Recitations

Textbook and examination problems, worked with aggregate

Stephen J. Mildenhall

2026-10-22 19:54 commit 144a5d0+

Table of contents

Preface 4
Companion volumes . . . 4
Running the programs . . . 5
About this edition . . . 5

1. Klugman, Panjer and Willmot on aggregate loss models 6
1.1. Method of Moments Approximations, Examples 9.3 and 9.4 . . . 6
1.2. Group Dental Insurance, Examples 9.5, 9.6 . . . 7
1.3. Compound Poisson, Example 9.9, 9.10 . . . 14
1.4. Zero-Modified Binomial, Example 9.11 . . . 16
1.5. Extended Truncated Negative Binomial, Example 9.12 . . . 17
1.6. Poisson Pareto, Example 9.14 . . . 21
1.7. Group Life Individual Risk Model, Example 9.15, 9.18 . . . 23
1.8. Group Life Individual Risk Model, Example 9.16, 9.17 . . . 26

2. Frees on severity, coverage modifications and aggregate losses 40
2.1. Distribution Examples . . . 40
2.2. Mixture Example (3.3.5) . . . 42
2.3. Coverage Modifications . . . 43
2.4. Aggregate Loss Distributions . . . 47
2.5. Portfolio Management . . . 55

3. Bahnemann on aggregate distributions and layer pricing 59
3.1. Simple Discrete Aggregate, Example 4.1 . . . 59
3.2. Poisson-Gamma (Tweedie) Aggregate, Example 4.2 . . . 60
3.3. Approximations to the Tweedie, Example 4.3 . . . 61
3.4. Poisson-Discrete Distribution, Example 4.4 . . . 62
3.5. Poisson-Gamma Distribution, Example 4.5 . . . 63
3.6. Poisson-Lognormal Distribution With Limit, Example 4.15 . . . 64
3.7. Poisson-Gamma Distribution and Approximations, Problems 4.7 and 13 . . . 65
3.8. Poisson-Lognormal Layer Statistics, Example 5.13 . . . 66
3.9. Lognormal Increased Limits Factors (ILFs), Example 6.3 . . . 67
3.10. Layer Premium, Example 6.4 . . . 69
3.11. Risk Loads, Example 6.5 . . . 69
3.12. Aggregate Premiums, Example 6.6 . . . 71
3.13. Deductible Credits, Example 6.7 . . . 72
3.14. Summary . . . 74

4. Brehm et al. on reinsurance in enterprise risk analysis 76
4.1. Reinsurance Example . . . 76
4.2. Stochastic Models and Baseline Analysis . . . 77

5. Casualty Actuarial Society examination questions 83
5.1. Exam 9, Spring 2018, Question 15 . . . 83

Appendices 85

A. Every program in the book 85
Klugman, Panjer and Willmot on aggregate loss models 85
Frees on severity, coverage modifications and aggregate losses 90
Bahnemann on aggregate distributions and layer pricing 93
Brehm et al. on reinsurance in enterprise risk analysis 96
Casualty Actuarial Society examination questions 97

References 98

Preface

A recitation is the class that sits beside a lecture, where an instructor works problems from the assigned text. This book is a recitation for actuarial textbooks and examinations. It works their examples and exercises with `aggregate`, a Python package for compound probability distributions with its own specification language, DecL. Every number in it is computed when the book is rendered, never transcribed, so an answer here can be set beside the one printed in the text.

The book is written for teachers and students. A teacher can use a chapter to show how a problem posed for hand calculation becomes a short program, and how the same program answers every variation on the question. A student can use it to check their own work, and to see the whole distribution where the text asks only for a mean or a single probability. We assume the source text is open beside this one: each chapter is named for the book it follows, and each section for the example or exercise it works.

The problems fall into three categories. A **translation problem** states a frequency, a severity, and a coverage, and asks for a moment or a probability; most of *Loss Models* Chapter 9 and *Distributions for Actuaries* Chapter 4 fall into this category. The work is writing the DecL, and the text is the authority: the program should reproduce its answer. The value to a student is everything else that comes with the answer, because the program computes the whole distribution and not just the requested quantity.

An **approximation problem** asks for a normal, lognormal, gamma, or compound Poisson approximation to an aggregate distribution. The texts teach approximations because historically the exact distribution was expensive to compute. Here the roles reverse: `aggregate` computes the exact answer, and the approximation becomes the quantity being checked.

An **application problem** uses the distribution to make a decision: increased limit factors and risk loads, aggregate limits and deductible credits, a reinsurance program, an allocation of capital. These problems carry the most arithmetic outside the distribution itself, and they show how far a short program reaches into actuarial practice.

Companion volumes

Three volumes divide the material. *Recitations* holds worked examples and exercises from textbooks, study texts, and examinations. *Reproductions* holds results from published research papers, rebuilt with `aggregate` and checked against the papers' own tables and figures. The monograph, *Working with Aggregate Probability Distributions*, holds original teaching material and the author's own analysis, including analysis built on top of a published example. A reader new to aggregate distributions should start with the monograph's two student chapters, the Student guide and the Actuarial Student guide. They are original material, with no published answer to check against, so they live in the monograph rather than here.

Running the programs

The aggregate Loss Lab, aLL for short, runs the library in the browser with no installation, and is the quickest way to try a program from this book. The same programs run through `build` in any Python session with `aggregate` installed. The appendix collects every program the book builds, chapter by chapter, and `recitations.agg`, beside the book's source, holds the same programs as one file that `build.load` reads, each named for the text and noted with the example or exercise it works.

About this edition

This book reports the environment that computed it. Every chapter is recomputed whenever the library or any other installed package changes, so all the numbers in a given copy come from the one environment described below.

```
Exception reporting mode: Minimal
Rendered 2026-10-22 19:54:35 GMT Daylight Time (UTC+0100)
----------------------------------------------------------
Python           | 3.14.7 on Windows-11-10.0.26200-SP0
aggregate        | 1.7.1, source at git commit 0036744
greater_tables   | 8.1.0
```

1. Klugman, Panjer and Willmot on aggregate loss models

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

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.

1.1. 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...

1.2. 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.

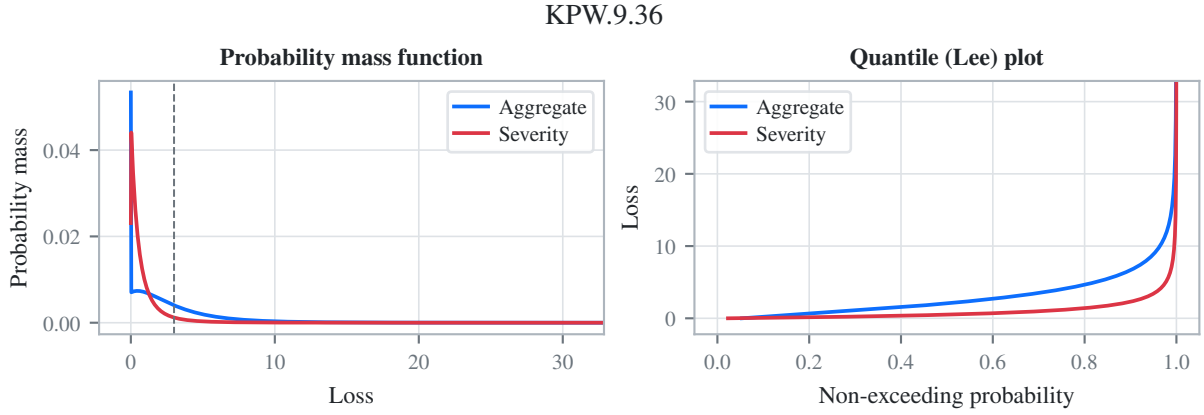
Exercise 9.36. The number of losses is Poisson with mean 3. Loss amounts have a Burr distribution with $\alpha = 3$, $\theta = 2$, and $\gamma = 1$. Determine the variance of aggregate losses.

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
```



1.3. 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
```

1.4. 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
```

1.5. 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
```

1.6. 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();
```

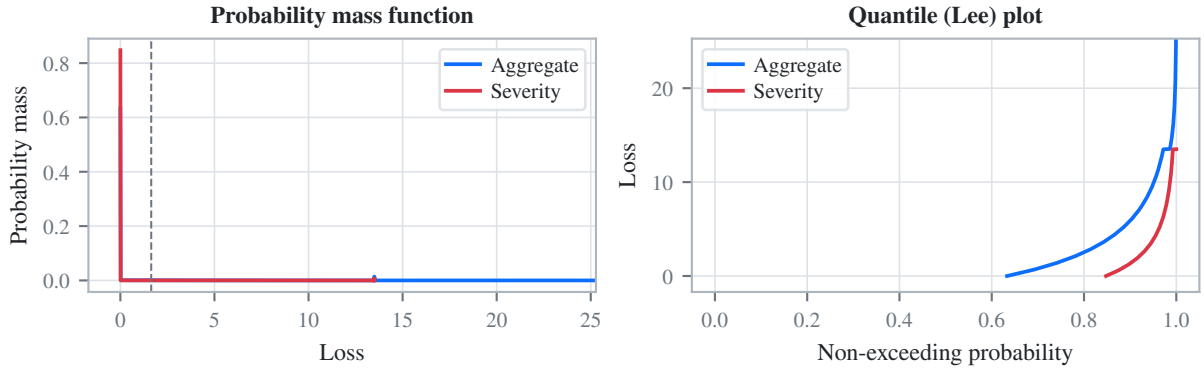
```
Aggregate object: KPW.9.14. Frequency distribution poisson. Severity
pareto, [0, 18], bounded; 18 xs 6. Updated with bucket size 1/128 and
log2 = 16. Validation: reinsurance; subject not unreasonable.

        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).

1.7. 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
```

1.8. 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

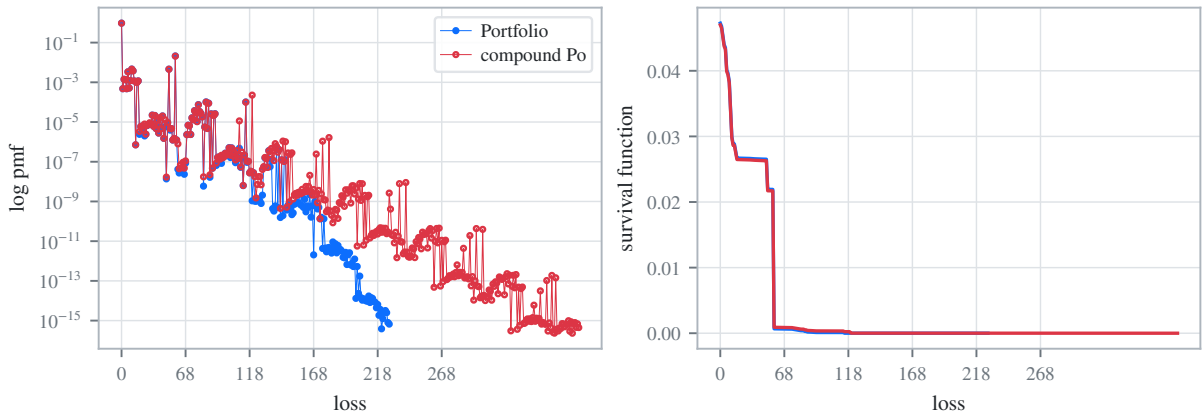
88	2.598867e-05	0.9998422	0.0001577853	2.540443e-05	0.9995993	0.0004006766
89	4.722525e-08	0.9998423	0.0001577381	4.726622e-08	0.9995994	0.0004006293
90	2.194831e-05	0.9998642	0.0001357898	2.147808e-05	0.9996208	0.0003791512
91	2.623287e-05	0.9998904	0.0001095569	2.56655e-05	0.9996465	0.0003534857
92	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
```

2. Frees on severity, coverage modifications and aggregate losses

Frees (2018), Loss Data Analytics, open online text, example numbers as of November 2022

Examples from Jed Frees' open source actuarial software. Text and github source available on-line.

2.1. Distribution Examples

2.1.1. Gamma distribution

```
import scipy.stats as ss
import numpy as np
import matplotlib.pyplot as plt

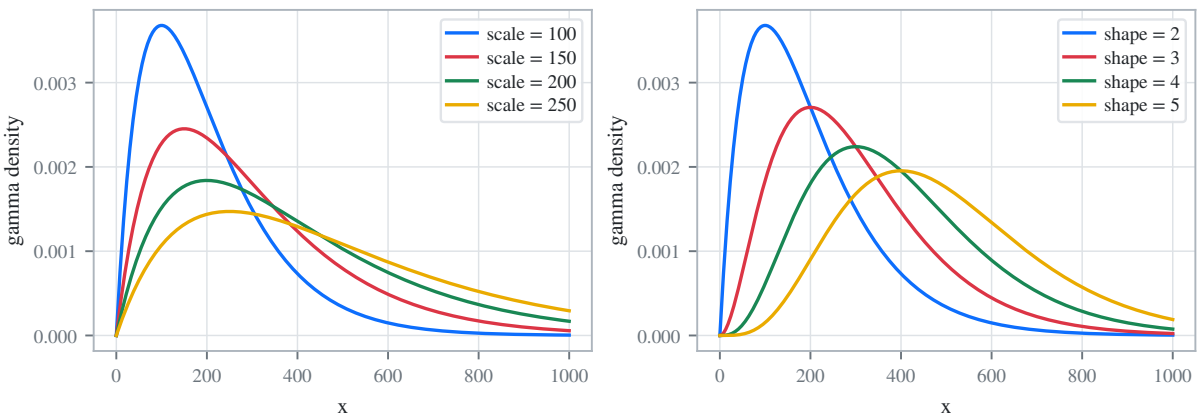
xs = np.linspace(0, 1000, 1001)

fig, axs = plt.subplots(1, 2, figsize=(2 * 3.5, 2.45), constrained_layout=True, squeeze=True)
ax0, ax1 = axs.flat

for scale in [100, 150, 200, 250]:
    ax0.plot(xs, ss.gamma(2, scale=scale).pdf(xs), label=f'scale = {scale}')

for shape in [2, 3, 4, 5]:
    ax1.plot(xs, ss.gamma(shape, scale=100).pdf(xs), label=f'shape = {shape}')

for ax in axs.flat:
    ax.legend(loc='upper right')
    ax.set(ylabel='gamma density', xlabel='x')
```



2.1.2. Pareto distribution

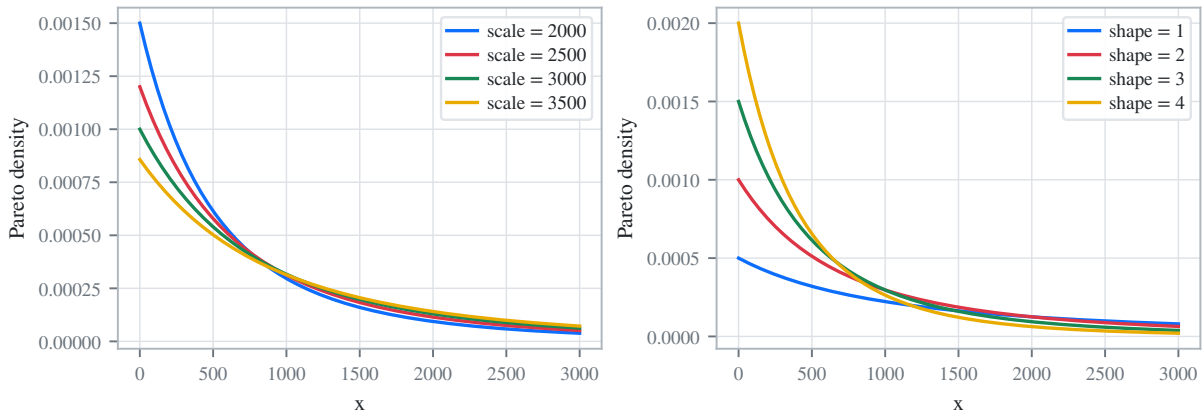
```
xs = np.linspace(0, 3000, 3001)

fig, axs = plt.subplots(1, 2, figsize=(2 * 3.5, 2.45), constrained_layout=True, squeeze=True)
ax0, ax1 = axs.flat

for scale in [2000, 2500, 3000, 3500]:
    ax0.plot(xs, ss.pareto(3, scale=scale, loc=-scale).pdf(xs), label=f'scale = {scale}')

for shape in [1,2,3,4]:
    ax1.plot(xs, ss.pareto(shape, scale=2000, loc=-2000).pdf(xs), label=f'shape = {shape}')

for ax in axs.flat:
    ax.legend(loc='upper right')
    ax.set(ylabel='Pareto density', xlabel='x')
```



2.1.3. Weibull distribution

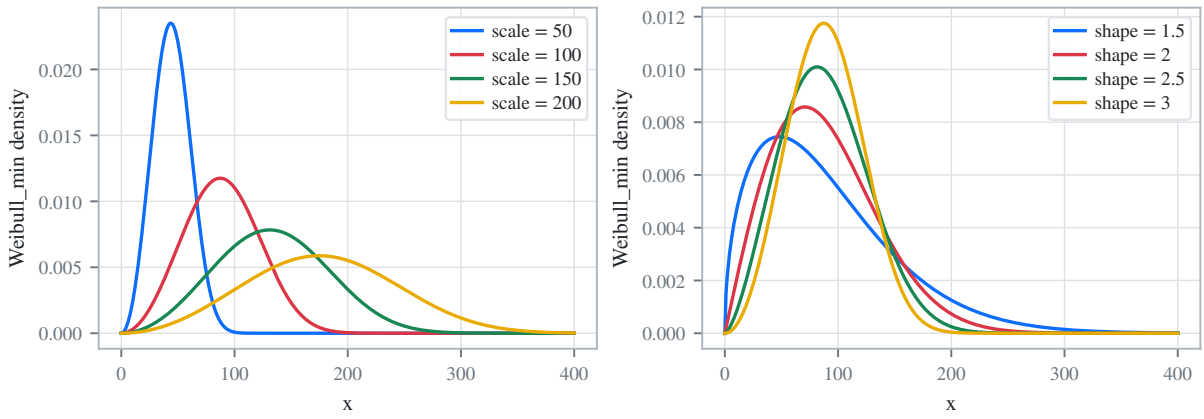
`scipy.stats` includes Weibull min (for positive x) and Weibull max (for negative x) distributions. We want the min version.

```
xs = np.linspace(0, 400, 401)

fig, axs = plt.subplots(1, 2, figsize=(2 * 3.5, 2.45), constrained_layout=True, squeeze=True)

ax0, ax1 = axs.flat

for scale in [50, 100, 150, 200]:
    ax0.plot(xs, ss.weibull_min(3, scale=scale).pdf(xs), label=f'scale = {scale}')
for shape in [1.5, 2, 2.5, 3]:
    ax1.plot(xs, ss.weibull_min(shape, scale=100).pdf(xs), label=f'shape = {shape}')
for ax in axs.flat:
    ax.legend(loc='upper right')
    ax.set(ylabel='Weibull_min density', xlabel='x')
```



2.2. Mixture Example (3.3.5)

A collection of insurance policies consists of two types. 25% of policies are Type 1 and 75% of policies are Type 2. For a policy of Type 1, the loss amount per year follows an exponential distribution with mean 200, and for a policy of Type 2, the loss amount per year follows a Pareto distribution with parameters $\alpha = 3$ and $\theta = 200$. For a policy chosen at random from the entire collection of both types of policies, find the probability that the annual loss will be less than 100, and find the average loss.

Solution. The function `pmv` (print mean and variance) is a convenience.

```
from aggregate import qd, mv
def pmv(m, v):
    print(f'mean     = {m:.6g}\n'
          f'variance = {v:.7g}')
```

Create the `Aggregate` object, display its `summary_df` dataframe and compare the cdf with the exact computation.

```
a = build('agg lda.3.3.5 '
          '1 claim '
          'sev [200 200] * [expon pareto] [1 3] + [0 -200] wts [.25 .75] '
          'fixed',
          normalize=False)
qd(a)
a.sev.cdf(100), 0.25 * (1 - np.exp(-0.5)) + 0.75 * (1 - (2/3)**3)
```

```
Aggregate object: lda.3.3.5. Frequency distribution fixed. Mixed
severity with 2 components. Updated with bucket size 4 and log2 = 16.
Validation: not unreasonable.

       Mean     SD     CV   Skew P01 Median P99
X
Freq      1      0      0
Sev  124.99 185.29 1.4824 15.629   0     64 816
Agg  124.99 185.29 1.4824 15.629   0     64 816

(np.float64(0.6261451128496194), np.float64(0.6261451128496194))
```

This example has a very thick tailed severity and it is best to specify `normalized=False` for the most accurate severity estimates. With default settings, `aggregate` suffers considerable discretization error, with an estimated mean well below the actual 125. The `sev.cdf` method exposes the actual underlying severity distribution cdf functions and reproduces the requested probability exactly. The object cdf function relies on the discretization and so is shifted by half a bucket size. (Also available: `sev.sf` and `sev.pdf`.)

```
a.cdf(100), a.sev.cdf(100 + a.bs/2)
```

```
(np.float64(0.6320397201726607), np.float64(0.6320397201726609))
```

2.3. Coverage Modifications

Deductible Example (3.4.1)

A claim severity distribution is exponential with mean 1000. An insurance company will pay the amount of each claim in excess of a deductible of 100. Calculate the variance of the amount paid by the insurance company for one claim, including the possibility that the amount paid is 0.

Solution. In this case we must use unconditional severity to include the possibility that the amount paid is 0. This is done by adding `!` at the end of the severity specification. The moments are computed exactly without updating.

```
import numpy as np
a = build('agg lda.3.4.1 1 claim '
          'inf xs 100 sev 1000 * expon 1 ! '
          'fixed', update=False)
qd(a)
m = 1000 * np.exp(-0.1)
mv(a)
pmv(m, (2 * 1000**2 * np.exp(-0.1)) - m**2)
```

```
Aggregate object: lda.3.4.1. Frequency distribution fixed. Severity
expon, [0, inf), exponential right tail; inf xs 100. Validation: n/a,
not updated.

       Mean     SD     CV   Skew P01 Median P99
X
Freq      1      0      0
Sev  904.84 995.46 1.1002 2.0257
Agg  904.84 995.46 1.1002 2.0257
mean     = 904.837
variance = 990944.1
std dev  = 995.462
mean     = 904.837
variance = 990944.1
```

Deductible Example (3.4.2)

For an insurance:

- Losses have a density function

$$f_X(x) = \begin{cases} 0.02x & 0 < x < 10, \\ 0 & \text{elsewhere.} \end{cases}$$

- The insurance has an ordinary deductible of 4 per loss.
- Y^P is the claim payment per payment random variable.

Solution. The trick here is to realize that X is a beta variable with $\alpha = 2$ and $\beta = 1$.

```
a = build('agg lda.3.4.2 1 claim 6 xs 4 sev 10 * beta 2 1 fixed')
qd(a)
mv(a)
```

```
Aggregate object: lda.3.4.2. Frequency distribution fixed. Severity
beta, [0, 6], bounded; 6 xs 4. Updated with bucket size 1/8,192 and log2
= 16. Validation: not unreasonable.

       Mean     SD      CV     Skew     P01 Median    P99
X
Freq      1      0       0
Sev  3.4286 1.6782 0.48947 -0.29313 0.10364 3.6157 5.9579
Agg  3.4286 1.6782 0.48947 -0.29313 0.10364 3.6157 5.9579
mean     = 3.42857
variance = 2.816327
std dev  = 1.67819
```

Limit Example (3.4.4)

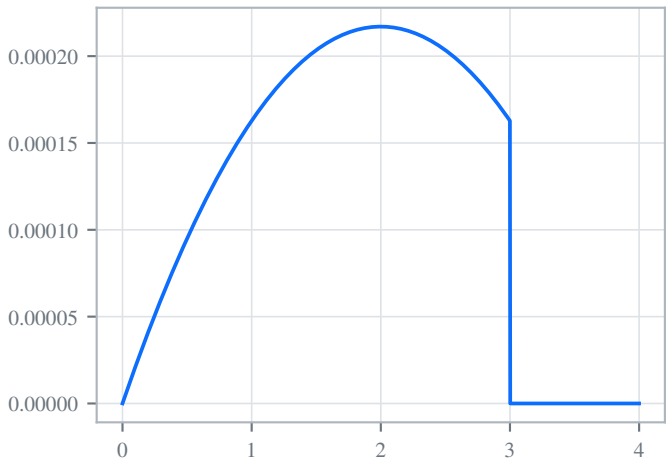
Under a group insurance policy, an insurer agrees to pay 100% of the medical bills incurred during the year by employees of a small company, up to a maximum total of one million dollars. The total amount of bills incurred, X, has *pdf*

$$f_X(x) = \begin{cases} \frac{x(4-x)}{9} & 0 < x < 3 \\ 0 & \text{elsewhere.} \end{cases}$$

where x is measured in millions. Calculate the total amount, in millions of dollars, the insurer would expect to pay under this policy.

Solution. In this case the distribution has no obvious parametric form, though it is related to a beta. We can solve it in `aggregate` by using a custom empirical distribution.

```
xs = np.linspace(0, 4, 2**13, endpoint=False)
F = np.where(xs<3,(xs * xs  * (2 - xs / 3)) / 9, 1)
ps = np.diff(F, append=1)
fig, ax = plt.subplots(1, 1, figsize=(3.5, 2.45), constrained_layout=True, squeeze=True)
ax.plot(xs, ps);
```



When the empirical distribution has many entries it is faster to build the `Aggregate` object directly, rather than use DecL. The moments of the severity and aggregate distribution are computed from the numerical approximation during creation. There is no need to update the object.

```
from aggregate import Aggregate

a = Aggregate('Example', exp_en=1, sev_name='dhistogram', sev_xs=xs, sev_ps=ps,
             exp_attachment=0, exp_limit=1, freq_name='fixed')
print(a)
```

```
aggregate object name    Example
value_type               loss
claim count              1.000
frequency distribution   fixed
severity distribution    dhistogram, [0, 4], bounded; 8192 atoms [0 0.000488281 0.000976562 ...
3.99902 3.99951] on [0, 3.99951]
approximate              exact
bs                       n/a
log2                     n/a
padding                  n/a
sev_calc                 n/a
dsev_bucket              linear
normalize                n/a
x_min                    n/a
x_max                    n/a
premium                  n/a
expected loss            n/a
loss ratio               n/a
P(X=0)                   n/a
validation_eps           0.0001
reinsurance              none
occurrence reinsurance   no reinsurance
aggregate reinsurance    no reinsurance
validation               n/a, not updated
frequency tail           fixed, count [1, 1], bounded
severity tail            dhistogram, [0, 4], bounded
aggregate tail           [0, 4], bounded; not concentrated (cv=0.183)
bounded                  True
id                       DC24AC39
        Mean      SD     CV    Skew  P01  Median  P99
X
Freq       1       0      0     NaN  NaN     NaN  NaN
Sev  0.93514 0.17076 0.1826 -2.9019  NaN     NaN  NaN
Agg  0.93514 0.17076 0.1826 -2.9019  NaN     NaN  NaN
```

Limit and Deductible Example (3.4.5)

The ground up loss random variable for a health insurance policy in 2006 is modeled with X, a random variable with an exponential distribution having mean 1000. An insurance policy pays the loss above an ordinary deductible of 100, with a maximum annual payment of 500. The ground up loss random variable is expected to be 5% larger in 2007, but the insurance in 2007 has the same deductible and maximum payment as in 2006. Find the percentage increase in the expected cost per payment from 2006 to 2007.

Solution. Trend increases the ground-up severity distribution but not the limit and attachment. The calculation is performed exactly on creation; again, there is no need to update the `Aggregate` object.

```
import pandas as pd

a06 = build('agg X06 1 claim 500 xs 100 sev 1000 * expon fixed', update=False)
a07 = build('agg X07 1 claim 500 xs 100 sev 1050 * expon fixed', update=False)
ans = pd.concat((a06.summary_df, a07.summary_df), keys=['2006', '2007'])
qd(ans)
ans.iloc[5, 0] / ans.iloc[2, 0] - 1
```

```
            Mean     SD      CV    Skew  P01  Median  P99
     X
2006 Freq      1      0       0     NaN  NaN     NaN  NaN
     Sev  393.47 159.97 0.40656 -1.1717  NaN     NaN  NaN
     Agg  393.47 159.97 0.40656 -1.1717  NaN     NaN  NaN
2007 Freq      1      0       0     NaN  NaN     NaN  NaN
     Sev   397.8 157.89 0.39691 -1.2342  NaN     NaN  NaN
     Agg   397.8 157.89 0.39691 -1.2342  NaN     NaN  NaN

np.float64(0.011000207063778467)
```

Reinsurance Example (3.4.6, modified)

Losses arising in a certain portfolio have a two-parameter Pareto distribution with $\alpha = 5$ and $\theta = 3,600$. A reinsurance arrangement has been made, under which (a) the reinsurer accepts 15% of losses up to $u = 5,000$ and all amounts in excess of 5,000 and (b) the insurer pays for the remaining losses.

1. Express the random variables for the reinsurer's and the insurer's payments as a function of X, the portfolio losses.
2. Calculate the mean amount paid on a single claim by the insurer.
3. Calculate the standard deviation of the amount paid on a single claim by the insurer (retaining the 15% copayment).

Solution. The net position can be modeled as:

```
agg insurer.net 1 claim
sev 3600 * pareto 5 - 3600
occurrence net of 15% po 5000 xs 0 and inf xs 5000
fixed
```

but this involves the thick-tailed Pareto across its entire range. It is better to recognize the severity is limited by the second excess layer and proceed as follows.

```
a = build('agg insurer.net 1 claim '
      '5000 xs 0 sev 3600 * pareto 5 - 3600 '
      'occurrence net of 15% po 5000 xs 0 '
      'fixed')
qd(a)
print('\n', a.actual_m, a.actual_sd)
```

```
Aggregate object: insurer.net. Frequency distribution fixed. Severity
pareto, [0, 5,000], bounded; 5,000 xs 0. Updated with bucket size 1/8
and log2 = 16. Validation: reinsurance; subject not unreasonable.

       Mean     SD     CV   Skew   P01 Median  P99
X
Freq      1      0      0
Sev  741.51 834.64 1.1256 2.0952 6.125    455 4250
Agg  741.51 834.64 1.1256 2.0952 6.125    455 4250
```

```
872.3650484486673 981.9314221673877
```

> **Note**
>
> In DecL a percentage is a share and a plain number is an amount. `15% po 5000 xs 0` places 15% of the layer 5,000 xs 0, and `750 po 5000 xs 0` places the same 15% as an amount, 750 of the 5,000 limit. A decimal written in place of the percentage, `0.15 po 5000 xs 0`, is therefore read as an amount of 0.15, a placement of 0.003% of the layer, and `aggregate` warns when a placement is that small. The insurer's mean checks by hand: it is $0.85\,\mathsf{P}(X \wedge 5000) = 0.85 \times 900\,(1 - (3600/8600)^4) = 0.85 \times 872.37 = 741.51$.

2.4. Aggregate Loss Distributions

Poisson/Discrete Example (5.3.1)

The number of accidents follows a Poisson distribution with mean 12. Each accident generates 1, 2, or 3 claimants with probabilities 1/2, 1/3, and 1/6 respectively.

Calculate the variance in the total number of claimants.

Solution.

```
a = build('agg QU 12 claims dsev [1 2 3] [1/2 1/3 1/6] poisson')
qd(a)
mv(a)
```

```
Aggregate object: QU. Frequency distribution poisson. Severity
dhistogram, [1, 3], bounded; atoms [1 2 3]. Updated with bucket size 1
and log2 = 7. Validation: not unreasonable.

       Mean      SD      CV    Skew P01 Median P99
X
Freq     12  3.4641 0.28868 0.28868
Sev  1.6667 0.74536 0.44721  0.6261   1      1   3
Agg      20  6.3246 0.31623 0.36366   7     20  36
mean     = 20
variance = 40
std dev  = 6.32456
```

As always, `a` contains the (exact) full distribution of outcomes. We could answer any question about it.

Discrete Example (5.3.2)

You are the producer of a television quiz show that gives cash prizes. The number of prizes, N, and prize amount, X, have the following distributions:

n	$\Pr(N = n)$	x	$\Pr(X = x)$
1	0.8	0	0.2
2	0.2	100	0.7
		1000	0.1

Your budget for prizes equals the expected aggregate cash prizes plus the standard deviation of aggregate cash prizes. Calculate your budget.

Solution. Just a matter of translating into DecL. No need to update the object.

```
a = build('agg lda.5.3.2 dfreq [1 2] [.8 .2] '
          'dsev [0 100 1000] [.2 .7 .1]', update=False)
display(a)
mv(a)
a.actual_m + a.actual_sd
```

X	Mean	SD	CV	Skew	P01	Median	P99
Freq	1.2	0.4	0.33333	1.5			
Sev	170	279.46	1.6439	2.5637			
Agg	204	313.6	1.5372	2.3992			

```
mean     = 204
variance = 98344
std dev  = 313.598

517.5984693840198
```

Geometric/Discrete Example (5.3.3 and 5.4.1)

The number of claims in a period has a geometric distribution with mean 4. The amount of each claim X follows $\Pr(X = x) = 0.25,\ x = 1, 2, 3, 4$, i.e. a discrete uniform distribution on $\{1, 2, 3, 4\}$. The number of claims and the claim amounts are independent. Let S_N denote the aggregate claim amount in the period. Calculate $F_{S_N}(3)$.

Solution. We can compute the entire distribution. Here we show up to the 99th percentile. If the probability clause in `dsev` is omitted then all outcomes are treated as equally likely.

```
a = build('agg lda.5.3.3 4 claims dsev [1:4] geometric')
qd(a, accuracy=4)
b = a.density_df.loc[0:a.q(0.99), ['p_total', 'F']]
b.index = b.index.astype(int)
qd(b, accuracy=4)
```

```
Aggregate object: lda.5.3.3. Frequency distribution geometric. Severity
dhistogram, [1, 4], bounded; atoms [1 2 3 4]. Updated with bucket size 1
and log2 = 8. Validation: not unreasonable.

      Mean     SD      CV   Skew P01 Median P99
X
Freq     4 4.4721   1.118 2.0125
Sev    2.5  1.118 0.44721      0   1      2   4
Agg     10 11.402  1.1402  2.024   0      6  51

        p_total       F
loss
0           0.2     0.2
1          0.04    0.24
2         0.048   0.288
3        0.0576  0.3456
4       0.06912 0.41472
5      0.042944 0.45766
6      0.043533  0.5012
7      0.042639 0.54384
8      0.039647 0.58348
9      0.033753 0.61724
```

```
10     0.031914 0.64915
11     0.029591 0.67874
...         ...     ...
40    0.0023287 0.97449
41    0.0021339 0.97662
42    0.0019554 0.97858
43    0.0017918 0.98037
44     0.001642 0.98201
45    0.0015046 0.98352
46    0.0013788  0.9849
47    0.0012634 0.98616
48    0.0011577 0.98732
49    0.0010609 0.98838
50   0.00097217 0.98935
51   0.00089085 0.99024
```

Moments Example (5.3.4)

You are given:

	Mean	Standard Deviation
Number of Claims	8	3
Individual Losses	10,000	3,937

As a benchmark, use the normal approximation to determine the probability that the aggregate loss will exceed 150% of the expected loss.

Solution. Use the `MomentAggregator` class to compute the moments of an aggregate from those of frequency and severity.

```
import scipy.stats as ss
from aggregate import MomentAggregator
mom = MomentAggregator.agg_from_fs2(8, 9, 10000, 3937**2)
fz = ss.norm(loc=mom.ex, scale=mom.sd)
mom['prob'] = fz.sf(1.5*mom.ex)
qd(mom)
```

```
ex         80000
var    1.024e+09
sd         32000
cv             0.4
prob     0.10565
```

Poisson/Uniform Example (5.3.5 and 5.4.2)

For an individual over 65:

1. The number of pharmacy claims is a Poisson random variable with mean 25.
2. The amount of each pharmacy claim is uniformly distributed between 5 and 95.
3. The amounts of the claims and the number of claims are mutually independent.

Estimate the probability that aggregate claims for this individual will exceed 2000 using the normal approximation.

Solution. Here is a close-to exact solution in addition to the normal approximation. Note that the uniform distribution has no shape parameter. The severity is made by shifting and scaling the base. Scaling is like multiplication and is applied before the location (addition) shift.

```
a = build('agg Pharma 25 claims sev 90 * uniform + 5 poisson')
qd(a)
```

```
Aggregate object: Pharma. Frequency distribution poisson. Severity
uniform, [0, 95], bounded. Updated with bucket size 1/8 and log2 = 16.
Validation: not unreasonable.

      Mean     SD      CV    Skew    P01 Median    P99
X
Freq    25      5     0.2     0.2
Sev     50 25.981 0.51962       0  5.875     50 94.125
Agg   1250 281.74 0.22539 0.25293 649.62 1238.1 1955.8
```

Here are the moments for the approximation. The `approximate` function returns a `scipy.stats` frozen normal object, which yields the approximation.

```
print(a.sf(2000), a.actual_m, a.actual_var)

fz = a.approximate('norm')
fz.sf(2000), a.sf(2000)
```

```
0.006980215362075826 1250.0 79375.0

(np.float64(0.00388310433626233), np.float64(0.006980215362075826))
```

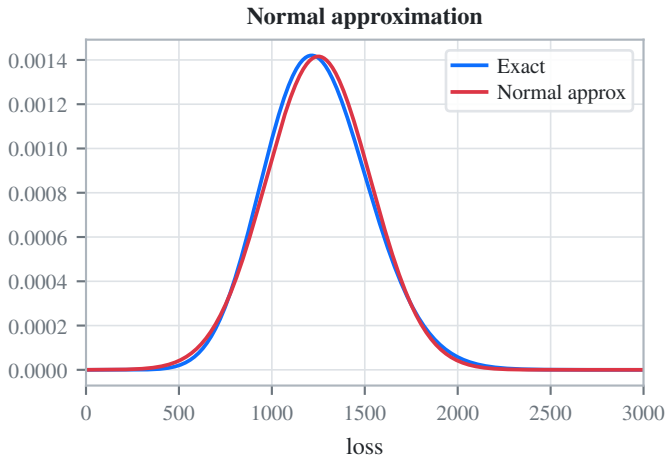
`approximate` will also provide (shifted) gamma and lognormal fits.

```
approx = a.approximate('all')
b = pd.DataFrame([[k, v.sf(2000)] for k, v in approx.items()],
             columns=['approx', 'prob']).set_index('approx')
b.loc['exact'] = a.sf(2000)
b.sort_values('prob')
```

approx	prob
norm	0.003883
exact	0.006980
sgamma	0.007101
slognorm	0.007175
gamma	0.009789
lognorm	0.013118

Here is a comparison of the FFT model with the normal approximation. Example 5.4.2 derives a similar probability using simulation.

```
fig, ax = plt.subplots(1, 1, figsize=(3.5, 2.45), constrained_layout=True, squeeze=True)
(a.density_df.p / a.bs).plot(label='Exact', ax=ax);
ax.plot(a.xs, fz.pdf(a.xs), label='Normal approx');
ax.set(xlim=[0, 3000], title='Normal approximation');
ax.legend(loc='upper right');
```



Geometric/Discrete Example (5.3.6 and 5.3.7)

In a given week, the number of projects that require you to work overtime has a geometric distribution with $\beta = 2$. For each project, the distribution of the number of overtime hours in the week, X, is as follows:

x	$f(x)$
5	0.2
10	0.3
20	0.5

The number of projects and the number of overtime hours are independent. You will get paid for overtime hours in excess of 15 hours in the week. Calculate the expected number of overtime hours for which you will get paid in the week.

Solution. This is a one-liner in `aggregate`. Remember that aggregate reinsurance is specified after the frequency clause. The first column in the describe dataframe shows the analytic gross answer and the second the FFT-computed net.

```
a = build('agg Projects 2 claims '
          'dsev [5 10 20] [.2 .3 .5] geometric '
          'aggregate net of 15 xs 0')
qd(a)
```

```
Aggregate object: Projects. Frequency distribution geometric. Severity
dhistogram, [5, 20], bounded; atoms [5 10 20]. Updated with bucket size
5 and log2 = 7. Validation: reinsurance; subject not unreasonable.

       Mean     SD      CV     Skew P01 Median P99
X
Freq      2 2.4495  1.2247   2.0412
Sev      14  6.245 0.44607 -0.23403   5     10  20
Agg  18.807 31.433  1.6713   2.6377   0      5 140
```

Example 5.3.7 uses a recursive calculation in steps of 5. We can replicate that using an aggregate excess cover. The cumulative excess (overtime) at attachment `A` is the ceded mean of an `aggregate ceded to inf xs A` cover, read from `reins_summary_df` (the `('agg', 'ceded', 'agg')` row, column `Est EX`).

```
import numpy as np
rows = []
for A in [0, 5, 10, 15]:
    aA = build('agg Projects.1 2 claims '
               'dsev [5 10 20] [.2 .3 .5] geometric '
               f'aggregate ceded to inf xs {A}')
    rows.append([A, aA.reins_summary_df.loc[('agg', 'ceded', 'agg'), 'Est EX']])
b = pd.DataFrame(rows, columns=['attach', 'cumul ex']).set_index('attach')
qd(b, accuracy=4)
```

```
        cumul ex
attach
0             28
5         24.667
10        21.556
15        18.807
```

> **Note**
>
> This example previously used the per-layer `reinsurance_audit_df`, removed in 1.0.0a19 in favour of the whole-structure `reins_summary_df` / `reins_stats_df` / `reins_density_df` objects. The excess at each attachment is recovered above by building one `ceded to inf xs A` cover per attachment and reading its ceded aggregate mean.

Zero-Modified Poisson/Burr Example (5.5.4)

Aggregate losses are modeled as follows:

1. The number of losses follows a zero-modified Poisson distribution with $\lambda = 3$ and $p_0^M = 0.5$.
2. The amount of each loss has a Burr distribution with $\alpha = 3, \theta = 50, \gamma = 1$.
3. There is a deductible of $d = 30$ on each loss.
4. The number of losses and the amounts of the losses are mutually independent.

Calculate $\mathsf{E}(N^P)$ and $\mathsf{Var}(N^P)$.

Solution. This is a pure frequency question: the deductible thins the loss count N^L into the payment count N^P. The problem states the zero-modified Poisson the textbook way (base parameter $\lambda = 3$ plus the modified zero mass p_0^M), which is exactly how DecL reads it: the exposure clause gives the *un-modified* mean and the $(a, b, 1)$ reweighting moves it. So `3 claims ... poisson zm 0.5` is a base Poisson(3), and $\mathsf{E}(N^L)$ is an output.

```
import scipy.stats as ss

# Burr(alpha=3, theta=50, gamma=1) is scipy's burr12(c=gamma, d=alpha)
burr = ss.burr12(1, 3, scale=50)
v = burr.sf(30)                     # survival at the deductible
v
```

```
np.float64(0.24414062499999994)
```

The frequency carries the whole calculation. `apply_deductible` implements the Loss Models §8.6 map: the base parameter scales by v and the zero mass becomes $\alpha^* = P_{N^L}(1 - v)$.

```
a = build('agg LDA554 3 claims sev 50 * burr12 1 3 poisson zm 0.5')
loss_count = a.frequency
loss_count.base_mean, loss_count.modify_mean(), loss_count.prob_eq_0

pay_count = loss_count.apply_deductible(v)
m1, m2, _ = pay_count.freq_moms(pay_count.base_mean)
print(f'lambda* = {pay_count.base_mean:.6f}, p0M* = {pay_count.freq_p0:.6f}')
print(f'E(N^P)   = {m1:.6f}')
print(f'Var(N^P) = {m2 - m1 * m1:.6f}')
```

```
lambda* = 1.461158, p0M* = 0.615015
E(N^P)   = 0.732422
Var(N^P) = 1.266164
```

The deductible has made "no payment this period" much more likely ($p_0^{M*} = 0.7268$ against $p_0^M = 0.5$), which is the general rule of Loss Models §8.6: a zero-*truncated* loss count always yields a zero-*modified* payment count.

As a check, any independent-thinning model must satisfy $\mathsf{E}(N^P) = v\,\mathsf{E}(N^L)$ and $\mathsf{Var}(N^P) = v^2\mathsf{Var}(N^L) + v(1-v)\mathsf{E}(N^L)$, which is an entirely different route to the same two numbers.

```
f1, f2, _ = loss_count.freq_moms(loss_count.base_mean)
var_nl = f2 - f1 * f1
v * f1, v ** 2 * var_nl + v * (1 - v) * f1
```

```
(np.float64(0.7324218750000012), np.float64(1.2661638068680663))
```

Negative Binomial Example (5.5.5 and 5.5.6 modified)

A group dental policy has a negative binomial claim count distribution with mean 300 and variance 800. Ground-up severity is given by the following table:

Severity	Probability
40	0.25
80	0.25
120	0.25
200	0.25

You expect severity to increase 50% with no change in frequency. You decide to impose a per claim deductible of 100. Calculate the expected total claim payment S after these changes. What is the variance of the total claim payment, $\mathsf{Var}(S)$? (Modified:) Compare the aggregate distributions before and after the policy change.

Solution. A negative binomial with mean 300 and variance 800 has $8/3 = 1 + 300c$ and giving a mixing cv of $\sqrt{c} = (5/900)^{0.5} = 0.0745$. Hence the aggregate program is

TODO: why is the answer not exact?

```
cv = ((8 / 3 - 1) / 300)**0.5
a0 = build(f'agg Original 300 claims dsev [4 8 12 20] mixed gamma {cv}')
a1 = build(f'agg Revised 300 claims inf xs 10 '
           f'sev dhistogram xps [{4*1.5} {8*1.5} {12*1.5} {20*1.5}] ! mixed gamma {cv}')
qd(a0)
qd(a1)
mv(a0)
mv(a1)
```

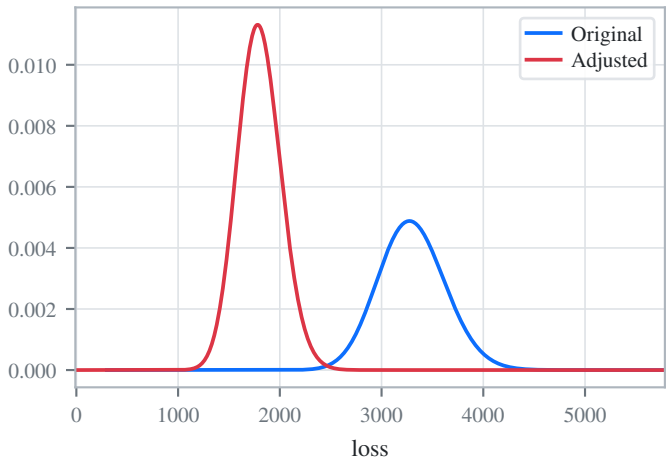
```
Aggregate object: Original. Frequency distribution gamma. Severity
dhistogram, [4, 20], bounded; atoms [4 8 12 20]. Updated with bucket
size 4 and log2 = 11. Validation: not unreasonable.

      Mean     SD       CV    Skew  P01 Median  P99
X
Freq   300 28.284 0.094281 0.15321
Sev     11 5.9161  0.53783 0.43465    4      8   20
Agg   3300 327.57 0.099263 0.15833 2576   3292 4100
Aggregate object: Revised. Frequency distribution gamma. Severity
dhistogram, [6, 30], bounded; atoms [6 12 18 30]. Updated with bucket
size 6 and log2 = 10. Validation: fails sev mean, agg mean.

      Mean     SD       CV    Skew  P01 Median  P99
X
Freq   300 28.284 0.094281 0.15321
Sev      6 7.3485   1.2247  0.8165    0      0   18
Agg   1800 212.13  0.11785 0.17008 1332   1794 2322
mean     = 3300
variance = 107300
std dev  = 327.567
mean     = 2250
variance = 63225
std dev  = 251.446
```

Here is a comparison of the two densities.

```
fig, ax = plt.subplots(1, 1, figsize=(3.5, 2.45), constrained_layout=True, squeeze=True)
a0.density_df.p_total.plot(ax=ax, label='Original');
a1.density_df.p_total.plot(ax=ax, label='Adjusted');
ax.set(xlim=[-10, 1.25 * a0.q(0.9999)]);
ax.legend(loc='upper right');
```



Poisson/Exponential Coverage and Underwriting Modification (Example 5.5.7)

A company insures a fleet of vehicles. Aggregate losses have a compound Poisson distribution. The expected number of losses is 20. Loss amounts, regardless of vehicle type, have exponential distribution with $\theta = 200$. To reduce the cost of the insurance, two modifications are to be made:

1. A certain type of vehicle will not be insured. It is estimated that this will reduce loss frequency by 20%.
2. A deductible of 100 per loss will be imposed.

Calculate the expected aggregate amount paid by the insurer after the modifications.

Solution. The `!` at the end of the severity clause indicates unconditional severity (including zero claims that fail to meet the deductible).

```
a = build(f'agg Auto {20 * 0.8} claims inf xs 100 sev 200 * expon ! poisson')
qd(a)
```

```
Aggregate object: Auto. Frequency distribution poisson. Severity expon,
[0, inf), exponential right tail; inf xs 100. Updated with bucket size
1/4 and log2 = 16. Validation: not unreasonable.

       Mean     SD      CV    Skew    P01 Median    P99
X
Freq     16      4    0.25    0.25
Sev  121.31 183.87  1.5157  2.4172      0  38.75    821
Agg  1940.9 881.11 0.45397 0.68096 364.25   1840 4408.2
```

If severity is conditional there are 16 claims in excess of the deductible, giving a much higher number. The mean severity is still 200 because of the exponential's memoryless property.

```
a = build(f'agg Auto {20 * 0.8} claims inf xs 100 sev 200 * expon poisson')
qd(a)
```

```
Aggregate object: Auto. Frequency distribution poisson. Severity expon,
[0, inf), exponential right tail; inf xs 100. Updated with bucket size
1/2 and log2 = 16. Validation: not unreasonable.

      Mean     SD      CV    Skew  P01 Median    P99
X
Freq    16      4    0.25    0.25
Sev    200    200       1       2    2  138.5    921
Agg   3200 1131.4 0.35355 0.53033 1033 3099.5 6254.5
```

2.5. Portfolio Management

VaR for a Discrete Variable Example (10.3.4)

Consider an insurance loss random variable with the following probability distribution:

$$\Pr[X = x] = \begin{cases} 0.75, & \text{for } x = 1 \\ 0.20, & \text{for } x = 3 \\ 0.05, & \text{for } x = 4. \end{cases}$$

Determine the VaR at $q = 0.6, 0.9, 0.95, 0.95001$.

Solution.

```
a = build('agg VaR 1 claim dsev [1 3 4] [.75 .2 .05] fixed')
[a.q(i) for i in [.6, .9, .95, .9501]]
```

```
[np.float64(1.0), np.float64(3.0), np.float64(3.0), np.float64(4.0)]
```

Multi-Unit Telecom Management Example (10.4.3.3)

You are the Chief Risk Officer of a telecommunications firm. Your firm has several property and liability risks. We will consider:

- X_1, buildings, modeled using a gamma distribution with mean 200 and scale parameter 100.
- X_2, motor vehicles, modeled using a gamma distribution with mean 400 and scale parameter 200.
- X_3, directors and executive officers risk, modeled using a Pareto distribution with mean 1000 and scale parameter 1000.
- X_4, cyber risks, modeled using a Pareto distribution with mean 1000 and scale parameter 2000.

Denote the total risk as $X = X_1 + X_2 + X_3 + X_4$. For simplicity, you assume that these risks are independent. (Later, we will consider the more complex case of dependence.)

To manage the risk, you seek some insurance protection. You wish to manage internally small building and motor vehicles amounts, up to M_1 and M_2, respectively. You seek insurance to cover all other risks. Specifically, the insurer's portion is

$$Y_{insurer} = (X_1 - M_1)_+ + (X_2 - M_2)_+ + X_3 + X_4,$$

so that your retained risk is $Y_{retained} = X - Y_{insurer} = \min(X_1, M_1) + \min(X_2, M_2)$. Using deductibles $M_1 = 100$ and $M_2 = 200$:

1. Determine the expected claim amount of (i) that retained, (ii) that accepted by the insurer, and (iii) the total overall amount.
2. Determine the 80th, 90th, 95th, and 99th percentiles for (i) that retained, (ii) that accepted by the insurer, and (iii) the total overall amount.
3. Compare the distributions by plotting the densities for (i) that retained, (ii) that accepted by the insurer, and (iii) the total overall amount.

Solution. Begin by figuring the gamma and Pareto parameters. For a gamma, the mean equals shape times scale, so shape equals 2 for building and motor. For a Pareto, the mean equals scale / (shape - 1), so shape equals 2 (no variance) for D&O and 3 for cyber (no third moment). The text's figures, however, come from a Pareto that starts at its scale rather than at zero: substituting it adds the two scales, 3,000, to every percentile of the total, and the text's expected losses of 269.05 retained, 5274.41 insured, and 5543.46 in total sit there, not at the 2,600 total the stated parameters imply. In DecL that Pareto is `1000 * pareto 2` without the shift `- 1000` used elsewhere in this chapter, and the programs below follow the text's figures. We model the results using three `Portfolio` objects, one for the retention, one for the insured amount, and one total. In each case the distribution gives total losses; the frequency component is trivial.

Since the insured and total aggregates have no variance it is hard to estimate an appropriate bucket size. The default method uses the standard deviation as a scale factor. We must use judgement (or trial and error), and select `log2=18` and `bs=1` to ensure there is enough "space". Checking the describe dataframe shows these values match the means well by unit and in total. The insured severity must be made unconditional.

```
retained = build('''port retained
    agg building 1 claim 100 xs 0 sev 100 * gamma 2 fixed
    agg motor    1 claim 200 xs 0 sev 200 * gamma 2 fixed
''')
qd(retained)

insured = build('''port insured
    agg building 1 claim inf xs 100 sev 100 * gamma 2 ! fixed
    agg motor    1 claim inf xs 200 sev 200 * gamma 2 ! fixed
    agg d.and.o  1 claim            sev 1000 * pareto 2 fixed
    agg cyber    1 claim            sev 2000 * pareto 3 fixed
''', log2=18, bs=1)
```

```
qd(insured)

total = build('''port total
    agg building 1 claim sev 100 * gamma 2 fixed
    agg motor    1 claim sev 200 * gamma 2 fixed
    agg d.and.o  1 claim sev 1000 * pareto 2 fixed
    agg cyber    1 claim sev 2000 * pareto 3 fixed
''', log2=18, bs=1)
qd(total)
```

```
Portfolio object: retained. Contains 2 aggregate components. Updated
with bucket size 1/128 and log2 = 16. Validation: not unreasonable.

                Mean     SD      CV    Skew    P01 Median P99
unit     X
building Freq      1      0       0
         Sev  89.636   21.5 0.23985 -2.1029 14.859    100 100
         Agg  89.636   21.5 0.23985 -2.1029 14.859    100 100
motor    Freq      1      0       0
         Sev  179.27 42.999 0.23985 -2.1029 29.711    200 200
         Agg  179.27 42.999 0.23985 -2.1029 29.711    200 200
total    Agg  268.91 48.075 0.17878 -1.6928 112.55    300 300
Portfolio object: insured. Contains 4 aggregate components. Updated with
bucket size 1 and log2 = 18. Validation: fails sev mean, agg mean.

                Mean     SD      CV   Skew  P01 Median   P99
unit     X
building Freq      1      0       0
         Sev  110.36 131.34  1.1901  1.757    0     68   564
         Agg  110.36 131.34  1.1901  1.757    0     68   564
motor    Freq      1      0       0
         Sev  220.73 262.68  1.1901  1.757    0    136  1128
         Agg  220.73 262.68  1.1901  1.757    0    136  1128
d.and.o  Freq      1      0       0
         Sev  1992.4 2677.4  1.3438 24.569 1005   1414  9993
         Agg  1992.4 2677.4  1.3438 24.569 1005   1414  9993
cyber    Freq      1      0       0
         Sev  2999.8 1705.7 0.56861 12.863 2007   2520  9283
         Agg  2999.8 1705.7 0.56861 12.863 2007   2520  9283
total    Agg  5323.3   3188 0.59888 16.519 3217   4601 15706
Portfolio object: total. Contains 4 aggregate components. Updated with
bucket size 1 and log2 = 18. Validation: fails sev mean, agg mean.

                Mean     SD      CV   Skew  P01 Median   P99
unit     X
building Freq      1      0       0
         Sev     200 141.42 0.70711 1.4142   15    168   664
         Agg     200 141.42 0.70711 1.4142   15    168   664
motor    Freq      1      0       0
         Sev     400 282.84 0.70711 1.4142   30    336  1328
         Agg     400 282.84 0.70711 1.4142   30    336  1328
d.and.o  Freq      1      0       0
         Sev  1992.4 2677.4  1.3438 24.569 1005   1414  9993
         Agg  1992.4 2677.4  1.3438 24.569 1005   1414  9993
cyber    Freq      1      0       0
         Sev  2999.8 1705.7 0.56861 12.863 2007   2520  9283
         Agg  2999.8 1705.7 0.56861 12.863 2007   2520  9283
total    Agg  5592.2 3190.2 0.57047 16.485 3442   4875 15977
```

The spacing in the agg programs is for clarity. We could also program using `dfreq` as `agg motor dfreq [1] 100 xs 0 sev...`. Next, assemble the requested data elements.

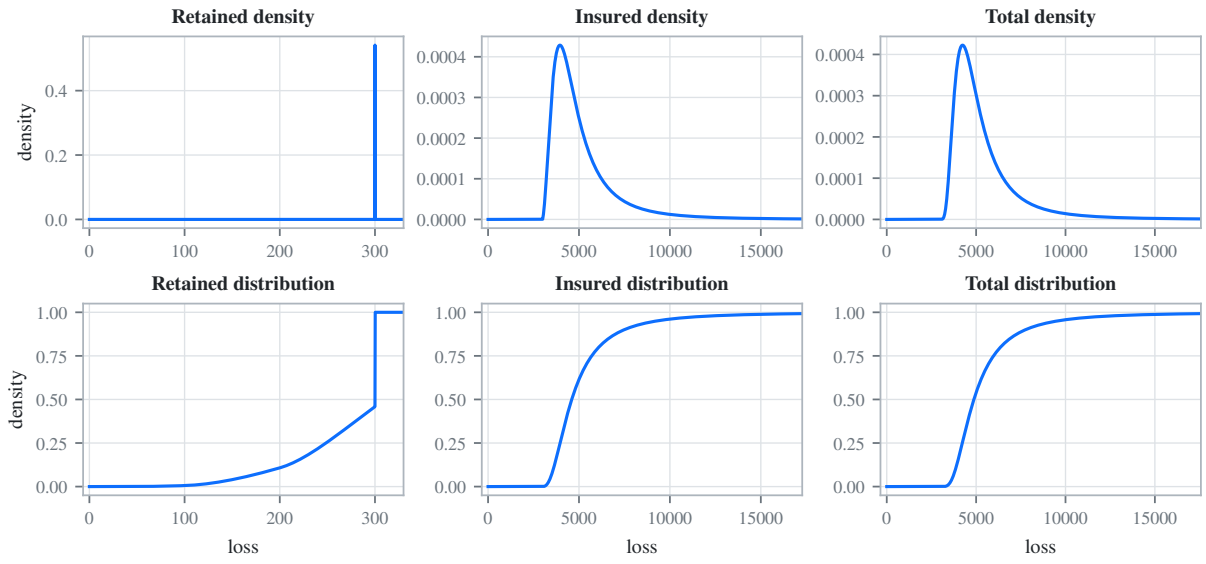
```
pfs = [retained, insured, total]
answers = pd.DataFrame(columns=['retained', 'insured', 'total'])
answers.index.name = 'statistic'
answers.loc['expected claim amount'] = [x.actual_m for x in pfs]
for p in [.8, .9, .95, .99]:
    answers.loc[f'claim p_{p:.2f}'] = [x.q(p) for x in pfs]
qd(answers)
```

statistic	retained	insured	total
expected claim amount	268.91	5331.1	5600
claim p_0.80	300	6090	6364
claim p_0.90	300	7483	7756
claim p_0.95	300	9244	9516
claim p_0.99	300	15706	15977

Finally, plot the densities. Compared to the text plot, the FFT reveals a discontinuous distribution for retained loss, with a large mass at 300. This is clearer on the lower plots, which show the distribution functions.

```
fig, axs = plt.subplots(2, 3, figsize=(7.5, 3.5), constrained_layout=True, squeeze=True)
xl = {}
for ax, pf in zip(axs.flat, pfs):
    pf.density_df.p_total.plot(ax=ax)
    # compute and store a reasonable x range
    q = pf.q(0.99) * 1.1
    xl[hash(pf)] = [-q / 50, q]
    ax.set(xlim=xl[hash(pf)], title=pf.name.title() + ' density')
    if pf is retained:
        ax.set(ylabel='density')

for ax, pf in zip(axs.flat[3:], pfs):
    pf.density_df.F.plot(ax=ax)
    ax.set(xlim=xl[hash(pf)], title=pf.name.title() + ' distribution', xlabel='loss')
    if pf is retained:
        ax.set(ylabel='density')
```



3. Bahnemann on aggregate distributions and layer pricing

Bahnemann (2015), Distributions for Actuaries, CAS Monograph 2, Chapters 4 to 6

Examples from the CAS published monograph Bahnemann (2015), Distributions for actuaries, which is a text for CAS Part 8. It is available for free on the CAS Website.

3.1. Simple Discrete Aggregate, Example 4.1

Assume that n = 0, 1, 2 are the only possible numbers of claims and they occur with probabilities 0.6, 0.3 and 0.1, and that there exist just three potential claim sizes: 1, 2, and 3 with probabilities 0.4, 0.5 and 0.1. (Note: the text uses claim sizes 100, 200 and 300.) Compute the distribution of possible outcomes and its mean and variance.

Imports and convenience functions.

```
from aggregate import qd, mv
import matplotlib.pyplot as plt
```

Build the aggregate and display key statistics.

```
a = build('agg Bahn.4.1 dfreq[0 1 2][.6 .3 .1] '
          'dsev[1 2 3][.4 .5 .1]')
qd(a)
mv(a)
```

```
Aggregate object: Bahn.4.1. Frequency distribution empirical. Severity
dhistogram, [1, 3], bounded; atoms [1 2 3]. Updated with bucket size 1
and log2 = 3. Validation: not unreasonable.

      Mean      SD      CV    Skew P01 Median P99
X
Freq   0.5 0.67082  1.3416 0.99381
Sev    1.7 0.64031 0.37665 0.36568   1      2   3
Agg   0.85   1.227  1.4435  1.3333   0      0   5
mean     = 0.85
variance = 1.5055
std dev  = 1.22699
```

Display all possible outcomes. Compare with the table on p. 107.

```
qd(a.density_df.query('p_total > 0') [['p_total', 'F']])
```

```
      p_total     F
loss
0.0       0.6   0.6
1.0      0.12  0.72
2.0     0.166 0.886
3.0      0.07 0.956
4.0     0.033 0.989
5.0      0.01 0.999
6.0     0.001     1
```

3.2. Poisson-Gamma (Tweedie) Aggregate, Example 4.2

The text considers a Tweedie with expected claim count $\lambda = 2.5$ and gammma shape 3 and scale 400. It computes the mean, variance and skewness, and uses the series expansion for the distribution to compute the CDF at various points (Table 4.1). These results can be replicated as follows.

```
a = build('agg Bahn.4.2 2.5 claims '
          'sev 400 * gamma 3 poisson')
qd(a)
mv(a)
```

```
Aggregate object: Bahn.4.2. Frequency distribution poisson. Severity
gamma, [0, inf), exponential right tail. Updated with bucket size 1 and
log2 = 16. Validation: not unreasonable.

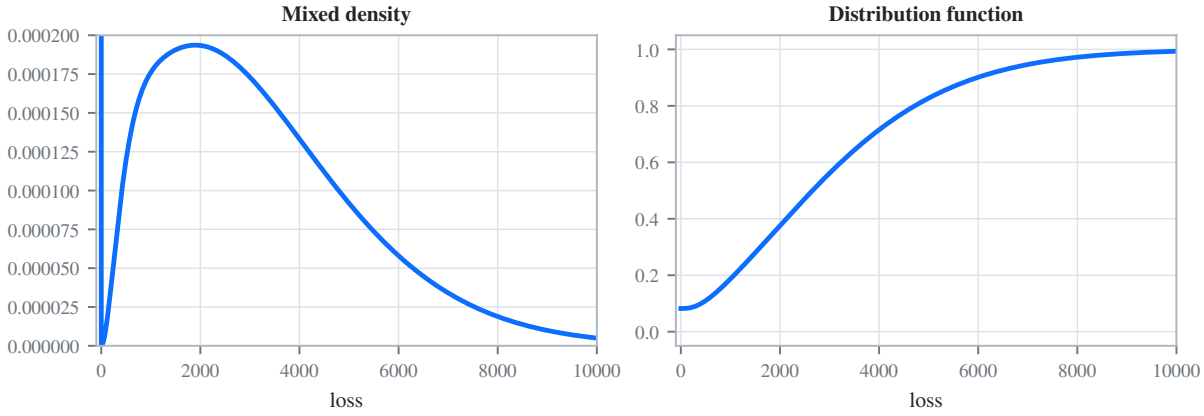
      Mean     SD      CV    Skew P01 Median  P99
X
Freq   2.5 1.5811 0.63246 0.63246
Sev   1200 692.82 0.57735  1.1547 174   1070 3362
Agg   3000 2190.9  0.7303 0.91287   0   2657 9435
mean     = 3000
variance = 4800000
std dev  = 2190.89
```

Extract various points of the pmf, cdf, and sf. The adjustment to the index is cosmetic. `aggregate` returns the entire distribution. The left plot shows the mixed density, with a mass at zero; right shows the cdf.

```
bit = a.density_df.loc[
    sorted(np.hstack((500, np.arange(0, 10000.5, 1000)))),
    ['p', 'F', 'S']]
qd(bit, accuracy=4)
fig, axs = plt.subplots(1, 2, figsize=(3.5*2, 2.45), constrained_layout=True, squeeze=True)
ax0, ax1 = axs.flat
(a.density_df.p / a.bs).plot(ylim=[0, 0.0002], xlim=[-100, 10000], lw=2, ax=ax0)
ax0.set(title='Density')
a.density_df.F.plot(ylim=[-0.05, 1.05], xlim=[-100, 10000], lw=2, ax=ax1)
ax0.set(title='Mixed density');
ax1.set(title='Distribution function');
```

```
                p        F         S
loss
0.0       0.082085 0.082085   0.91792
500.0   0.00011953  0.10961   0.89039
1000.0  0.00017612  0.18681   0.81319
2000.0  0.00019345  0.37563   0.62437
```

```
3000.0  0.00017303  0.56136   0.43864
4000.0  0.00013322  0.71522   0.28478
5000.0  9.1789e-05  0.82733   0.17267
6000.0  5.7942e-05  0.90137  0.098632
7000.0  3.4045e-05  0.94654  0.053459
8000.0   1.883e-05  0.97234  0.027657
9000.0  9.8863e-06  0.98628  0.013725
10000.0 4.9597e-06  0.99344 0.0065598
```



3.3. Approximations to the Tweedie, Example 4.3

`aggregate` largely circumvents the need for approximations, but it does support their creation. The following reproduces Table 4.3.

```
fz = a.approximate('all')
bit['Normal'] = fz['norm'].cdf(bit.index)
bit['Norm err'] = bit.Normal / bit.F - 1
bit['sGamma'] = fz['sgamma'].cdf(bit.index)
bit['sGamma err'] = bit.sGamma / bit.F - 1
qd(bit, accuracy=4)
```

```
                p        F         S   Normal  Norm err   sGamma  sGamma err
loss
0.0       0.082085 0.082085   0.91792 0.085452  0.041016 0.045932    -0.44044
500.0   0.00011953  0.10961   0.89039  0.12692   0.15789  0.10104   -0.078199
1000.0  0.00017612  0.18681   0.81319  0.18066 -0.032961  0.17751   -0.049818
2000.0  0.00019345  0.37563   0.62437  0.32404  -0.13734  0.36803   -0.020213
3000.0  0.00017303  0.56136   0.43864      0.5  -0.10931  0.56073  -0.0011223
4000.0  0.00013322  0.71522   0.28478  0.67596 -0.054892  0.71854   0.0046408
5000.0  9.1789e-05  0.82733   0.17267  0.81934 -0.009651  0.83105    0.004497
6000.0  5.7942e-05  0.90137  0.098632  0.91455  0.014623  0.90379   0.0026876
7000.0  3.4045e-05  0.94654  0.053459  0.96606  0.020617  0.94752   0.0010376
8000.0   1.883e-05  0.97234  0.027657  0.98876  0.016885  0.97238  3.8292e-05
9000.0  9.8863e-06  0.98628  0.013725  0.99692  0.010788  0.98589 -0.00038837
10000.0 4.9597e-06  0.99344 0.0065598   0.9993 0.0058994  0.99298 -0.00046794
```

Here is Table 4.4. The FFT overstates $F(0)$ because of discretization error.

```
a2 = build('agg Bahn.4.2b 10 claims '
           'sev 6000 * gamma 0.05 poisson')
qd(a2)
fz = a2.approximate('all')
```

```
bit = a2.density_df.loc[
    sorted(np.hstack((500, np.arange(0, 20000, 2000)))),
    ['p', 'F', 'S']]
bit['Normal'] = fz['norm'].cdf(bit.index)
bit['Norm err'] = bit.Normal / bit.F - 1
bit['sGamma'] = fz['sgamma'].cdf(bit.index)
bit['sGamma err'] = bit.sGamma / bit.F - 1
qd(bit, accuracy=4)
```

```
Aggregate object: Bahn.4.2b. Frequency distribution poisson. Severity
gamma, [0, inf), exponential right tail. Updated with bucket size 10 and
log2 = 16. Validation: fails sev mean, agg mean.

       Mean     SD      CV    Skew P01 Median   P99
X
Freq     10 3.1623 0.31623 0.31623
Sev  299.88 1341.7   4.474   8.944   0      0  6530
Agg  2998.8 4347.4  1.4497  2.8293   0   1290 20340

                 p        F        S  Normal  Norm err  sGamma  sGamma err
loss
0.0       0.061167 0.061167  0.93883 0.24516     3.008 0.12372      1.0227
500.0    0.0028183  0.34051  0.65949 0.28272  -0.16972 0.33463   -0.017272
2000.0   0.0011075  0.59287  0.40713 0.40914   -0.3099 0.58876  -0.0069326
4000.0  0.00057291  0.75165  0.24835 0.59107  -0.21364 0.75046   -0.001581
6000.0  0.00034106  0.84035  0.15965 0.75501  -0.10156 0.84024 -0.00013316
8000.0  0.00021479  0.89473  0.10527 0.87501 -0.022042 0.89494  0.00024274
10000.0 0.00013943   0.9295 0.070498 0.94635  0.018122 0.92977  0.00028566
12000.0  9.225e-05  0.95229 0.047706  0.9808  0.029929 0.95251  0.00023149
14000.0 6.1831e-05  0.96747 0.032529 0.99431  0.027737 0.96763  0.00016081
16000.0 4.1834e-05  0.97769  0.02231 0.99861  0.021395 0.97779  9.9642e-05
18000.0 2.8507e-05  0.98463 0.015371 0.99972  0.015327 0.98468  5.4005e-05
```

3.4. Poisson-Discrete Distribution, Example 4.4

The claim-count random variable is Poisson distributed with mean 1.75. Severity has a discrete distribution with outcomes 1, 2, 3, 4, 5 occurring with probabilities 0.2, 0.4, 0.2, 0.15, 0.05 respectively. Compute the aggregate distribution.

Here is Table 4.5.

```
a = build('agg Bahn.4.4 1.75 claims '
          'dsev [1 2 3 4 5] [.2 .4 .2 .15 .05] '
          'poisson')
qd(a)
qd(a.density_df.query('p > .001')[['p', 'F', 'S']], accuracy=4)
```

```
Aggregate object: Bahn.4.4. Frequency distribution poisson. Severity
dhistogram, [1, 5], bounded; atoms [1 2 3 4 5]. Updated with bucket size
1 and log2 = 6. Validation: not unreasonable.

       Mean     SD      CV    Skew P01 Median P99
X
Freq   1.75 1.3229 0.75593 0.75593
Sev    2.45 1.1169 0.45588 0.55603   1      2   5
Agg  4.2875  3.562 0.83078 0.95453   0      4  15

             p       F         S
loss
```

```
0.0    0.17377 0.17377   0.82623
1.0   0.060821 0.23459   0.76541
2.0    0.13229 0.36688   0.63312
3.0    0.10464 0.47152   0.52848
4.0    0.11704 0.58855   0.41145
5.0   0.093249  0.6818    0.3182
6.0   0.078644 0.76045   0.23955
7.0   0.064101 0.82455   0.17545
8.0   0.049889 0.87444   0.12556
9.0   0.037655 0.91209  0.087908
10.0   0.02737 0.93946  0.060537
11.0  0.019672 0.95913  0.040865
12.0  0.013764  0.9729  0.027101
13.0 0.0094201 0.98232  0.017681
14.0 0.0063168 0.98864  0.011364
15.0 0.0041654  0.9928 0.0071986
16.0 0.0027032  0.9955 0.0044954
17.0 0.0017249 0.99723 0.0027705
18.0 0.0010842 0.99831 0.0016863
```

3.5. Poisson-Gamma Distribution, Example 4.5

Aggregate losses have Poisson frequency with mean 2.5 and gamma severity with shape 3 and scale 400. Hence the aggregate mean equals 1,200 and variance equals 480,000. Now approximate the distribution function using FFT with a fine bucket size and the midpoint method for assigning claim-size probabilities and then `bs=20` and `bs=100`.

Here is Table 4.6, comparing the distributions. The `update` method re-runs the FFT computation with different options, here altering `bs`.

```
import numpy as np
import pandas as pd
a = build('agg Bahn.4.5 2.5 claims '
          'sev 400 * gamma 3 poisson')
qd(a)
xs = sorted(np.hstack((500, np.arange(0, 10001, 1000))))
bit = a.density_df.loc[xs, ['F']]
a.update(bs=100)
bit100 = a.density_df.loc[xs,  ['F']]
a.update(bs=20)
bit20 = a.density_df.loc[xs,  ['F']]
bit = pd.concat((bit, bit100, bit20), axis=1, keys=['h0.25', 'h100', 'h20'])
bit[('h100', 'Rel Error')] = bit[('h100', 'F')] / bit[('h0.25', 'F')] - 1
bit[('h20', 'Rel Error')] = bit[('h20', 'F')] / bit[('h0.25', 'F')] - 1
bit = bit.sort_index(axis=1)
qd(bit, accuracy=4)
```

```
Aggregate object: Bahn.4.5. Frequency distribution poisson. Severity
gamma, [0, inf), exponential right tail. Updated with bucket size 1 and
log2 = 16. Validation: not unreasonable.

      Mean     SD      CV    Skew P01 Median  P99
X
Freq   2.5 1.5811 0.63246 0.63246
Sev   1200 692.82 0.57735  1.1547 174   1070 3362
Agg   3000 2190.9  0.7303 0.91287   0   2657 9435

           h0.25     h100                 h20
               F        F  Rel Error        F  Rel Error
```

```
loss
0.0     0.082085 0.082146 0.00074147 0.082086 6.3888e-06
500.0    0.10961  0.11579   0.056419  0.11076   0.010459
1000.0   0.18681  0.19565   0.047279  0.18849  0.0089807
2000.0   0.37563  0.38525   0.025617  0.37747  0.0048979
3000.0   0.56136  0.56991   0.015216  0.56301  0.0029267
4000.0   0.71522  0.72175  0.0091319  0.71648  0.0017661
5000.0   0.82733   0.8318  0.0054039   0.8282  0.0010505
6000.0   0.90137  0.90417  0.0031115  0.90192 0.00060788
7000.0   0.94654  0.94818  0.0017304  0.94686 0.00033969
8000.0   0.97234  0.97324 0.00092617  0.97252 0.00018266
9000.0   0.98628  0.98675 0.00047663  0.98637 9.4437e-05
10000.0  0.99344  0.99367 0.00023603  0.99349 4.6979e-05
```

3.6. Poisson-Lognormal Distribution With Limit, Example 4.15

Consider an aggregate distribution with mean 3 Poisson frequency and lognormal claim size with parameters $(\mu, \sigma) = (6, 1.5)$. Moreover, claim size is limited by a policy limit of 1,000. Graph the aggregate distribution.

The log density (left) shows the probability masses at outcomes consisting of only limit losses. The distribution (right) shows the corresponding jumps. Compare with Figure 4.4.

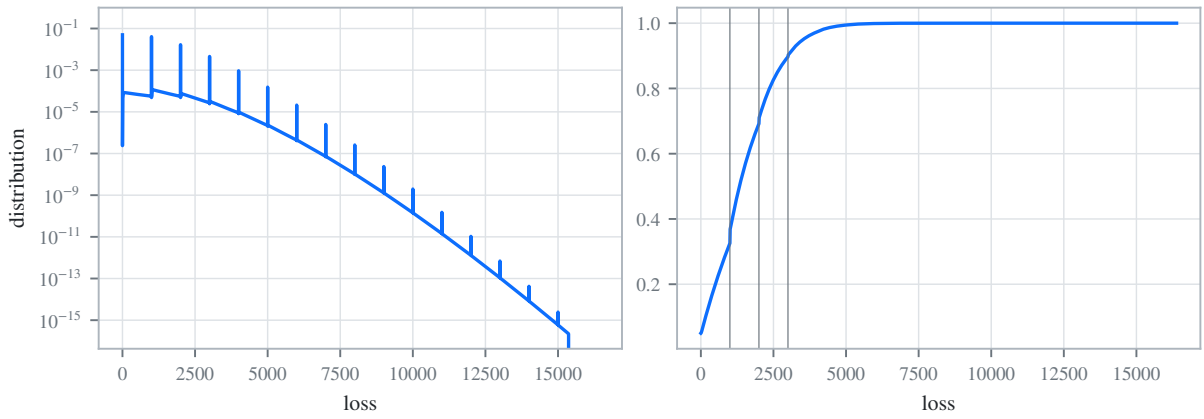
```
a = build('agg Bahn.4.15 '
          '3 claims '
          '1000 xs 0 '
          'sev exp(6) * lognorm 1.5 '
          'poisson')
qd(a)

fig, axs = plt.subplots(1, 2, figsize=(2*3.5, 2.45), constrained_layout=True)
ax0, ax1 = axs.flat
a.density_df.p.plot(ax=ax0, logy=True, label='FFT');
a.density_df.F.plot(ax=ax1, label='FFT');
ax0.set(ylabel='log density');
ax0.set(ylabel='distribution', ylim=[0,1]);
ax1.axvline(1000, c='C7', lw=.5);
ax1.axvline(2000, c='C7', lw=.5);
ax1.axvline(3000, c='C7', lw=.5);
```

```
Aggregate object: Bahn.4.15. Frequency distribution poisson. Severity
lognorm, [0, 1,000], bounded; 1,000 xs 0. Updated with bucket size 1/4
and log2 = 16. Validation: not unreasonable.

       Mean     SD      CV    Skew   P01 Median    P99
X
Freq      3 1.7321 0.57735 0.57735
Sev  502.97 375.98 0.74753 0.23542 12.25  403.5   1000
Agg  1508.9 1087.7 0.72083 0.82314     0 1327.8 4626.8

C:\Users\steve\AppData\Local\Temp\ipykernel_32032\2899995292.py:13: UserWarning: Attempt to set
non-positive ylim on a log-scaled axis will be ignored.
  ax0.set(ylabel='distribution', ylim=[0,1]);
```



3.7. Poisson-Gamma Distribution and Approximations, Problems 4.7 and 13

An aggregate distribution has mean 8 Poisson frequency and gamma severity with shape 0.2 and scale 3750. Compute the distribution and compare with normal and shifted-gamma approximations.

```
a = build('agg Bahn.4.7 '
      '8 claims '
      'sev 3750 * gamma 0.2 '
      'poisson')
qd(a)
xs = np.arange(0, 30000,3000)
qd(a.density_df.loc[xs, ['p', 'F','S']], accuracy=4)
```

```
Aggregate object: Bahn.4.7. 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      8 2.8284 0.35355 0.35355
Sev  749.95 1677.1  2.2362   4.472   0     76  8260
Agg  5999.6 5196.2 0.86608  1.5877  48   4648 23552

                 p         F         S
loss
0.0      0.0023109 0.0023109   0.99769
3000.0  0.00041852   0.34222   0.65778
6000.0  0.00028642   0.60713   0.39287
9000.0  0.00017445   0.77755   0.22245
12000.0 0.00010003   0.87826   0.12174
15000.0 5.5233e-05   0.93495  0.065048
18000.0 2.9705e-05   0.96587  0.034134
21000.0 1.5667e-05   0.98234  0.017661
24000.0 8.1387e-06   0.99097 0.0090339
27000.0 4.1768e-06   0.99542 0.0045775
```

`aggregate` readily computes approximations and returns frozen `scipy.stats` objects.

```
fz = a.approximate('all')
comp = pd.DataFrame({k: v.cdf(xs) for k, v in fz.items()}, index=xs)
comp['agg'] = a.density_df.loc[xs, 'F',]
comp.loc[:, [f'{k} err' for k in fz.keys()]] = comp.loc[:, fz.keys()].values / comp.loc[:,
['agg']].values - 1
```

```
comp = comp.sort_index(axis=1)
qd(comp, accuracy=4)
```

```
           agg   gamma   gamma err  lognorm  lognorm err    norm  norm err   sgamma  sgamma err  \
0     0.0023109       0          -1        0           -1 0.12412    52.712 0.026323      10.391
3000    0.34222  0.3398  -0.0070686  0.29034     -0.15158 0.28188  -0.17632  0.33627   -0.017387
6000    0.60713 0.61488    0.012756  0.64585     0.063778 0.50003   -0.1764  0.60544  -0.0027872
9000    0.77755 0.78364   0.0078297   0.8202     0.054849 0.71817 -0.076365  0.77825  0.00090158
12000   0.87826 0.88085   0.0029516  0.90331      0.02853 0.87591 -0.002674  0.87926   0.0011442
15000   0.93495  0.9352   0.0002659  0.94508     0.010837 0.95837  0.025052   0.9356  0.00068859
18000   0.96587 0.96506 -0.00083072  0.96731    0.0014959 0.98954  0.024513  0.96614  0.00028188
21000   0.98234 0.98128  -0.0010749  0.97975   -0.0026307 0.99805  0.015998  0.98239   4.773e-05
24000   0.99097 0.99002 -0.00095325  0.98703   -0.0039689 0.99973 0.0088479  0.99091 -5.3322e-05
27000   0.99542  0.9947 -0.00072519  0.99145   -0.0039887 0.99997 0.0045718  0.99534 -7.9709e-05

       slognorm  slognorm err
0      0.058639        24.375
3000    0.31391     -0.082718
6000    0.59174     -0.025344
9000    0.77907     0.0019498
12000   0.88466     0.0072901
15000   0.94022      0.005635
18000   0.96881     0.0030474
21000    0.9835     0.0011869
24000   0.99113     0.0001663
27000   0.99515   -0.00027821
```

3.8. Poisson-Lognormal Layer Statistics, Example 5.13

Consider an aggregate distribution with mean 15 Poisson frequency and lognormal claim size with parameters $(\mu, \sigma) = (5.9809, 1.8)$. What are the distribution characteristics for random variable S for claims in the layer 5,000 excess of 3,000?

The exact and FFT-estimated mean, cv, and skewness are reported in the `summary_df` dataframe, for frequency and severity. The values reported agree with the text, up to rounding.

```
a = build('agg Bahn.5.13 '
      '15 claims 5000 xs 3000 '
      'sev exp(5.9809) * lognorm 1.8 ! '
      'poisson')
qd(a)
mv(a)
```

```
Aggregate object: Bahn.5.13. Frequency distribution poisson. Severity
lognorm, [0, 5,000], bounded; 5,000 xs 3,000. Updated with bucket size 2
and log2 = 16. Validation: not unreasonable.

       Mean     SD      CV    Skew P01 Median   P99
X
Freq     15  3.873  0.2582  0.2582
Sev  385.68 1207.9  3.1319  3.1942   0      0  5000
Agg  5785.3 4910.9 0.84887 0.93405   0   5000 20028
mean     = 5785.25
variance = 2.411727e+07
std dev  = 4910.93
```

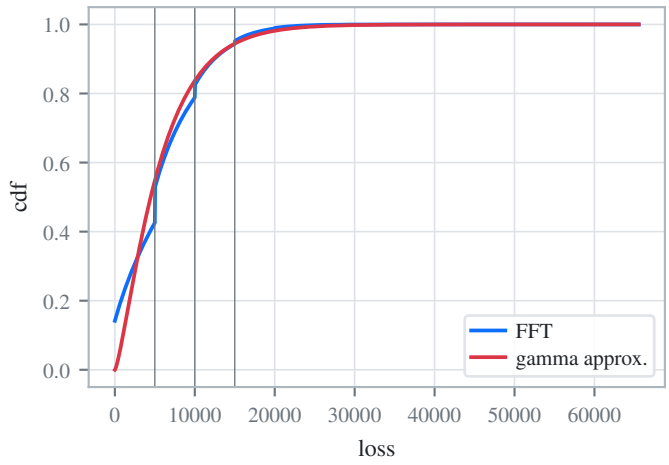
The exact severity can be accessed directly, as `a.sevs[0].fz`, allowing us to compute the expected layer claim count. The aggregate can then be written in conditional form, producing the same statistics. The distribution function shows probability masses at multiples of the limit.

```
xs = 15 * a.sevs[0].fz.sf(3000)
print(f'excess claim count = {xs:.5f}')

a = build('agg Bahn.5.13b '
          f'{xs} claims 5000 xs 3000 '
          'sev exp(5.9809) * lognorm 1.8 '
          'poisson')
qd(a)
fig, ax = plt.subplots(1,1,figsize=(3.5, 2.45))
a.density_df.F.plot(ax=ax, label='FFT');
fz = a.approximate('gamma')
ax.plot(a.density_df.loss, fz.cdf(a.density_df.loss), c='C1', label='gamma approx.');
ax.axvline(5000,  c='C7', lw=.5);
ax.axvline(10000, c='C7', lw=.5);
ax.axvline(15000, c='C7', lw=.5);
ax.set(ylabel='cdf');
ax.legend(loc='lower right');
```

```
excess claim count = 1.95359
Aggregate object: Bahn.5.13b. Frequency distribution poisson. Severity
lognorm, [0, 5,000], bounded; 5,000 xs 3,000. Updated with bucket size 1
and log2 = 16. Validation: not unreasonable.

       Mean     SD      CV     Skew P01 Median   P99
X
Freq 1.9536 1.3977 0.71546  0.71546
Sev  2961.3 1890.9 0.63853 -0.16374  34   3030  5000
Agg  5785.3 4910.9 0.84887  0.93405   0   5000 20029
```



3.9. Lognormal Increased Limits Factors (ILFs), Example 6.3

Indemnity losses for a portfolio of insurance policies have a lognormal claim-size distribution with parameters $(\mu, \sigma) = (7, 2.4)$. The policy per-claim limit applies only to the indemnity portion of a claim, and the average per-claim loss adjustment expense is 2,200. Claim frequency for these policies is 0.0005 per exposure unit, and variable expenses equal 35% of premium.

A lognormal with $\sigma = 2.4$ has cv $\sqrt{\exp(2.4^2) - 1} = 17.78$ and is extremely thick-tailed, despite having moments of all orders. It is challenging to approximate numerically. Luckily, we only need to compute up to 5M. The `aggregate` parameters deliberately select a range that is too narrow for the entire distribution, but adequate for our purposes. Use `log2=17` and select `bs` greater than `5e6 // 2**17 = 38`. We use `bs=50`. It is important to set `normalize=False` to avoid rescaling bucket probabilities to sum to one. These parameters are not a good model for the entire distribution; the mean error is too high.

The `density_df` dataframe includes limited expected values. Here is a sample.

```
a = build('agg Bahn.6.3 '
          '1 claim '
          'sev exp(7) * lognorm 2.4 '
          'fixed',
          bs=50, log2=17,
          normalize=False,
         )
qd(a)
xs = [1e5,  5e5, 7.5e5, 1e6, 2e6, 3e6, 4e6, 5e6]
qd(a.density_df.loc[xs, ['F', 'S', 'lev']], accuracy=4)
```

```
Aggregate object: Bahn.6.3. Frequency distribution fixed. Severity
lognorm, [0, inf), subexponential right tail. Updated with bucket size
50 and log2 = 17. Validation: grid infeasible for this severity; fails
pmf deficit 1.455e-04, sev mean, agg mean.

      Mean         SD     CV   Skew P01 Median        P99
X
Freq     1          0      0
Sev  18328 1.4284e+05 7.7935 27.885   0   1100 2.9165e+05
Agg  18328 1.4284e+05 7.7935 27.885   0   1100 2.9165e+05

                F          S    lev
loss
100000.0  0.96998   0.030021 8895.8
500000.0  0.99463  0.0053706  13625
750000.0  0.99674  0.0032647  14668
1000000.0 0.99774   0.002257  15345
2000000.0 0.99912 0.00087817  16738
3000000.0 0.99951 0.00048765  17390
4000000.0 0.99968 0.00031609  17782
5000000.0 0.99978 0.00022372  18048

V:\worktrees\aggregate_REFACTOR\src\aggregate\_bucket_window.py:1430: InfeasibleGridWarning:
Bahn.6.3: severity 'lognorm' cannot be reproduced on this grid. The body needs bs <= 61.2967
(0.003512% of the mean is supplied below the first half bucket, and is placed at 0); the mean is
not complete until the grid reaches 3,951,734,870, so matching it to 0.0001 needs log2 = 26, and
log2 = 17 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)
```

The following reproduces Table 6.1. The ILF factors assume fixed (middle) and variable ALAE (right).

```
alae = 2200
bit = a.density_df.loc[xs, ['lev']]
bit['Fixed ALAE'] = (bit.lev + alae) / (bit.lev.iloc[0] + alae)
bit['Prop ALAE'] = bit.lev / bit.lev.iloc[0]
qd(bit, accuracy=4)
```

loss	lev	Fixed ALAE	Prop ALAE
100000.0	8895.8	1	1
500000.0	13625	1.4262	1.5317
750000.0	14668	1.5202	1.6488
1000000.0	15345	1.5812	1.725
2000000.0	16738	1.7067	1.8815
3000000.0	17390	1.7655	1.9549
4000000.0	17782	1.8009	1.9989
5000000.0	18048	1.8248	2.0288

3.10. Layer Premium, Example 6.4

(Continues Example 6.3.) Calculate the premium for successive excess layers of insurance for a policy with exposure equal 400. Use the ILFs under the assumption that the average per-claim ALAE payment is 2,200. Premium amounts for the successive million-dollar layers obtained from these layer factors applied to the basic-limit premium are displayed in Table 6.2 and reproduced below.

```
exposure = 400
var_exp = 0.35
frequency = 0.0005
bit['Premium'] = exposure * frequency * (bit['lev'] + alae) / (1 - var_exp)
bit['Layer Premium'] = np.diff(bit.Premium, prepend=0)
qd(bit)
```

loss	lev	Fixed ALAE	Prop ALAE	Premium	Layer Premium
100000.0	8895.8	1	1	3414.1	3414.1
500000.0	13625	1.4262	1.5317	4869.3	1455.2
750000.0	14668	1.5202	1.6488	5190.1	320.73
1000000.0	15345	1.5812	1.725	5398.4	208.38
2000000.0	16738	1.7067	1.8815	5827	428.52
3000000.0	17390	1.7655	1.9549	6027.7	200.71
4000000.0	17782	1.8009	1.9989	6148.3	120.63
5000000.0	18048	1.8248	2.0288	6230.1	81.76

3.11. Risk Loads, Example 6.5

Example 6.5, computes risk loads as a percentage of standard deviation. `aggregate` can compute multiple limits at once, and the `report_df` dataframe returns individual severity and aggregate distribution statistics. The risk loads can be deduced from these. The risk load can be computed as `k' * ex2` or `k * actual_cv` (not shown).

The following code reproduces Table 6.3. First, define the controlling variables, and then set up the tower of limits within one object, using Vectorization: Limit Profiles and Mixed Severity.

```
k_prime = 0.0277
m = 400
ϕ = 0.0005
u = 0.2
k = k_prime / np.sqrt(m * ϕ)

limits = [1e5, 5e5, 1e6, 2e6, 3e6, 4e6, 5e6]
```

```
bl = build('agg Bahn.6.5 '
           f'{m * ϕ} claims '
           f'{limits} xs 0 '
           'sev exp(7) * lognorm 2.4 '
           'poisson'
           , bs=50, log2=18)
qd(bl.stats_df.iloc[:, :len(limits)], accuracy=4)
```

```
                          e0.m0      e1.m0      e2.m0      e3.m0      e4.m0      e5.m0      e6.m0
component measure
meta      limit            1e+05      5e+05      1e+06      2e+06      3e+06      4e+06      5e+06
          attachment           0          0          0          0          0          0          0
          el              1779.2     2725.1       3069     3347.6       3478     3556.5     3609.6
          prem                 0          0          0          0          0          0          0
          lr                   0          0          0          0          0          0          0
          sevcv_param          0          0          0          0          0          0          0
          mix_cv               0          0          0          0          0          0          0
          wt             0.14286    0.14286    0.14286    0.14286    0.14286    0.14286    0.14286
freq      ex1                0.2        0.2        0.2        0.2        0.2        0.2        0.2
          ex2               0.24       0.24       0.24       0.24       0.24       0.24       0.24
          ex3              0.328      0.328      0.328      0.328      0.328      0.328      0.328
          mean               0.2        0.2        0.2        0.2        0.2        0.2        0.2
...                          ...        ...        ...        ...        ...        ...        ...
sev       ex1               8896      13626      15345      16738      17390      17782      18048
          ex2        5.1251e+08 2.8271e+09 5.2833e+09 9.2441e+09 1.2442e+10 1.5158e+10 1.7534e+10
          ex3        4.2417e+13 1.0687e+15 3.8053e+15 1.2593e+16 2.4512e+16 3.8722e+16 5.4725e+16
          mean              8896      13626      15345      16738      17390      17782      18048
          cv              2.3401     3.7719       4.63     5.6565     6.3357      6.851     7.2685
          skew            3.3416     7.0585     9.9524     14.303     17.849     20.975     23.827
agg       ex1               1779.2     2725.1       3069     3347.6       3478     3556.5     3609.6
          ex2        1.0567e+08 5.7284e+08 1.0661e+09   1.86e+09 2.5005e+09 3.0442e+09 3.5199e+09
          ex3        9.0362e+12 2.1839e+14 7.7081e+14 2.5373e+15 4.9284e+15 7.7768e+15 1.0983e+16
          mean              1779.2     2725.1       3069     3347.6       3478     3556.5     3609.6
          cv                5.6903     8.7257     10.592     12.844     14.342     15.482     16.406
          skew              8.1747     15.898     22.157     31.683     39.494     46.396     52.704
```

`stats_df` carries one column per vectorized limit followed by the whole-object summary columns (`mixed`, `independent`, `after_occ`, `empirical` and the impact and error columns). Slicing to `len(limits)` keeps the per-limit components and drops the summary, whatever the summary block happens to contain.

Next, extract the required columns from `stats_df` and manipulate to compute the ILFs.

```
bit = bl.stats_df.loc[
    [('sev', 'mean'), ('sev', 'cv'), ('agg', 'mean'), ('agg', 'cv')]
].iloc[:, :len(limits)].T
bit.columns = ['sev_m', 'sev_cv', 'actual_m', 'actual_cv']
bit.index = limits
bit.index.name = 'limit'
bit['vx'] = (bit.sev_m * bit.sev_cv) ** 2
bit['ex2'] = bit.vx + bit.sev_m**2
bit['risk load'] = k_prime * bit.ex2 ** 0.5
bit['lev'] = (1+u) * bit.sev_m
bit['ILF w/o risk'] = bit['lev'] / bit.loc[100000, 'lev']
bit['ILF with risk'] = (bit['lev'] + bit['risk load']) / (bit.loc[100000, 'lev'] +
bit.loc[100000, 'risk load'])
qd(bit, accuracy=4)
```

```
           sev_m  sev_cv  actual_m  actual_cv         vx         ex2  risk load   lev  \
```

```
limit
100000.0    8896  2.3401    1779.2     5.6903 4.3337e+08 5.1251e+08     627.09 10675
500000.0   13626  3.7719    2725.1     8.7257 2.6414e+09 2.8271e+09     1472.8 16351
1000000.0  15345    4.63      3069     10.592 5.0478e+09 5.2833e+09     2013.4 18414
2000000.0  16738  5.6565    3347.6     12.844  8.964e+09 9.2441e+09     2663.3 20085
3000000.0  17390  6.3357      3478     14.342 1.2139e+10 1.2442e+10     3089.7 20868
4000000.0  17782   6.851    3556.5     15.482 1.4842e+10 1.5158e+10     3410.4 21339
5000000.0  18048  7.2685    3609.6     16.406 1.7208e+10 1.7534e+10     3667.9 21658

           ILF w/o risk  ILF with risk
limit
100000.0              1              1
500000.0         1.5316          1.577
1000000.0        1.7249         1.8074
2000000.0        1.8815         2.0127
3000000.0        1.9548         2.1197
4000000.0        1.9989         2.1897
5000000.0        2.0288         2.2407
```

3.12. Aggregate Premiums, Example 6.6

(Continues Example 6.3.) Compute expected losses across a variety of occurrence and aggregate limit combinations. Assume 20% ALAE outside the limits, expected claim count 1.2 with contagion parameter 0.1 (cv of mixing $\sqrt{0.1}$), and lognormal severity $(\mu, \sigma) = (7.6, 2.4)$ (see errata).

The following code calculates Table 6.4 using FFT aggregate distributions. The last column, showing unlimited aggregate losses, agrees, but the other columns are slightly different because Bahnemann uses a shifted gamma approximation.

First, we compute all the aggregates.

```
b = {}
for per_claim in [0.5e6, 1e6, 2e6, 3e6, 4e6, 5e6]:
    tower = np.array([0]  + [i for i in [0, 1e6, 2e6, 3e6, 4e6, 5e6, np.inf]
            if i >= per_claim])
    b[per_claim] = build('agg Bahn.6.6 1.2 claims '
           f'{per_claim} xs 0 '
           'sev exp(7.6) * lognorm 2.4 '
           f'mixed gamma {0.1}**.5 '
           f'aggregate ceded to tower {tower} '
           , bs=50, log2=18,
           normalize=False,
          )
qd(pd.concat([i.summary_df[['Mean', 'CV', 'Skew']]
              for i in b.values()],
    keys=b.keys(), names=['Occ limit', 'X']),
    accuracy=4)
```

```
                Mean      CV   Skew
Occ limit X
500000.0  Freq   1.2 0.96609 1.0696
          Sev  21742  3.1549 5.3107
          Agg  26091  3.0377 4.9943
1000000.0 Freq   1.2 0.96609 1.0696
          Sev  25279  3.8779 7.3846
          Agg  30335  3.6694 6.7822
2000000.0 Freq   1.2 0.96609 1.0696
          Sev  28338   4.758 10.437
```

```
          Agg  34006  4.4496 9.4728
3000000.0 Freq   1.2 0.96609 1.0696
          Sev  29850  5.3511 12.892
          Agg  35820  4.9795  11.66
4000000.0 Freq   1.2 0.96609 1.0696
          Sev  30791  5.8075  15.04
          Agg  36949  5.3888 13.582
5000000.0 Freq   1.2 0.96609 1.0696
          Sev  31444  6.1815  16.99
          Agg  37733   5.725 15.332
```

The tower cedes everything above zero, so the ceded view `summary_df` reports here is also the subject (gross) view. Where a cession is partial, read the subject statistics off the `('agg', 'subject', 'agg')` row of `reins_summary_df` instead.

Next, determine the layer loss costs. The cumulative ceded loss to an aggregate limit `L` is the limited expected value $\mathsf{P}(S \wedge L)$, which equals the whole-structure ceded mean of an `aggregate ceded to L xs 0` cover. Read it from `reins_summary_df` (the `('agg', 'ceded', 'agg')` row, column `Est EX`).

> **Note**
>
> This example previously used the per-layer `reinsurance_audit_df` dataframe, removed in 1.0.0a19 in favour of the whole-structure `reins_summary_df` / `reins_stats_df` / `reins_density_df` objects. To rebuild the per-agg-limit ILF table, loop over the aggregate breakpoints `L` and the per-claim occurrence limits, build `aggregate ceded to L xs 0` for each, and collect the ceded aggregate mean from `reins_summary_df`; normalise by the `500000` per-claim, unlimited-aggregate cell for the ILF.

Here is a reconciliation to Table 6.4 of the 2M per claim and 2M aggregate limit expected loss, using the shifted gamma approximation. The limited aggregate loss is computed using the integral of the survival function `fz.sf`. `quad` is a general purpose numerical integration routine. It returns the integral and estimated error.

```
fz = b[2000000].approximate('sgamma')
print(fz.stats())
mv(b[2000000])
from scipy.integrate import quad
quad(fz.sf, 0, 2000000)
```

```
(np.float64(34005.90148666896), np.float64(22895161273.613277))
mean     = 34006.1
variance = 2.289516e+10
std dev  = 151311

(33523.06665834545, 0.00040650276002622316)
```

3.13. Deductible Credits, Example 6.7

(Continues Example 6.3.) Consider a portfolio of policies for which the ground-up indemnity claim size has a lognormal distribution with parameters $(\mu, \sigma) = (7.0, 2.4)$ and allocated loss adjustment expense is 20% of the indemnity amount. The basic limit is 100,000. Calculate the credit factors, as well as the resulting frequency and severity, for six straight deductible options: 1,000; 2,000; 3,000; 4,000; 5,000; and 10,000. Base frequency equals 0.0005.

We can build all of the required distributions simultaneously using vectorization. Remember that the basic limit is ground up. The severity is unconditional, indicated by `!` at the end of the severity clause. The limit is eroded by the deductible.

```
deductibles = [0, 1e3, 2e3, 3e3, 4e3, 5e3, 10e3]
limits = [100000 - i for i in deductibles]
ϕ = 0.0005
alae = 1.2
bl = build('agg Bahn.6.7 '
           f'{ϕ} claims '
           f'{limits} xs {deductibles} '
           'sev exp(7) * lognorm 2.4 ! '
           'poisson'
           , bs=50, log2=18)
qd(bl.stats_df.iloc[:, :len(limits)], accuracy=4)
```

		e0.m0	e1.m0	e2.m0	e3.m0	e4.m0	e5.m0	e6.m0
component	measure							
meta	limit	1e+05	99000	98000	97000	96000	95000	90000
	attachment	0	1000	2000	3000	4000	5000	10000
	el	4.448	4.1183	3.8926	3.7092	3.5517	3.4124	2.8758
	prem	0	0	0	0	0	0	0
	lr	0	0	0	0	0	0	0
	sevcv_param	0	0	0	0	0	0	0
	mix_cv	0	0	0	0	0	0	0
	wt	0.14286	0.14286	0.14286	0.14286	0.14286	0.14286	0.14286
freq	ex1	0.0005	0.0005	0.0005	0.0005	0.0005	0.0005	0.0005
	ex2	0.00050025	0.00050025	0.00050025	0.00050025	0.00050025	0.00050025	0.00050025
	ex3	0.00050075	0.00050075	0.00050075	0.00050075	0.00050075	0.00050075	0.00050075
	mean	0.0005	0.0005	0.0005	0.0005	0.0005	0.0005	0.0005
...		...	...	...	...	...	...	...
sev	ex1	8896	8236.6	7785.2	7418.4	7103.4	6824.9	5751.7
	ex2	5.1251e+08	4.9544e+08	4.7944e+08	4.6425e+08	4.4973e+08	4.3581e+08	3.7327e+08
	ex3	4.2417e+13	4.0906e+13	3.9443e+13	3.8028e+13	3.6657e+13	3.5329e+13	2.9274e+13
	mean	8896	8236.6	7785.2	7418.4	7103.4	6824.9	5751.7
	cv	2.3401	2.5106	2.6287	2.7269	2.813	2.8907	3.2067
	skew	3.3416	3.3681	3.4054	3.4444	3.4833	3.5215	3.6997
agg	ex1	4.448	4.1183	3.8926	3.7092	3.5517	3.4124	2.8758
	ex2	2.5627e+05	2.4774e+05	2.3973e+05	2.3214e+05	2.2488e+05	2.1792e+05	1.8664e+05
	ex3	2.1212e+10	2.0456e+10	1.9725e+10	1.9017e+10	1.8331e+10	1.7667e+10	1.4639e+10
	mean	4.448	4.1183	3.8926	3.7092	3.5517	3.4124	2.8758
	cv	113.81	120.85	125.78	129.89	133.51	136.79	150.22
	skew	163.49	165.89	168.03	170.02	171.89	173.66	181.54

Next, manipulate the `stats_df` dataframe to compute the required quantities. The final exhibit replicates Table 6.5.

```
bit = bl.stats_df.iloc[:, :len(limits)].loc[
    [('meta', 'attachment'), ('freq', 'mean'), ('sev', 'mean'), ('agg', 'mean')]
].T
bit.columns = ['attachment', 'freq_m', 'sev_m', 'actual_m']
bit = bit.rename(columns={'attachment': 'deductible'}).set_index('deductible')
bit['F(d)'] = np.array([bl.sevs[0].fz.cdf(i) for i in bit.index])
bit['freq_m'] = bit.loc[0, 'freq_m'] * (1 - bit['F(d)'])
bit['E[X;d]'] = (bit.sev_m[0] - bit.sev_m)
bit['C(d)'] = bit['E[X;d]'] / bit.sev_m[0]
bit['sev_m'] = bit['sev_m'] / (1 - bit['F(d)']) * alae
bit = bit.iloc[:, [-2, 3, -1, 0, 1]]
bit['pure prem'] = bit.freq_m * bit.sev_m
qd(bit, accuracy=4)
```

```
            E[X;d]    F(d)     C(d)     freq_m  sev_m  pure prem
deductible
0.0              0       0        0     0.0005  10675     5.3376
1000.0      659.42 0.48467 0.074125 0.00025766  19180      4.942
2000.0      1110.8 0.59885  0.12487 0.00020057  23289     4.6711
3000.0      1477.7 0.66251  0.16611 0.00016875  26377      4.451
4000.0      1792.7 0.70512  0.20151 0.00014744  28907      4.262
5000.0      2071.2 0.73636  0.23282 0.00013182  31064     4.0949
10000.0     3144.4 0.82147  0.35346 8.9266e-05  38660      3.451
```

3.14. Summary

Here is a summary of all the objects created in this section.

```
from aggregate import format_program
for n, r in build.discover('^Bahn').iterrows():
    print(format_program(r.program))
```

```
agg Bahn.4.1
  dfreq [0 1 2] [0.6 0.3 0.1]
  dsev [1 2 3] [0.4 0.5 0.1]
agg Bahn.4.15
  3 claims
  1000 xs 0
  sev 403.4287934927351 * lognorm 1.5
  poisson
agg Bahn.4.2
  2.5 claims
  sev 400 * gamma 3
  poisson
agg Bahn.4.2b
  10 claims
  sev 6000 * gamma 0.05
  poisson
agg Bahn.4.4
  1.75 claims
  dsev [1 2 3 4 5] [0.2 0.4 0.2 0.15 0.05]
  poisson
agg Bahn.4.5
  2.5 claims
  sev 400 * gamma 3
  poisson
agg Bahn.4.7
  8 claims
  sev 3750 * gamma 0.2
  poisson
agg Bahn.5.13
  15 claims
  5000 xs 3000
  sev 395.7964246880697 * lognorm 1.8 !
  poisson
agg Bahn.5.13b
  1.9535923221305609 claims
  5000 xs 3000
  sev 395.7964246880697 * lognorm 1.8
  poisson
agg Bahn.6.3
  1 claim
  sev 1096.6331584284585 * lognorm 2.4
```

```
  fixed
agg Bahn.6.5
  0.2 claims
  [100000 500000 1000000 2000000 3000000 4000000 5000000] xs 0
  sev 1096.6331584284585 * lognorm 2.4
  poisson
agg Bahn.6.6
  1.2 claims
  5000000 xs 0
  sev 1998.1958951041172 * lognorm 2.4
  mixed gamma 0.31622776601683794
  aggregate ceded to
    5000000 xs 0 and
    inf xs 5000000
agg Bahn.6.7
  0.0005 claims
  [100000 99000 98000 97000 96000 95000 90000] xs [0 1000 2000 3000 4000 5000 10000]
  sev 1096.6331584284585 * lognorm 2.4 !
  poisson
```

4. Brehm et al. on reinsurance in enterprise risk analysis

Brehm, Perry, Venter and Witcraft (2007), Enterprise Risk Analysis for Property and Liability Insurance Companies, Section 2.5

This section re-analyzes reinsurance structure alternatives introduced in Brehm et al. (2007), Enterprise Risk Analysis for Property & Liability Insurance Companies. This book is the ERM text on the syllabus for CAS Exam Part 7.

4.1. Reinsurance Example

This section analyzes the example given in Section 2.5 of Enterprise Risk Analysis.

Assumptions. ABCD writes 33M excess property and casualty business.

- ABCD total gross
  - Loss ratio: 69.36%
  - Expense ratio: 23%
  - Combined ratio: 92.36%
  - Margin 2.52M
- Casualty
  - 14M premium
  - 78% expected loss ratio
  - 5M limits
  - 4M xs 1M reinsurance, ceded premium 4.41M
- Property
  - 19M premium
  - 63% expected loss ratio
  - 20M limits
  - 17M xs 3M per risk reinsurance, ceded premium 2.36M
  - 95% share of 24M xs 1M cat reinsurance, ceded premium 1.53M with 1@100%
  - Cat program designed to cover to 250-year event.
- Reinsurance total
  - Average recoveries 5.08M
  - Ceded premium 8.3M
  - Net premium 24.7M

- Mythical alternative program
  - Stop-loss, 20 xs 30, ceded premium 1.98

Additional assumptions. There are no details of the stochastic model, so we assume

- Frequency and severity models per DecL below,
- Split the property losses into cat and non-cat by assuming that cat premium equals 2M, non-cat premium 17M, at the same loss ratios (this is just a split of losses, the by line loss ratios are not used), and
- Cat, non-cat and casualty are independent.
- Free and unlimited reinstatements on the catastrophe protection. See REF for a discussion of reinstatements.

4.2. Stochastic Models and Baseline Analysis

Construct the gross and net portfolios. All amounts in millions.

4.2.1. Gross Portfolio

The 250-year cat PML is printed last, to compare with the 25M program.

```
from aggregate import qd, mv
import pandas as pd
import matplotlib.pyplot as plt

abcd = build('port ABCD '
         'agg Casualty 14.0 premium at 78% lr '
             '5 xs 0 '
             'sev lognorm 0.1 cv 10 '
             'mixed gamma 0.3 '
         'agg PropertyNC 17.0 premium at 63% lr '
             '25 xs 0 '
             'sev lognorm [0.1 1] cv [5 10] wts [.7 .3] '
             'mixed gamma 0.1 '
         'agg PropertyC 2.0 premium at 63% lr '
             '150 xs 0 '
             'sev 3 * pareto 2.375 - 3 '
             'poisson ', bs=1/128)
qd(abcd)
mv(abcd)
print(abcd['PropertyC'].q(0.996))
```

```
Portfolio object: ABCD. Contains 3 aggregate components. Updated with
bucket size 1/128 and log2 = 16. Validation: fails sev mean, agg mean.

                    Mean      SD      CV    Skew      P01    Median    P99
unit       X
Casualty   Freq   125.93  39.409 0.31296 0.60054
           Sev  0.086436 0.34822  4.0286  9.5552        0 0.0078125 1.4766
           Agg    10.885  5.1839 0.47626 0.88565   2.4844    10.102 26.102
PropertyNC Freq   34.918  6.8638 0.19657 0.24744
           Sev   0.30659   1.506  4.9121  11.569        0   0.03125 5.4531
           Agg    10.706  9.1446 0.85419  1.9358   1.4531    7.5938 42.375
PropertyC  Freq   0.5801 0.76165  1.3129  1.3129
           Sev     2.172  4.3889  2.0207  11.511 0.015625    1.0156 17.859
```

```
           Agg      1.26  3.7298  2.9601  12.398        0         0 14.867
total      Agg     22.85  11.154 0.48813  1.6193   7.0234    20.375 58.766
mean     = 22.89
variance = 124.4768
std dev  = 11.1569
22.859375
```

4.2.2. Net Portfolio

```
abcd_net = build('port ABCD:Net '
         'agg Casualty 14.0 premium at 78% lr '
             '5 xs 0 '
             'sev lognorm 0.1 cv 10 '
             'occurrence net of 4 xs 1 '
             'mixed gamma 0.3 '
         'agg PropertyNC 17.0 premium at 63% lr '
             '25 xs 0 '
             'sev lognorm [0.1 1] cv [5 10] wts [.7 .3] '
             'occurrence net of 17 xs 3 '
             'mixed gamma 0.1 '
         'agg PropertyC 2.0 premium at 63% lr '
             '150 xs 0 '
             'sev 3 * pareto 2.375 - 3 '
             'occurrence net of 24 xs 1 '
             'poisson ', bs=1/128)
qd(abcd_net)
qd(abcd_net.est_sd)
```

```
Portfolio object: ABCD:Net. Contains 3 aggregate components. Updated
with bucket size 1/128 and log2 = 16. Validation: reinsurance; subject
fails sev mean, agg mean.
```

```
                    Mean      SD      CV    Skew      P01    Median    P99
unit       X
Casualty   Freq   125.93  39.409 0.31296 0.60054
           Sev  0.065564   0.166  2.5318  4.1715        0 0.0078125      1
           Agg    8.2563  3.1853  0.3858 0.65534   2.4375    7.9062 17.141
PropertyNC Freq   34.918  6.8638 0.19657 0.24744
           Sev    0.2098 0.60011  2.8604  6.1847        0   0.03125      3
           Agg    7.3259  3.8274 0.52244  1.0361   1.4531    6.6641 18.984
PropertyC  Freq   0.5801 0.76165  1.3129  1.3129
           Sev   0.80417   2.199  2.7345  36.228 0.015625         1      1
           Agg    0.4665  1.7834  3.8228  40.649        0         0    2.5
total      Agg    16.049  5.2892 0.32957  2.0938    6.625    15.484 30.305
5.2892
```

4.2.3. Ceded Portfolio

```
abcd_ceded = build('port ABCD:Ceded '
         'agg Casualty 14.0 premium at 78% lr '
             '5 xs 0 '
             'sev lognorm 0.1 cv 10 '
             'occurrence ceded to 4 xs 1 '
             'mixed gamma 0.3 '
         'agg PropertyNC 17.0 premium at 63% lr '
             '25 xs 0 '
```

```
            'sev lognorm [0.1 1] cv [5 10] wts [.7 .3] '
            'occurrence ceded to 17 xs 3 '
            'mixed gamma 0.1 '
        'agg PropertyC 2.0 premium at 63% lr '
            '150 xs 0 '
            'sev 3 * pareto 2.375 - 3 '
            'occurrence ceded to 24 xs 1 '
            'poisson ', bs=1/128)
qd(abcd_ceded)
qd(abcd_ceded.est_sd)
```

```
Portfolio object: ABCD:Ceded. Contains 3 aggregate components. Updated
with bucket size 1/128 and log2 = 16. Validation: reinsurance; subject
fails sev mean, agg mean.

                    Mean      SD      CV    Skew P01   Median     P99
unit       X
Casualty   Freq   125.93  39.409 0.31296 0.60054
           Sev  0.020872 0.23387  11.205  14.019   0        0 0.47656
           Agg    2.6283  2.7503  1.0464  1.3572   0   1.7578  11.461
PropertyNC Freq   34.918  6.8638 0.19657 0.24744
           Sev  0.096787  1.0315  10.657   13.51   0        0  2.4531
           Agg    3.3796  6.1312  1.8142  2.3095   0        0  25.344
PropertyC  Freq   0.5801 0.76165  1.3129  1.3129
           Sev    1.3679  3.0831   2.254  4.2512   0 0.015625  16.859
           Agg    0.7935   2.569  3.2375  5.6851   0        0  13.312
total      Agg    6.8014  7.1941  1.0577  1.7643   0   4.4844  31.516
7.1941
```

4.2.4. Reinsurance Summary

The bottom table shows expected losses, counts, severity, loss ratios and margins implicit in the given reinsurance structure, pricing, and the gross stochastic model. implicit in the given reinsurance structure, pricing, and the gross stochastic model. The non-cat property reinsurance has the highest ceded loss ratio and the cat program the lowest.

> **Note**
>
> This summary previously used the per-unit, per-layer `reinsurance_occ_layer_df` dataframe, removed in 1.0.0a19 in favour of the whole-structure `reins_summary_df` / `reins_stats_df` / `reins_density_df` objects. To rebuild it, take each unit's ceded occurrence aggregate mean and CV from the `('occ', 'ceded', 'agg')` row of `reins_summary_df`, columns `Est EX` and `Est CV`, as the layer loss cost `ex` and `cv`; the ceded count is `unit.n * unit.sev.sf(attach)`. Combine with the given premium to form the loss ratio and margin. For a per-layer split, build each layer as its own occurrence cover.

4.2.5. Underwriting Result Distributions

Make the underwriting result distributions, including the proposed stop loss reinsurance (computed by hand). The dataframe `compare` accumulates the gross, ceded, and net probability mass functions. We use these to determine statistics and to plot.

```
compare = abcd.density_df[['loss', 'p_total']]; \
compare.columns = ['loss', 'gross']; \
compare['gross_uw'] = 33 - compare.loss; \
compare['net_current'] = abcd_net.density_df.p_total; \
compare['net_current_uw'] = 33 - 4.41 - 2.36 - 1.53 - compare.loss;
from aggregate.distributions import make_ceder_netter
compare['net_stoploss'] = abcd.density_df.p_total; \
c, n = make_ceder_netter([(1, 20, 30)]); \
compare['nsll'] = n(compare.loss); \
g = compare.groupby('nsll').net_stoploss.sum(); \
compare['net_stoploss'] = 0.0; \
compare.loc[g.index, 'net_stoploss'] = g; \
compare['net_stoploss_uw'] = 33 - 1.98 - compare.loss;
```

4.2.6. Comparison with the Book's Figures

Statistics summary, compare Figure 2.5.2.

```
from aggregate import MomentWrangler
from scipy.interpolate import interp1d
ans = []; cdfs = []
for xs, den in [(compare.gross_uw, compare.gross), (compare.net_current_uw, compare.net_current),
                (compare.net_stoploss_uw, compare.net_stoploss)]:
    xd = xs * den
    ex1 = np.sum(xd)
    xd *= xs
    ex2 = np.sum(xd)
    ex3 = np.sum(xd * xs)
    mw = MomentWrangler()
    mw.noncentral = ex1, ex2, ex3
    ans.append(mw)
    cdfs.append(interp1d(den.cumsum(), xs))

fig_252 = pd.concat([i.stats for i in ans], keys=['Gross', 'Current', 'StopLoss'], axis=1)
for p in [0.01, 0.99]:
    fig_252.loc[f'q({p})'] = [float(i(p)) for i in cdfs]
qd(fig_252)
```

	Gross	Current	StopLoss
ex	10.15	8.6513	10.021
var	124.41	27.975	58.777
sd	11.154	5.2892	7.6666
cv	1.0989	0.61137	0.76502
skew	-1.6193	-2.0938	-1.0088
q(0.01)	25.98	18.081	24
q(0.99)	-25.76	-5.6036	-7.7399

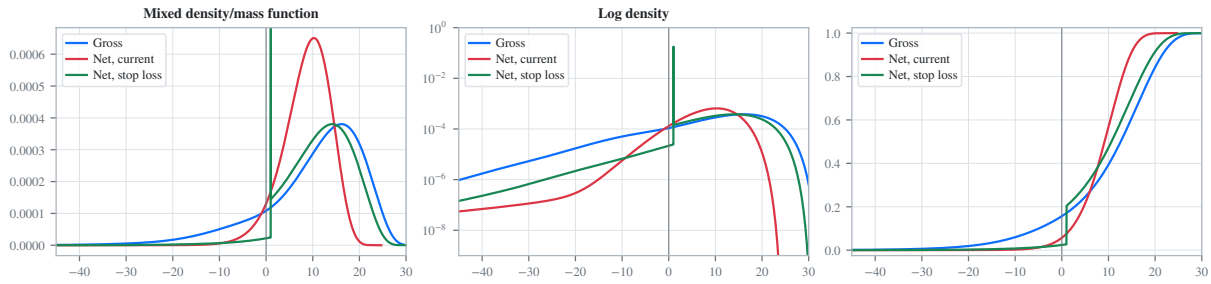
Plot of densities and distributions, compare Figure 2.5.3 and 2.5.4.

```
fig, axs = plt.subplots(1, 3, figsize=(3 * 3.5, 2.45), constrained_layout=True)
ax0, ax1, ax2 = axs.flat

for ax in [ax0, ax1]:
    ax.plot(compare.gross_uw, compare.gross, label='Gross')
    ax.plot(compare.net_current_uw, compare.net_current, label='Net, current')
    yl = ax.get_ylim()
    ax.plot(compare.net_stoploss_uw, compare.net_stoploss, label='Net, stop loss')
    ax.legend(loc='upper left')
```

```
    ax.set(xlim=[-45, 30], ylim=yl)
    ax.axvline(0, lw=.25, c='C7')
ax1.set(yscale='log', ylim=[1e-9, 1], title='Log density'); \
ax0.set(title='Mixed density/mass function');

ax2.plot(compare.gross_uw, 1 - compare.gross.cumsum(), label='Gross'); \
ax2.plot(compare.net_current_uw, 1 - compare.net_current.cumsum(), label='Net, current'); \
ax2.plot(compare.net_stoploss_uw, 1 - compare.net_stoploss.cumsum(), label='Net, stop loss'); \
ax2.legend(loc='upper left'); \
ax2.set(xlim=[-45, 30], ylim=[-0.025, 1.025]);
ax2.axvline(0, lw=.25, c='C7');
```



Numerical distribution of underwriting results at various return points, compare Figure 2.5.5. Given there was no information about the stochastic model provided, and the model here is based on common benchmarks, the agreement between the two distributions is striking.

```
fig_255 = pd.DataFrame(columns=['Gross', 'Current', 'StopLoss'], dtype=float)

for p in [.0025, .005, 0.0075, .01, .0125, .015, .0175, .02,
          .04, .06, .08, .1, .12, .14, .16, .18, .2, .22, .24,
          .25, .26, .28, .3, .32, .34, .36, .38, .4, .42, .44,
          .46, .48, .5]:
    fig_255.loc[p] = [float(i(1-p)) for i in cdfs]
fig_255.index.name = 'p'
qd(fig_255, float_format=lambda x: f'{x:.3f}', max_rows=len(fig_255))
```

p	Gross	Current	StopLoss
0.0025	-38.503	-10.257	-20.483
0.0050	-32.113	-7.826	-14.093
0.0075	-28.394	-6.514	-10.374
0.0100	-25.760	-5.604	-7.740
0.0125	-23.740	-4.902	-5.720
0.0150	-22.114	-4.328	-4.094
0.0175	-20.758	-3.842	-2.738
0.0200	-19.595	-3.420	-1.575
0.0400	-13.603	-1.182	1.021
0.0600	-9.962	0.182	1.021
0.0800	-7.211	1.186	1.022
0.1000	-4.936	1.991	1.023
0.1200	-2.978	2.669	1.024
0.1400	-1.263	3.259	1.025
0.1600	0.246	3.784	1.026
0.1800	1.575	4.259	1.027
0.2000	2.752	4.695	1.028
0.2200	3.801	5.100	1.821
0.2400	4.745	5.478	2.765
0.2500	5.183	5.659	3.203
0.2600	5.603	5.835	3.623

0.2800	6.391	6.173	4.411
0.3000	7.121	6.496	5.141
0.3200	7.802	6.805	5.822
0.3400	8.442	7.103	6.462
0.3600	9.047	7.390	7.067
0.3800	9.622	7.669	7.642
0.4000	10.171	7.941	8.191
0.4200	10.697	8.206	8.717
0.4400	11.203	8.465	9.223
0.4600	11.692	8.720	9.712
0.4800	12.167	8.970	10.187
0.5000	12.628	9.217	10.648

5. Casualty Actuarial Society examination questions

Questions from actuarial society examinations

This chapter works questions from actuarial society examinations. Each section names the examination sitting and the question number. The first question comes from the Casualty Actuarial Society's Exam 9 and applies the percentile layer capital allocation of Bodoff (2007), which the monograph, *Working with Aggregate Probability Distributions*, treats at length in its chapter on Bodoff's method.

5.1. Exam 9, Spring 2018, Question 15

An insurer has exposure to two independent perils, wind and earthquake:

- Wind has a 15% chance of a $5 million loss, and an 85% chance of no loss.
- Earthquake has a 1 % chance of a $15 million loss, and a 99% chance of no loss.

Using the capital allocation by percentile layer methodology with a 99.5% VaR capital requirement, determine how much capital should be allocated to each peril.

Solution.

The last row gives the percentile layer capital.

```
from aggregate import qd
from aggregate.pedagogy import bodoff_exhibit

cas15 = build('port CASq15 '
     'agg X1 1 claim dsev [0,  5] [0.85, 0.15] fixed '
     'agg X2 1 claim dsev [0, 15] [0.99, 0.01] fixed ')
qd(cas15)
# cas15.update(bs=1, log2=8, remove_fuzz=True, padding=1)
cas15.density_df = cas15.density_df.apply(lambda x: np.round(x, 10))
basic = bodoff_exhibit(cas15, reg_p=.995)
qd(basic, col_space=10)
df = cas15.density_df.query('S > 0 or p_total > 0')
```

```
Portfolio object: CASq15. Contains 2 aggregate components. Updated with
bucket size 1 and log2 = 11. Validation: not unreasonable.

            Mean     SD     CV   Skew P01 Median P99
unit  X
X1    Freq     1      0      0
      Sev   0.75 1.7854 2.3805 1.9604   0      0   5
      Agg   0.75 1.7854 2.3805 1.9604   0      0   5
X2    Freq     1      0      0
      Sev   0.15 1.4925 9.9499 9.8494   0      0   0
      Agg   0.15 1.4925 9.9499 9.8494   0      0   0
total Agg    0.9  2.327 2.5856 3.4839   0      0  15
```

method	X1	X2	total
EX	0.75	0.15	0.9
sa VaR	5	15	15
sa TVaR	5	15	16.5
pct EX	12.5	2.5	15
coVaR	-0	15	15
alt coVaR	0.5625	14.438	15
naive coTVaR	0.71429	14.286	15
coTVaR	0.75	15	15.75
plc	5.0714	9.9286	15

The calculation of `plc` as the integral of α for unit 1 is simply:

```
df.exi_xgta_X1.shift(1, fill_value=0).cumsum().loc[15] * cas15.bs
```

```
np.float64(5.071372239499997)
```

A. Every program in the book

The DecL declarations behind the recitations, collected

This appendix collects every DecL program the book builds, chapter by chapter, in the order the chapters appear, each chapter's programs headed by the text they come from and a link back to the chapter. It is not transcribed. Each chapter writes the programs it parses into a store as it runs, and this page reads the store back, so the appendix cannot drift from the chapters the way a hand-maintained list would. What is printed is the canonical form: the declaration re-parsed and written back from the resulting specification, which is what the library understood rather than what was typed.

Three things follow from generating it this way. A chapter that builds nothing does not appear, and neither does a portfolio assembled in Python from aggregates already built, since no declaration was parsed to make it. Programs are collected by declaration rather than by object, so a chapter that rebuilds one model at several bucket sizes contributes one entry and not several, and where it swept the grid the `hints{}` clause is dropped, since no one setting is the model's. And the numerical arguments some chapters pass to `build` as keyword arguments are not here at all: those are instructions to the solver, not part of the declaration, and the chapters that depend on them say so where they use them.

Declarations that share a shape are sampled when there are more than three of them, where a shape is the declaration with its numbers taken out. Two are shown, followed by a count of the rest. A group life example that writes one Bernoulli aggregate per employee contributes two of its fourteen, and nothing is hidden by this that is not an arithmetic variant of something on the page beside it. A shape with three or fewer members is printed in full, because in a book of exercises two declarations of one shape are usually two different problems.

Every program below runs as it stands, in the aggregate Loss Lab with no installation, or through `build` in a Python session.

The same declarations are also available as one loadable file, `recitations.agg` in the project root, written from the same store by `make_agg.py`. Each one there carries a `note{}` naming the text it comes from and the example or exercise it works, and each is renamed by citation key so that nothing in the file shadows anything else: `build.load('recitations.agg')` puts all of them in the recipe base, after which `build('Klugman2019.KPW.9.19')` rebuilds that exercise by name. A model that more than one chapter builds appears there once.

Klugman, Panjer and Willmot on aggregate loss models

From Klugman et al. (2019), worked in Chapter 1.

```
agg KPW.95
  dfreq [0 1 2 3 4 5 6 7 8] [0.05 0.1 0.15 0.2 0.25 0.15 0.06 0.03 0.01]
  dsev [1 2 3 4 5 6 7 8 9 10] [0.15 0.2 0.25 0.125 0.075 0.05 0.05 0.05 0.025 0.025]
```

```
agg KPW.9.19
  dfreq [0 1 3] [0.5 0.4 0.1]
  dsev [1 10] [0.9 0.1]
```

```
agg KPW.9.23
  dfreq [0 1 2 3] [0.5 0.2 0.2 0.1]
  sev 3 * norm + 100
```

```
agg KPW.9.24
  dfreq [0 1 2 3 4] [0.0625 0.25 0.375 0.25 0.0625]
  dsev [2 3] [0.4 0.6]
  aggregate net of
    inf xs 5
```

```
agg KPW.9.31
  [2 3] claims
  sev [1000 200] * uniform
  occurrence ceded to
    inf xs 100
  poisson
```

```
agg KPW.9.34
  5 claims
  dsev [1 5 10] [0.8 0.16 0.04]
  poisson
```

```
agg KPW.9.9
  3 claims
  dsev [1 2 3] [0.6333333333333333 0.26666666666666666 0.09999999999999999]
  poisson
```

```
agg KPW.9.35
  96 claims
  sev 8 * uniform
  mixed gamma 0.25
```

```
agg KPW.9.36
  3 claims
  sev 2 * burr12 1 3
  poisson
```

```
port KPW.9.9.p
  agg A
    2 claims
    dsev [1 2] [0.6 0.4]
    poisson
  agg B
    1 claim
    dsev [1 3] [0.7 0.3]
    poisson
```

```
agg KPW.9.39
  1.2 claims
  sev expon
  binomial 0.4
```

```
port kpw_9_40
  agg A
    50 claims
    dsev [10]
    poisson
  agg B
    60 claims
    dsev [20]
    poisson
  agg C
    30 claims
    dsev [40]
    poisson
```

```
agg KPW.9.11
  0.9 claims
  dsev [0 50 150] [0.3 0.5 0.2]
  binomial 0.3 zm 0.4
```

```
agg KPW.9.45
  6 claims
  dsev [1 2 4]
  poisson
```

```
agg KPW.9.47
  2 claims
  dsev [1 2] [0.25 0.75]
  poisson
```

```
agg KPW.9.57
  3 claims
  dsev [1 2 3 4] [0.4 0.3 0.2 0.1]
  poisson
```

```
agg KPW.9.58
  3 claims
  dsev [1 2 3 4] [0.4 0.3 0.2 0.1]
  mixed gamma 0.408248290463863
```

```
agg KPW.9.59
  5 claims
  sev 2500 * gamma 0.5
  poisson
```

```
agg KPW.9.60
  40 claims
  dsev [5 10] [0.75 0.25]
  neyman 4
```

```
agg KPW.9.14
  3 claims
  18 xs 6
  sev 10 * pareto 4 + -10 !
  occurrence net of
    25% po inf xs 0
  poisson
```

```
agg KPW.9.63
  1.4 claims
  125 xs 50
  sev 100 * gamma 2 !
  mixed gamma 0.7071067811865476
```

```
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
```

```
agg KPW.9.15.w
  0.75 claims
  dsev [50 75 100 150] [0.4666666666666666 0.2333333333333333 0.19999999999999998
  0.09999999999999999]
  binomial 0.01
```

```
agg KPW.9.15.cp
  0.75 claims
  dsev [50 75 100 150] [0.4666666666666666 0.2333333333333333 0.19999999999999998
  0.09999999999999999]
  poisson
```

```
agg ee.1
  0.00149 claims
  dsev [15]
  bernoulli
```

```
agg ee.2
  0.00142 claims
  dsev [16]
  bernoulli
```

and 12 more of this shape, differing only in their numbers.

```
agg kpw_9_16.po
  0.048130000000000006 claims
  dsev [14 15 16 17 18 19 20 24 26 28 30 31 55 60] [0.01038853 0.03095782 0.02950343 0.01038853
  0.07334303 0.01121961 0.02659464 0.10056098 0.08186162 0.02534802 0.02140037 0.02555579
  0.09952213 0.4533555]
  poisson
```

```
agg Class.1
  50 claims
  dsev [1]
  binomial 0.1
```

```
agg Class.2
  10 claims
  dsev [2]
  binomial 0.02
```

and 3 more of this shape, differing only in their numbers.

```
agg Class.2.c
  10 claims
  dsev [1]
  binomial 0.02
```

```
agg Class.3.c
  10 claims
  dsev [2]
  binomial 0.02
```

and 2 more of this shape, differing only in their numbers.

```
agg CP.Approx
  100 claims
  dsev [1 2] [0.9 0.1]
  poisson
```

```
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
```

```
agg KPW.9.74.cp
  [10 7.5 10] claims
  sev [500 750 1000] * expon
  poisson
```

Frees on severity, coverage modifications and aggregate losses

From Frees (2018), worked in Chapter 2.

```
agg lda.3.3.5
  1 claim
  sev [200 200] * [expon pareto] [1 3] + [0 -200] wts [0.25 0.75]
  fixed
```

```
agg lda.3.4.1
  1 claim
  inf xs 100
  sev 1000 * expon 1 !
  fixed
```

```
agg lda.3.4.2
  1 claim
  6 xs 4
  sev 10 * beta 2 1
  fixed
```

```
agg X06
  1 claim
  500 xs 100
  sev 1000 * expon
  fixed
```

```
agg X07
  1 claim
  500 xs 100
  sev 1050 * expon
  fixed
```

```
agg insurer.net
  1 claim
  5000 xs 0
  sev 3600 * pareto 5 + -3600
  occurrence net of
    15% po 5000 xs 0
  fixed
```

```
agg QU
  12 claims
  dsev [1 2 3] [0.5 0.3333333333333333 0.16666666666666666]
  poisson
```

```
agg lda.5.3.2
  dfreq [1 2] [0.8 0.2]
  dsev [0 100 1000] [0.2 0.7 0.1]
```

```
agg lda.5.3.3
  4 claims
  dsev [1 2 3 4]
  geometric
```

```
agg Pharma
  25 claims
  sev 90 * uniform + 5
  poisson
```

```
agg Projects
  2 claims
  dsev [5 10 20] [0.2 0.3 0.5]
  geometric
  aggregate net of
    15 xs 0
```

```
agg Projects.1
  2 claims
  dsev [5 10 20] [0.2 0.3 0.5]
  geometric
  aggregate ceded to
    inf xs 0
```

```
agg Projects.1
  2 claims
  dsev [5 10 20] [0.2 0.3 0.5]
  geometric
  aggregate ceded to
    inf xs 5
```

and 2 more of this shape, differing only in their numbers.

```
agg LDA554
  3 claims
  sev 50 * burr12 1 3
  poisson zm 0.5
```

```
agg Original
  300 claims
  dsev [4 8 12 20]
  mixed gamma 0.07453559924999298
```

```
agg Revised
  300 claims
  inf xs 10
  sev dhistogram xps [6 12 18 30] [0.25 0.25 0.25 0.25] !
  mixed gamma 0.07453559924999298
```

```
agg Auto
  16 claims
  inf xs 100
  sev 200 * expon !
  poisson
```

```
agg Auto
  16 claims
  inf xs 100
  sev 200 * expon
  poisson
```

```
agg VaR
  1 claim
  dsev [1 3 4] [0.75 0.2 0.05]
  fixed
```

```
port retained
  agg building
    1 claim
    100 xs 0
    sev 100 * gamma 2
    fixed
  agg motor
    1 claim
    200 xs 0
    sev 200 * gamma 2
    fixed
```

```
port insured
  agg building
    1 claim
    inf xs 100
    sev 100 * gamma 2 !
```

```
    fixed
  agg motor
    1 claim
    inf xs 200
    sev 200 * gamma 2 !
    fixed
  agg d.and.o
    1 claim
    sev 1000 * pareto 2
    fixed
  agg cyber
    1 claim
    sev 2000 * pareto 3
    fixed
```

```
port total
  agg building
    1 claim
    sev 100 * gamma 2
    fixed
  agg motor
    1 claim
    sev 200 * gamma 2
    fixed
  agg d.and.o
    1 claim
    sev 1000 * pareto 2
    fixed
  agg cyber
    1 claim
    sev 2000 * pareto 3
    fixed
```

Bahnemann on aggregate distributions and layer pricing

From Bahnemann (2015), worked in Chapter 3.

```
agg Bahn.4.1
  dfreq [0 1 2] [0.6 0.3 0.1]
  dsev [1 2 3] [0.4 0.5 0.1]
```

```
agg Bahn.4.2
  2.5 claims
  sev 400 * gamma 3
  poisson
```

```
agg Bahn.4.5
  2.5 claims
  sev 400 * gamma 3
  poisson
```

```
agg Bahn.4.7
  8 claims
  sev 3750 * gamma 0.2
  poisson
```

```
agg Bahn.4.2b
  10 claims
  sev 6000 * gamma 0.05
  poisson
```

```
agg Bahn.4.4
  1.75 claims
  dsev [1 2 3 4 5] [0.2 0.4 0.2 0.15 0.05]
  poisson
```

```
agg Bahn.4.15
  3 claims
  1000 xs 0
  sev 403.4287934927351 * lognorm 1.5
  poisson
```

```
agg Bahn.5.13
  15 claims
  5000 xs 3000
  sev 395.7964246880697 * lognorm 1.8 !
  poisson
```

```
agg Bahn.5.13b
  1.9535923221305609 claims
  5000 xs 3000
  sev 395.7964246880697 * lognorm 1.8
  poisson
```

```
agg Bahn.6.3
  1 claim
  sev 1096.6331584284585 * lognorm 2.4
  fixed
```

```
agg Bahn.6.5
  0.2 claims
  [100000 500000 1000000 2000000 3000000 4000000 5000000] xs 0
  sev 1096.6331584284585 * lognorm 2.4
  poisson
```

```
agg Bahn.6.6
  1.2 claims
  500000 xs 0
  sev 1998.1958951041172 * lognorm 2.4
  mixed gamma 0.31622776601683794
```

```
  aggregate ceded to
    1000000 xs 0 and
    1000000 xs 1000000 and
    1000000 xs 2000000 and
    1000000 xs 3000000 and
    1000000 xs 4000000 and
    inf xs 5000000
```

```
agg Bahn.6.6
  1.2 claims
  1000000 xs 0
  sev 1998.1958951041172 * lognorm 2.4
  mixed gamma 0.31622776601683794
  aggregate ceded to
    1000000 xs 0 and
    1000000 xs 1000000 and
    1000000 xs 2000000 and
    1000000 xs 3000000 and
    1000000 xs 4000000 and
    inf xs 5000000
```

```
agg Bahn.6.6
  1.2 claims
  2000000 xs 0
  sev 1998.1958951041172 * lognorm 2.4
  mixed gamma 0.31622776601683794
  aggregate ceded to
    2000000 xs 0 and
    1000000 xs 2000000 and
    1000000 xs 3000000 and
    1000000 xs 4000000 and
    inf xs 5000000
```

```
agg Bahn.6.6
  1.2 claims
  3000000 xs 0
  sev 1998.1958951041172 * lognorm 2.4
  mixed gamma 0.31622776601683794
  aggregate ceded to
    3000000 xs 0 and
    1000000 xs 3000000 and
    1000000 xs 4000000 and
    inf xs 5000000
```

```
agg Bahn.6.6
  1.2 claims
  4000000 xs 0
  sev 1998.1958951041172 * lognorm 2.4
  mixed gamma 0.31622776601683794
  aggregate ceded to
    4000000 xs 0 and
    1000000 xs 4000000 and
    inf xs 5000000
```

```
agg Bahn.6.6
  1.2 claims
  5000000 xs 0
  sev 1998.1958951041172 * lognorm 2.4
  mixed gamma 0.31622776601683794
  aggregate ceded to
    5000000 xs 0 and
    inf xs 5000000
```

```
agg Bahn.6.7
  0.0005 claims
  [100000 99000 98000 97000 96000 95000 90000] xs [0 1000 2000 3000 4000 5000 10000]
  sev 1096.6331584284585 * lognorm 2.4 !
  poisson
```

Brehm et al. on reinsurance in enterprise risk analysis

From Brehm et al. (2007), worked in Chapter 4.

```
port ABCD
  agg Casualty
    14 premium at 0.78 lr
    5 xs 0
    sev lognorm 0.1 cv 10
    mixed gamma 0.3
  agg PropertyNC
    17 premium at 0.63 lr
    25 xs 0
    sev lognorm [0.1 1] cv [5 10] wts [0.7 0.3]
    mixed gamma 0.1
  agg PropertyC
    2 premium at 0.63 lr
    150 xs 0
    sev 3 * pareto 2.375 + -3
    poisson
```

```
port ABCD:Net
  agg Casualty
    14 premium at 0.78 lr
    5 xs 0
    sev lognorm 0.1 cv 10
    occurrence net of
      4 xs 1
    mixed gamma 0.3
  agg PropertyNC
    17 premium at 0.63 lr
    25 xs 0
    sev lognorm [0.1 1] cv [5 10] wts [0.7 0.3]
    occurrence net of
      17 xs 3
    mixed gamma 0.1
  agg PropertyC
    2 premium at 0.63 lr
    150 xs 0
    sev 3 * pareto 2.375 + -3
```

```
    occurrence net of
      24 xs 1
    poisson
```

```
port ABCD:Ceded
  agg Casualty
    14 premium at 0.78 lr
    5 xs 0
    sev lognorm 0.1 cv 10
    occurrence ceded to
      4 xs 1
    mixed gamma 0.3
  agg PropertyNC
    17 premium at 0.63 lr
    25 xs 0
    sev lognorm [0.1 1] cv [5 10] wts [0.7 0.3]
    occurrence ceded to
      17 xs 3
    mixed gamma 0.1
  agg PropertyC
    2 premium at 0.63 lr
    150 xs 0
    sev 3 * pareto 2.375 + -3
    occurrence ceded to
      24 xs 1
    poisson
```

Casualty Actuarial Society examination questions

From the examinations cited in Chapter 5.

```
port CASq15
  agg X1
    1 claim
    dsev [0 5] [0.85 0.15]
    fixed
  agg X2
    1 claim
    dsev [0 15] [0.99 0.01]
    fixed
```

5 chapters build 97 distinct programs between them, of which 78 are printed above and 19 are further declarations of a shape already shown.

References

Bahnemann, D. (2015). *Distributions for Actuaries*. Casualty Actuarial Society Monographs No. 2. www.casact.org

Bodoff, N. M. (2007). Capital Allocation by Percentile Layer. *Variance*, *3*(1), 13–30.

Brehm, P. J., Perry, J. H., Venter, G. G., & Witcraft, S. E. (2007). *Enterprise Risk Analysis*. Casualty Actuarial Society.

Frees, E. (2018). *Loss Data Analytics*. Open Actuarial Texts. https://openacttexts.github.io/Loss-Data-Analytics/

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