335e-06
154  8.158174e-10         1 1.634506e-08 2.534458e-09 0.9999969 3.145801e-06
155  9.122321e-10         1 1.543283e-08 2.028215e-09 0.9999969 3.143773e-06
156  8.566022e-10         1 1.457623e-08 2.859701e-09 0.9999969 3.140913e-06
157   2.40212e-09         1 1.217411e-08  2.93432e-09 0.9999969 3.137979e-06
158  6.608837e-10         1 1.151322e-08  2.44401e-09 0.9999969 3.135535e-06
159  2.304181e-09         1 9.209041e-09 4.077408e-09 0.9999969 3.131457e-06
160  5.036921e-10         1 8.705349e-09 2.915487e-09 0.9999969 3.128542e-06
161  1.300943e-09         1 7.404406e-09 3.161405e-09 0.9999969  3.12538e-06
162  3.006818e-10         1 7.103724e-09 5.729582e-09 0.9999969 3.119651e-06
163  5.986384e-10         1 6.505086e-09 3.201579e-09 0.9999969 3.116449e-06
164  5.047601e-10         1 6.000326e-09 5.670897e-09 0.9999969 3.110778e-06
165  2.172347e-09         1 3.827979e-09 2.073139e-08 0.9999969 3.090047e-06
166  1.623502e-10         1 3.665628e-09 3.083291e-09 0.9999969 3.086964e-06
167  6.086006e-10         1 3.057028e-09 2.082057e-09 0.9999969 3.084882e-06
168   2.04573e-12         1 3.054982e-09 3.995334e-09 0.9999969 3.080886e-06
169  1.009057e-09         1 2.045925e-09 2.109975e-09 0.9999969 3.078776e-06
170  6.145147e-10         1 1.431411e-09 2.439393e-07 0.9999972 2.834837e-06
171  4.192868e-10         1 1.012124e-09 8.580199e-10 0.9999972 2.833979e-06
172  5.027193e-10         1 5.094045e-10 3.601279e-09 0.9999972 2.830378e-06
173  1.333299e-10         1 3.760746e-10 1.352801e-10 0.9999972 2.830242e-06
174  1.590085e-10         1 2.170661e-10 2.382985e-09 0.9999972 2.827859e-06
175  4.288268e-12         1 2.127779e-10 1.088124e-06 0.9999983 1.739736e-06
176  1.340885e-10         1 7.868939e-11 1.228987e-09 0.9999983 1.738507e-06
177  4.486189e-12         1  7.42032e-11 1.192855e-09 0.9999983 1.737314e-06
178  2.920835e-12         1 7.128231e-11  3.02749e-10 0.9999983 1.737011e-06
179  4.562823e-12         1 6.671952e-11 3.683841e-10 0.9999983 1.736643e-06
180   4.82662e-12         1 6.189294e-11  1.65027e-06 0.9999999 8.637285e-08
181  4.454918e-12         1 5.743805e-11 3.319206e-10 0.9999999 8.604093e-08
182  2.514366e-12         1 5.492373e-11 2.035136e-10 0.9999999 8.583741e-08
183  9.143925e-12         1 4.577982e-11 8.424069e-11 0.9999999 8.575317e-08
184  3.826298e-12         1 4.195355e-11 1.503577e-10 0.9999999 8.560282e-08
185  7.320083e-12         1 3.463352e-11 4.000487e-10 0.9999999 8.520277e-08
186  2.580997e-12         1 3.205247e-11  3.64933e-10 0.9999999 8.483783e-08
187  6.060502e-12         1 2.599199e-11 1.396254e-10 0.9999999 8.469821e-08
188  3.909233e-12         1 2.208278e-11 8.727237e-10 0.9999999 8.382548e-08
189  3.768936e-12         1 1.831379e-11 7.747876e-10 0.9999999  8.30507e-08
190  3.132506e-12         1  1.51813e-11 1.950703e-09 0.9999999 8.109999e-08
191  1.538332e-12         1 1.364298e-11  1.62618e-09 0.9999999 7.947381e-08
192  1.447887e-12         1 1.219513e-11 5.652266e-10 0.9999999 7.890859e-08
193  2.725839e-12         1 9.469314e-12 3.871408e-09 0.9999999 7.503718e-08
194  6.673823e-13         1 8.801959e-12  2.58819e-09 0.9999999 7.244899e-08
195  2.106876e-12         1 6.695089e-12 3.878648e-09 0.9999999 6.857034e-08
196  2.600252e-12         1 4.094836e-12 3.322862e-09 0.9999999 6.524748e-08
197  6.347046e-13         1 3.460121e-12 8.305254e-10 0.9999999 6.441695e-08
198  7.662416e-13         1 2.693845e-12 6.129516e-09 0.9999999 5.828744e-08
199  5.279871e-13         1 2.165823e-12 6.156754e-09 0.9999999 5.213068e-08
200  1.257773e-12         1 9.080514e-13  2.36799e-09         1 4.976269e-08
201  1.318117e-14         1 8.948398e-13 4.582293e-09         1  4.51804e-08
202  5.310998e-13         1 3.637091e-13 5.655193e-12         1 4.517475e-08
203  2.332878e-14         1 3.403944e-13 1.331814e-09         1 4.384293e-08
204  1.743247e-13         1 1.660894e-13 7.991045e-09         1 3.585189e-08
205  1.421974e-14         1 1.518785e-13 1.125462e-09         1 3.472643e-08
206  1.067367e-14         1 1.412204e-13  7.84441e-09         1 2.688202e-08
207   1.25542e-14         1 1.286748e-13 6.361132e-12         1 2.687566e-08
208  1.108534e-14         1 1.175726e-13 2.022885e-09         1 2.485277e-08
209  9.803895e-15         1 1.078027e-13   1.1113e-11         1 2.484166e-08
210  9.503246e-15         1 9.825474e-14 1.710189e-09         1 2.313147e-08
211  1.723955e-14         1 8.104628e-14 2.045013e-09         1 2.108646e-08
212    8.1441e-15         1 7.294165e-14 1.447721e-11         1 2.107198e-08
213  1.419198e-14         1  5.87308e-14  1.99869e-11         1 2.105199e-08
```

214	1.3069e-14	1	4.563017e-14	2.498278e-11	1	2.102701e-08
215	9.181941e-15	1	3.641532e-14	1.943114e-11	1	2.100758e-08
216	8.387458e-15	1	2.797762e-14	2.588086e-11	1	2.09817e-08
217	4.424357e-15	1	2.353673e-14	3.051194e-11	1	2.095118e-08
218	6.462206e-15	1	1.709743e-14	2.291723e-11	1	2.092827e-08
219	1.852358e-15	1	1.521006e-14	3.87301e-11	1	2.088954e-08
220	3.657275e-15	1	1.154632e-14	4.736282e-11	1	2.084217e-08
221	3.860015e-16	1	1.121325e-14	2.793092e-11	1	2.081424e-08
222	1.257828e-15	1	9.992007e-15	4.501632e-11	1	2.076923e-08
223	2.590693e-15	1	7.438494e-15	2.96758e-11	1	2.073955e-08
224	3.203264e-15	1	4.218847e-15	4.462978e-11	1	2.069492e-08
226	2.513175e-15	1	1.665335e-15	2.380553e-11	1	2.02578e-08
228	7.999946e-16	1	8.881784e-16	2.928074e-11	1	2.020762e-08
230	6.668424e-16	1	2.220446e-16	2.644072e-09	1	1.754439e-08
225	NaN	NaN	NaN	4.133196e-10	1	2.02816e-08
227	NaN	NaN	NaN	2.090073e-11	1	2.02369e-08
229	NaN	NaN	NaN	1.915667e-11	1	2.018846e-08
231	NaN	NaN	NaN	8.206596e-12	1	1.753618e-08
232	NaN	NaN	NaN	2.810018e-11	1	1.750808e-08
233	NaN	NaN	NaN	1.469198e-12	1	1.750661e-08
234	NaN	NaN	NaN	1.791823e-11	1	1.748869e-08
235	NaN	NaN	NaN	7.914547e-09	1	9.574146e-09
236	NaN	NaN	NaN	9.459713e-12	1	9.564687e-09
237	NaN	NaN	NaN	8.997999e-12	1	9.555689e-09
238	NaN	NaN	NaN	2.312339e-12	1	9.553376e-09
239	NaN	NaN	NaN	2.852463e-12	1	9.550524e-09
240	NaN	NaN	NaN	9.002972e-09	1	5.475518e-10
241	NaN	NaN	NaN	2.816682e-12	1	5.447351e-10
242	NaN	NaN	NaN	1.639809e-12	1	5.430953e-10
243	NaN	NaN	NaN	1.558068e-12	1	5.415373e-10
244	NaN	NaN	NaN	1.794712e-12	1	5.397426e-10
245	NaN	NaN	NaN	4.569199e-12	1	5.351734e-10
246	NaN	NaN	NaN	3.991432e-12	1	5.311819e-10
247	NaN	NaN	NaN	1.48429e-12	1	5.296976e-10
248	NaN	NaN	NaN	9.45804e-12	1	5.202396e-10
249	NaN	NaN	NaN	7.367475e-12	1	5.128722e-10
250	NaN	NaN	NaN	1.533776e-11	1	4.975345e-10
251	NaN	NaN	NaN	1.288323e-11	1	4.846512e-10
252	NaN	NaN	NaN	4.119825e-12	1	4.805314e-10
253	NaN	NaN	NaN	2.848954e-11	1	4.520418e-10
254	NaN	NaN	NaN	2.153303e-11	1	4.305089e-10
255	NaN	NaN	NaN	2.401756e-11	1	4.064913e-10
256	NaN	NaN	NaN	2.364938e-11	1	3.828419e-10
257	NaN	NaN	NaN	4.549046e-12	1	3.782928e-10
258	NaN	NaN	NaN	3.505759e-11	1	3.432352e-10
259	NaN	NaN	NaN	4.316564e-11	1	3.000695e-10
260	NaN	NaN	NaN	1.428383e-11	1	2.857857e-10
261	NaN	NaN	NaN	3.440264e-11	1	2.513831e-10
262	NaN	NaN	NaN	4.775371e-14	1	2.513354e-10
263	NaN	NaN	NaN	9.695624e-12	1	2.416398e-10
264	NaN	NaN	NaN	4.360835e-11	1	1.980315e-10
265	NaN	NaN	NaN	8.192854e-12	1	1.898386e-10
266	NaN	NaN	NaN	4.524329e-11	1	1.445953e-10
267	NaN	NaN	NaN	5.492676e-14	1	1.445404e-10
268	NaN	NaN	NaN	1.10606e-11	1	1.334798e-10
269	NaN	NaN	NaN	9.141531e-14	1	1.333884e-10
270	NaN	NaN	NaN	9.353045e-12	1	1.240353e-10
271	NaN	NaN	NaN	1.118703e-11	1	1.128483e-10
272	NaN	NaN	NaN	1.160788e-13	1	1.127322e-10
273	NaN	NaN	NaN	1.367947e-13	1	1.125954e-10
274	NaN	NaN	NaN	1.828941e-13	1	1.124125e-10
275	NaN	NaN	NaN	1.578659e-13	1	1.122547e-10
276	NaN	NaN	NaN	1.748276e-13	1	1.120798e-10

277	NaN	NaN	NaN	2.204866e-13	1	1.118593e-10
278	NaN	NaN	NaN	1.607222e-13	1	1.116985e-10
279	NaN	NaN	NaN	2.648363e-13	1	1.114338e-10
280	NaN	NaN	NaN	6.394332e-13	1	1.107943e-10
281	NaN	NaN	NaN	1.809613e-13	1	1.106133e-10
282	NaN	NaN	NaN	2.707042e-13	1	1.103426e-10
283	NaN	NaN	NaN	1.970099e-13	1	1.101456e-10
284	NaN	NaN	NaN	2.664808e-13	1	1.098791e-10
285	NaN	NaN	NaN	4.382093e-12	1	1.054971e-10
286	NaN	NaN	NaN	1.39744e-13	1	1.053573e-10
287	NaN	NaN	NaN	1.477012e-13	1	1.052096e-10
288	NaN	NaN	NaN	1.615113e-13	1	1.050481e-10
289	NaN	NaN	NaN	1.259819e-13	1	1.049221e-10
290	NaN	NaN	NaN	1.916858e-11	1	8.575352e-11
291	NaN	NaN	NaN	5.610937e-14	1	8.569745e-11
292	NaN	NaN	NaN	1.640703e-13	1	8.553336e-11
293	NaN	NaN	NaN	1.077525e-14	1	8.552259e-11
294	NaN	NaN	NaN	1.011083e-13	1	8.542145e-11
295	NaN	NaN	NaN	4.317568e-11	1	4.224576e-11
296	NaN	NaN	NaN	5.481943e-14	1	4.219092e-11
297	NaN	NaN	NaN	5.09905e-14	1	4.213996e-11
298	NaN	NaN	NaN	1.407967e-14	1	4.212586e-11
299	NaN	NaN	NaN	1.710588e-14	1	4.210876e-11
300	NaN	NaN	NaN	3.929457e-11	1	2.814193e-12
301	NaN	NaN	NaN	1.864976e-14	1	2.795542e-12
302	NaN	NaN	NaN	1.020366e-14	1	2.785328e-12
303	NaN	NaN	NaN	1.615138e-14	1	2.769229e-12
304	NaN	NaN	NaN	1.513258e-14	1	2.75413e-12
305	NaN	NaN	NaN	3.489847e-14	1	2.719269e-12
306	NaN	NaN	NaN	3.002578e-14	1	2.689293e-12
307	NaN	NaN	NaN	1.066542e-14	1	2.678635e-12
308	NaN	NaN	NaN	6.886326e-14	1	2.609801e-12
309	NaN	NaN	NaN	5.3001e-14	1	2.556844e-12
310	NaN	NaN	NaN	9.019267e-14	1	2.466694e-12
311	NaN	NaN	NaN	7.905873e-14	1	2.387646e-12
312	NaN	NaN	NaN	2.253257e-14	1	2.365108e-12
313	NaN	NaN	NaN	1.580755e-13	1	2.207012e-12
314	NaN	NaN	NaN	1.355169e-13	1	2.071454e-12
315	NaN	NaN	NaN	1.186181e-13	1	1.952882e-12
316	NaN	NaN	NaN	1.363327e-13	1	1.816547e-12
317	NaN	NaN	NaN	1.995752e-14	1	1.796563e-12
318	NaN	NaN	NaN	1.623999e-13	1	1.634137e-12
319	NaN	NaN	NaN	2.302032e-13	1	1.403988e-12
320	NaN	NaN	NaN	7.027308e-14	1	1.333711e-12
321	NaN	NaN	NaN	1.935343e-13	1	1.140199e-12
322	NaN	NaN	NaN	3.045386e-16	1	1.139866e-12
323	NaN	NaN	NaN	5.294901e-14	1	1.086908e-12
324	NaN	NaN	NaN	1.904142e-13	1	8.965051e-13
325	NaN	NaN	NaN	4.473726e-14	1	8.517631e-13
326	NaN	NaN	NaN	2.081715e-13	1	6.435963e-13
327	NaN	NaN	NaN	3.647886e-16	1	6.432632e-13
328	NaN	NaN	NaN	4.840723e-14	1	5.948575e-13
329	NaN	NaN	NaN	5.77513e-16	1	5.943024e-13
330	NaN	NaN	NaN	4.096973e-14	1	5.533352e-13
331	NaN	NaN	NaN	4.899164e-14	1	5.043743e-13
332	NaN	NaN	NaN	7.182596e-16	1	5.037082e-13
333	NaN	NaN	NaN	7.538122e-16	1	5.02931e-13
334	NaN	NaN	NaN	1.05143e-15	1	5.019318e-13
335	NaN	NaN	NaN	1.217349e-15	1	5.007106e-13
336	NaN	NaN	NaN	9.343233e-16	1	4.998224e-13
337	NaN	NaN	NaN	1.22625e-15	1	4.986012e-13
338	NaN	NaN	NaN	8.943982e-16	1	4.97713e-13
339	NaN	NaN	NaN	1.411856e-15	1	4.962697e-13

```
340          NaN       NaN          NaN 5.993825e-15         1 4.902745e-13
341          NaN       NaN          NaN 9.226669e-16         1 4.893863e-13
342          NaN       NaN          NaN 1.322259e-15         1  4.88054e-13
343          NaN       NaN          NaN 1.015269e-15         1 4.870548e-13
344          NaN       NaN          NaN   1.2857e-15         1 4.857226e-13
345          NaN       NaN          NaN  3.14429e-14         1 4.543033e-13
346          NaN       NaN          NaN 6.647672e-16         1 4.536371e-13
347          NaN       NaN          NaN  8.05246e-16         1   4.5286e-13
348          NaN       NaN          NaN 7.189838e-16         1 4.521938e-13
349          NaN       NaN          NaN 6.461936e-16         1 4.515277e-13
350          NaN       NaN          NaN 1.043606e-13         1 3.471667e-13
351          NaN       NaN          NaN 2.984117e-16         1 3.468337e-13
352          NaN       NaN          NaN 7.639679e-16         1 3.460565e-13
354          NaN       NaN          NaN 4.532481e-16         1 3.456124e-13
355          NaN       NaN          NaN 1.884291e-13         1 1.572076e-13
356          NaN       NaN          NaN 2.545627e-16         1 1.569855e-13
357          NaN       NaN          NaN  2.32703e-16         1 1.567635e-13
360          NaN       NaN          NaN 1.429317e-13         1 1.387779e-14
368          NaN       NaN          NaN 3.755857e-16         1 1.354472e-14
369          NaN       NaN          NaN  3.00849e-16         1 1.321165e-14
370          NaN       NaN          NaN 4.267127e-16         1 1.276756e-14
371          NaN       NaN          NaN 3.991153e-16         1 1.232348e-14
373          NaN       NaN          NaN 7.037503e-16         1 1.165734e-14
374          NaN       NaN          NaN 6.777835e-16         1 1.099121e-14
375          NaN       NaN          NaN 4.888442e-16         1 1.054712e-14
376          NaN       NaN          NaN 6.513329e-16         1 9.880985e-15
378          NaN       NaN          NaN 6.364895e-16         1 9.214851e-15
379          NaN       NaN          NaN 9.880324e-16         1  8.21565e-15
380          NaN       NaN          NaN 2.915068e-16         1 7.882583e-15
381          NaN       NaN          NaN 8.687768e-16         1 6.994405e-15
383          NaN       NaN          NaN 2.313705e-16         1  6.77236e-15
384          NaN       NaN          NaN 7.051074e-16         1 6.106227e-15
386          NaN       NaN          NaN 7.944802e-16         1 5.329071e-15
410          NaN       NaN          NaN 4.555483e-16         1 4.884981e-15
415          NaN       NaN          NaN 6.833523e-16         1 4.218847e-15
420          NaN       NaN          NaN 4.459691e-16         1 3.774758e-15
```



Exercise 9.73.

An insurance company sold one-year term life insurance on a group of 2,300 independent lives as given in the next table.

Class	Benefit	q	Number
1	100,000	0.1	500
2	200,000	0.02	500
3	300,000	0.02	500
4	200,000	0.1	300
5	200,000	0.1	500

The insurance company reinsures amounts in excess of 100,000 on each life. The reinsurer wishes to charge a premium that is sufficient to guarantee that it will lose money 5% of the time on such groups. Obtain the appropriate premium by each of the following ways:

1. Using a normal approximation to the aggregate claims distribution.
2. Using a lognormal approximation.
3. Using a gamma approximation.
4. Using the compound Poisson approximation that matches the means.

In order to make the answer self-contained, the code below includes the data munging to re-create the table, pasted from a pdf.

```
data = '''1
100,000
0.10
500
2
200,000
0.02
500
3
300,000
0.02
500
4
200,000
0.10
300
5
200,000
0.10
500'''
sdata = data.split('\n')
df = pd.DataFrame(zip(*[iter(sdata)]*4),
                  columns=['Class', 'Benefit', 'q', 'Number'])
df.Benefit = df.Benefit.str.replace(',','').astype(float)
df.q = df.q.astype(float)
df.Number = df.Number.astype(int)
df = df.set_index('Class')
qd(df)
```

```
       Benefit    q  Number
Class
1        1e+05  0.1     500
2        2e+05 0.02     500
3        3e+05 0.02     500
4        2e+05  0.1     300
5        2e+05  0.1     500
```

Next, build the exact solution for the gross book as a `Portfolio` (extra credit).

```
a = [build(f'agg Class.{i} {r.q * r.Number} claims '
           f'dsev [{r.Benefit / 100000}] '
           f'binomial {r.q}')
         for i, r in df.iterrows()]

p = Portfolio('KPW.9.73p', a)
p.update(log2=10, bs=1, remove_fuzz=True)
qd(p)
```

```
Portfolio object: KPW.9.73p. Contains 5 aggregate components. Updated
with bucket size 1 and log2 = 10. Validation: not unreasonable.

              Mean     SD       CV     Skew P01 Median P99
unit    X
Class.1 Freq    50 6.7082  0.13416  0.11926
        Sev      1      0        0           1      1   1
        Agg     50 6.7082  0.13416  0.11926  35     50  66
Class.2 Freq    10 3.1305  0.31305  0.30666
        Sev      2      0        0           2      2   2
        Agg     20  6.261  0.31305  0.30666   8     20  36
Class.3 Freq    10 3.1305  0.31305  0.30666
        Sev      3      0        0           3      3   3
        Agg     30 9.3915  0.31305  0.30666  12     30  54
Class.4 Freq    30 5.1962  0.17321  0.15396
        Sev      2      0        0           2      2   2
        Agg     60 10.392  0.17321  0.15396  38     60  86
Class.5 Freq    50 6.7082  0.13416  0.11926
        Sev      2      0        0           2      2   2
        Agg    100 13.416  0.13416  0.11926  70    100 132
total   Agg    260 21.457 0.082527 0.083622 211    260 311
```

Build the reinsurer's loss distribution exactly, as `p_ceded`, a `Portfolio`, and the compound Poisson approximation `cp_ceded`, an `Aggregate`. The temporary variable `bit` is used to calculate the mixed severity distribution.

```
a_ceded = [build(f'agg Class.{i}.c {r.q * r.Number} claims '
           f'dsev [{r.Benefit / 100000 - 1}] '
           f'binomial {r.q}')
         for i, r in df.query('Benefit > 100000').iterrows()]

p_ceded = Portfolio('KPW.9.73pc', a_ceded)
p_ceded.update(log2=10, bs=1, remove_fuzz=True)
qd(p_ceded)

bit = df.query('Benefit > 100000')
bit['Claims'] = bit.q * bit.Number
bit.groupby('Benefit').Claims.sum()
cp_ceded = build('agg CP.Approx '
                 f'{bit.Claims.sum()} claims '
                 f'dsev [1 2] [0.9 0.1] '
                 'poisson')
qd(cp_ceded)
```

```
p_total has 32 negative values; NOT setting to zero...

Portfolio object: KPW.9.73pc. Contains 4 aggregate components. Updated
with bucket size 1 and log2 = 10. Validation: not unreasonable.

                Mean     SD      CV    Skew P01 Median P99
unit      X
Class.2.c Freq    10 3.1305 0.31305 0.30666
```

```
          Sev      1      0       0           1      1   1
          Agg     10 3.1305 0.31305 0.30666   4     10  18
Class.3.c Freq    10 3.1305 0.31305 0.30666
          Sev      2      0       0           2      2   2
          Agg     20  6.261 0.31305 0.30666   8     20  36
Class.4.c Freq    30 5.1962 0.17321 0.15396
          Sev      1      0       0           1      1   1
          Agg     30 5.1962 0.17321 0.15396  19     30  43
Class.5.c Freq    50 6.7082 0.13416 0.11926
          Sev      1      0       0           1      1   1
          Agg     50 6.7082 0.13416 0.11926  35     50  66
total     Agg    110     11     0.1 0.10689  85    110 136
Aggregate object: CP.Approx. Frequency distribution poisson. Severity
dhistogram, [1, 2], bounded; atoms [1 2]. Updated with bucket size 1 and
log2 = 8. Validation: not unreasonable.

      Mean     SD      CV    Skew P01 Median P99
X
Freq   100     10     0.1     0.1
Sev    1.1    0.3 0.27273  2.6667   1      1   2
Agg    110 11.402 0.10365 0.11469  84    110 137
```

Compute the various estimated premiums, the 95%iles of the aggregate loss distribution.

```
prem_confidence = 0.95
appx = p_ceded.approximate('all')
ans = {k: v.ppf(prem_confidence) for k, v in appx.items()}
ans['FFT'] = p_ceded.q(prem_confidence)
ans['Comp Po'] = cp_ceded.q(prem_confidence)
qd(pd.DataFrame(ans.values(),
                index=pd.Index(ans.keys(), name='method'),
                columns=['premium']).sort_values('premium'),
    accuracy=5)
```

```
          premium
method
FFT           128
norm       128.09
slognorm   128.42
sgamma     128.42
gamma       128.7
lognorm    128.97
Comp Po       129
```

Exercise 9.74. A group insurance contract covers 1,000 employees. An employee can have at most one claim per year. For 500 employees, there is a 0.02 probability of a claim, and when there is a claim, the amount has an exponential distribution with mean 500. For 250 other employees, there is a 0.03 probability of a claim and amounts are exponential with mean 750. For the remaining 250 employees, the probability is 0.04 and the mean is 1,000. Determine the exact mean and variance of total claims payments. Next, construct a compound Poisson model with the same mean and determine the variance of this model.

```
kpw_9_74p = build('port KPW.9.74p '
                  'agg A 10. claims sev  500 * expon binomial 0.02 '
                  'agg B 7.5 claims sev  750 * expon binomial 0.03 '
                  'agg C 10. claims sev 1000 * expon binomial 0.04 ')
qd(kpw_9_74p)
mv(kpw_9_74p)
```

```
Portfolio object: KPW.9.74p. Contains 3 aggregate components. Updated
with bucket size 2 and log2 = 16. Validation: not unreasonable.

            Mean     SD      CV    Skew  P01 Median   P99
unit  X
A     Freq    10 3.1305 0.31305 0.30666
      Sev    500    500       1       2    6    346  2302
      Agg   5000 2224.9 0.44497 0.66748  994   4750 11210
B     Freq   7.5 2.6972 0.35963 0.34851
      Sev    750    750       1       2    8    520  3454
      Agg   5625 2882.9 0.51251 0.76882  686   5252 13864
C     Freq    10 3.0984 0.30984 0.29693
      Sev   1000   1000       1       2   10    694  4606
      Agg  10000 4427.2 0.44272 0.66417 2014   9506 22348
total Agg  20625 5732.4 0.27794 0.44276 9214  20202 35772
mean     = 20625
variance = 3.286094e+07
std dev  = 5732.45
```

Compound Poisson approximation is easy to construct as a mixture.

```
kpw_9_74cp = build('agg KPW.9.74.cp [10 7.5 10] claims sev [500 750 1000] * expon poisson')
qd(kpw_9_74cp)
mv(kpw_9_74cp)
```

```
Aggregate object: KPW.9.74.cp. Frequency distribution poisson. Mixed
severity with 3 components. Updated with bucket size 2 and log2 = 16.
Validation: not unreasonable.

      Mean     SD      CV    Skew  P01 Median   P99
X
Freq  27.5  5.244 0.19069 0.19069
Sev    750 808.34  1.0778   2.372    6    492  3804
Agg  20625 5782.5 0.28036 0.44729 9134  20194 35922
mean     = 20625
variance = 3.34375e+07
std dev  = 5782.52
```

Klugman, S. A., Panjer, H. H., & Willmot, G. E. (2019). *Loss Models: from data to decisions* (Fifth.). John Wiley & Sons.