Frees on severity, coverage modifications and aggregate losses

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

Stephen J. Mildenhall

2026-10-22 19:57 commit b7145b5+

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

1 Distribution Examples

1.1 Gamma distribution

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

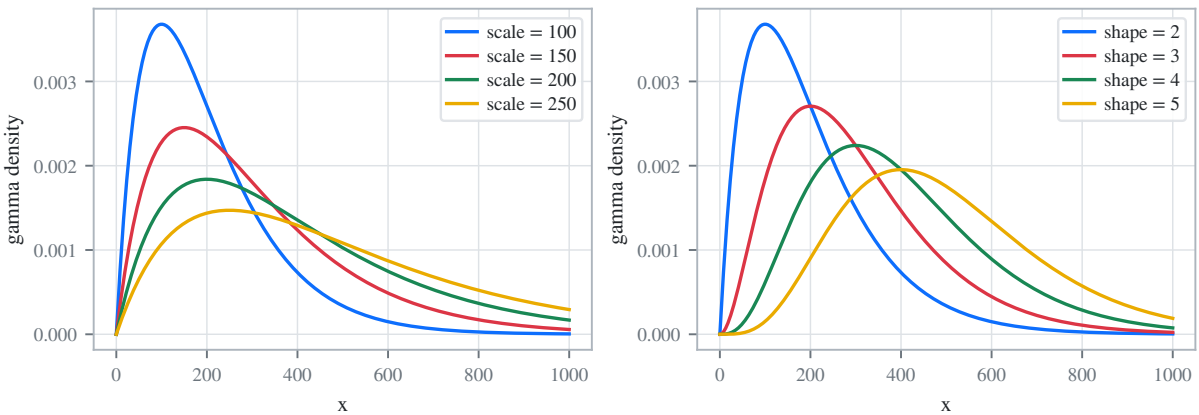
xs = np.linspace(0, 1000, 1001)

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')
```



1.2 Pareto distribution

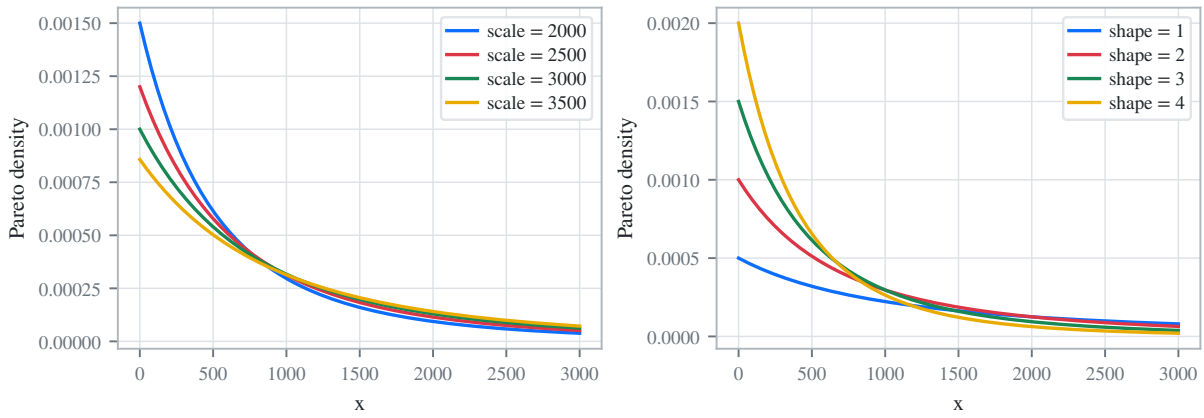
```
xs = np.linspace(0, 3000, 3001)

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')
```



1.3 Weibull distribution

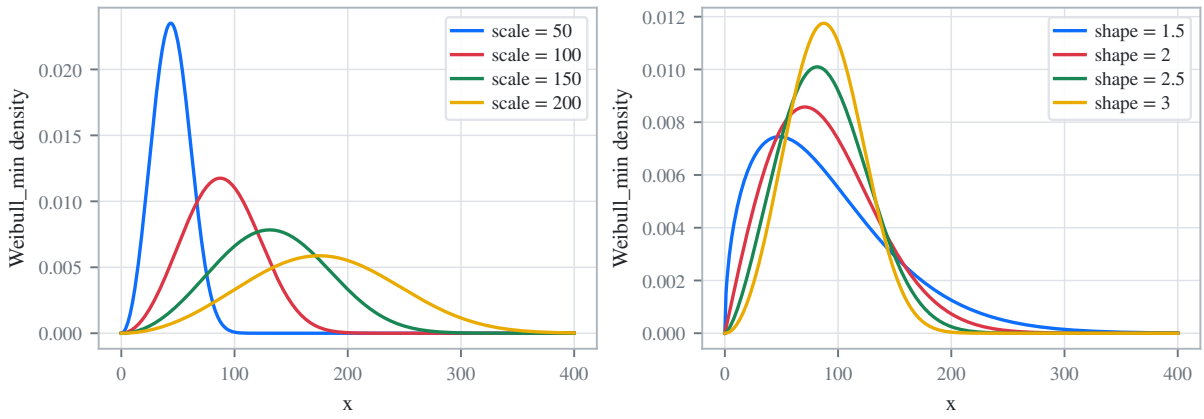
`scipy.stats` includes Weibull min (for positive x) and Weibull max (for negative x) distributions. We want the min version.

```
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 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))
```

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)

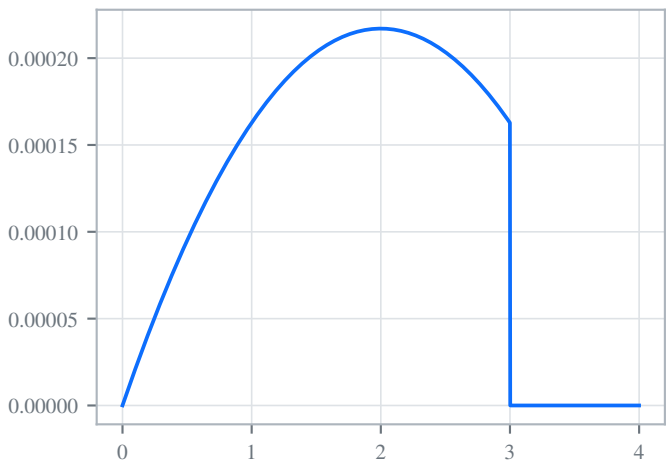
Under a group insurance policy, an insurer agrees to pay 100% of the medical bills incurred during the year by employees of a small company, up to a maximum total of one million dollars. The total amount of bills incurred, X, has *pdf*

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

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 $0.85\,\mathsf{P}(X \wedge 5000) = 0.85 \times 900\,(1 - (3600/8600)^4) = 0.85 \times 872.37 = 741.51$.

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))
```

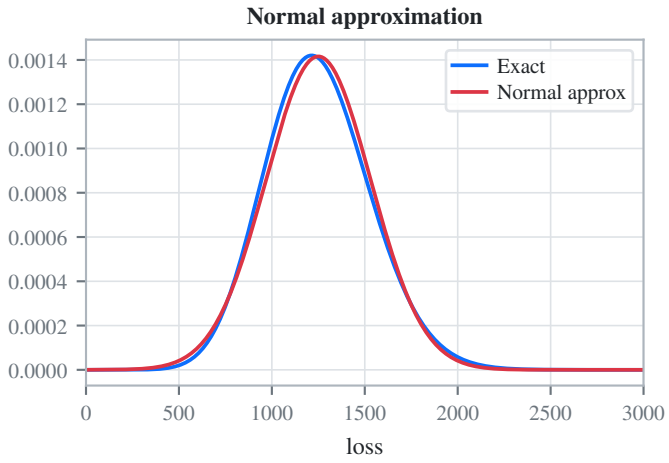
`approximate` will also provide (shifted) gamma and lognormal fits.

```
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)
```

attach	cumul ex
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)
```

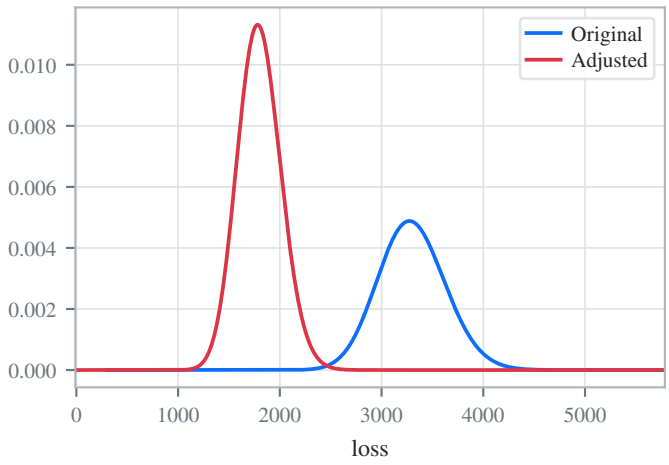
```
Aggregate object: Original. Frequency distribution gamma. Severity
dhistogram, [4, 20], bounded; atoms [4 8 12 20]. Updated with bucket
size 4 and log2 = 11. Validation: not unreasonable.

      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
```

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')
```

