Using `ssev`

Signed severities with ssev, and aggregates or portfolios as severities

Stephen J. Mildenhall

2026-10-30

The DecL keyword `ssev`[1] declares a **signed severity**: it keeps the negative part of the support of a severity, where `sev` moves it onto an atom at zero. So `sev X` is the loss $X^+ = \max(X, 0)$, and `ssev X` has the law of X itself. A negative loss is a gain, so with `ssev` a profit can enter an aggregate. A signed aggregate stays on the loss convention, with `value_type` equal to `loss`: positive values are losses and negative values are gains.

Either keyword also takes an aggregate or a portfolio stored in the recipe base (see Using recipes), as `sev agg.NAME` or `sev port.NAME`. The severity is then the law of the total S of that aggregate or portfolio: its computed probability mass function is used exactly as a `dsev`. With an outer count N the result is $S_1 + \cdots + S_N$, for example a book of policies whose loss per policy is itself a compound.

A layers clause `y xs a` conditions a referenced severity on exceeding the attachment, as it does any severity, so `300 xs 0` turns S into $S \mid S > 0$: the atom $p_0 = \Pr(S = 0)$ is removed and the rest is scaled by $1/(1 - p_0)$. A trailing `!` makes the layer unconditional and keeps the atom; with no layers clause the atom stays as well. For a Poisson count the two are thinnings of each other: λ policies with the atom are $\lambda(1 - p_0)$ policies without it.

```
sev  10 * norm + 5              # a loss: the negative part is clamped to zero
ssev 10 * norm + 5              # signed: the negative part is kept
sev  agg.NAME                   # the total of agg NAME as the severity
300 xs 0 sev agg.NAME !         # limited at 300, the atom at zero kept
ssev port.NAME as "Desk"        # a signed portfolio total, with a label
```

The referenced declaration must fix its own grid with `hints{log2=...; bs=...}`. A reference cannot be modified, so scaling, shifting, picks, and splicing are not available.

```
import warnings
import numpy as np
from scipy import stats as ss
from aggregate import build, qd, __version__
from style import wprint, GT, refuse, figures
figures()
print(__version__)
```

```
1.43.1
```

[1] From aggregate 1.0, as are severity references to aggregates and portfolios.

A normal severity under sev and ssev

A normal severity with mean $\mu = 5$ and standard deviation $\sigma = 10$ shows the difference between the two keywords. Under `sev` the mean is $\mathsf{P}X^+ = \sigma\phi(\mu/\sigma) + \mu\Phi(\mu/\sigma) = 6.978$, where ϕ and Φ are the standard normal density and distribution function. Under `ssev` it is 5.

```
a = build('agg Clamped dfreq [1] sev 10 * norm + 5')
b = build('agg Signed dfreq [1] ssev 10 * norm + 5')
qd(a)
qd(b)
print(10 * ss.norm.pdf(0.5) + 5 * ss.norm.cdf(0.5))
```

```
Aggregate object: Clamped. Frequency distribution empirical. Severity
norm, [0, inf), super-exponential right tail. Updated with bucket size
1/256 and log2 = 16. Validation: not unreasonable.

      Mean     SD     CV   Skew P01 Median    P99
X
Freq     1      0      0
Sev  6.978 7.4394 1.0661 1.0139   0      5 28.262
Agg  6.978 7.4394 1.0661 1.0139   0      5 28.262
Aggregate object: Signed. Frequency distribution empirical. Severity
norm, (-inf, inf), super-exponential both tails; signed, support [-inf,
inf]. Updated with bucket size 1/256 and log2 = 16. Validation: not
unreasonable.

      Mean  SD  CV       Skew     P01 Median    P99
X
Freq     1   0   0
Sev      5  10   2 3.4656e-10 -18.262      5 28.262
Agg      5  10   2 3.4656e-10 -18.262      5 28.262
6.97796557401306
```

Both laws put probability $\Phi(-1/2) = 0.309$ at or below zero: the clamped law as an atom at zero, the signed law spread over the negative half line. The signed aggregate still has `value_type` equal to `loss`.

```
print(a.prob_eq_0, a.cdf(0), b.cdf(0), ss.norm.cdf(-0.5), b.value_type)
```

```
0.3086063048423492 0.3086063048423492 0.3086063048416497 0.3085375387259869 loss
```

An aggregate as a severity

An aggregate used as a severity must fix its own grid with `hints`. Our example is the loss on one policy, a Poisson count with mean 1/2 and a lognormal severity limited to 100. Its atom at zero is $p_0 = e^{-1/2}$, the probability of no claim, since a lognormal puts almost no probability near zero.

```
policy = build('agg Policy 0.5 claims 100 xs 0 sev lognorm 50 cv 1 poisson '
               'hints{log2=12; bs=1/4}')
p0 = policy.density_df.p_total.iloc[0]
print(p0, np.exp(-0.5))
```

```
0.6065306597144531 0.6065306597126334
```

A book of 100 policies is built four ways: with no layers clause, with a 300 limit and the default conditional layer, with the same limit and `!`, and with the conditional layer on the thinned count $100(1 - p_0) = 39.35$. The first and third keep the atom, so their severity mean is the policy mean, 21.83. The second removes it, and the mean rises to $21.83/(1 - p_0) = 55.47$. The last two are the same law, with mean 2,182.6 and standard deviation 430.25.

```
books = [build('agg Open 100 claims sev agg.Policy poisson'),
         build('agg Cond 100 claims 300 xs 0 sev agg.Policy poisson'),
         build('agg Uncond 100 claims 300 xs 0 sev agg.Policy ! as "Per policy" poisson'),
         build(f'agg Thin {100 * (1 - p0)} claims 300 xs 0 sev agg.Policy poisson')]
for bk in books:
    print(f'{bk.name:8s} {bk.n:8.3f} {bk.est_sev_m:8.3f} {bk.est_m:10.3f} {bk.est_sd:10.4f}')
```

```
Open      100.000   21.828   2182.808   430.3828
Cond      100.000   55.472   5547.160   685.9122
Uncond    100.000   21.826   2182.637   430.2531
Thin       39.347   55.472   2182.637   430.2531
```

The attribute `pprogram` of an aggregate shows its program as DecL, here with the reference and its label. The label is also available as `labels.severity`.

```
print(books[2].pprogram)
print(books[2].labels.severity)
```

```
agg Uncond
  100 claims
  300 xs 0
  sev agg.Policy ! as "Per policy"
  poisson
Per policy
```

The tail of a referenced severity is the tail of its source, not that of its finite set of atoms. The aggregate `Policy` has a Poisson count, so the sum of two policies is unbounded until a limit applies.

```
print(build('agg Two dfreq [2] sev agg.Policy').bounded,
      build('agg TwoCapped dfreq [2] 300 xs 0 sev agg.Policy !').bounded)
```

```
False True
```

A signed aggregate as a severity

A signed aggregate is used as a severity with `ssev`. Our example is a trading day, a Poisson number of trades with mean 4, each a gain of 10 (entered as a loss of −10) or a loss of 5 or 20. A `dsev` keeps a negative outcome as given, so the aggregate is signed. The method `with_hints()` returns the program with the grid chosen by the build, here bucket size 5, ready to register as a source.

```
day = build('agg Day 4 claims dsev [-10 5 20] [0.3 0.5 0.2] poisson')
wprint(day.with_hints())
day = build(day.with_hints())
```

```
agg Day 4 claims dsev [-10 5 20] [0.3 0.5 0.2] poisson hints{log2=7; bs=5;
normalize=True}
```

A month is the sum of 21 independent days, `ssev agg.Day` with a fixed count. A sum of independent compound Poisson variables with a common severity is compound Poisson, so the month is a compound Poisson with mean count 84. On this lattice, with bucket size 5, the two computations agree to within 2×10^{-14}.

```
month = build('agg Month dfreq [21] ssev agg.Day')
check = build('agg Check 84 claims dsev [-10 5 20] [0.3 0.5 0.2] poisson')
print([float(month.q(p)) for p in [0.01, 0.5, 0.99]],
      [float(check.q(p)) for p in [0.01, 0.5, 0.99]])
print(np.abs(month.density_df.p_total - check.density_df.p_total).max())
```

```
[65.0, 290.0, 540.0] [65.0, 290.0, 540.0]
1.5602102942935403e-14
```

Under plain `sev` the negative outcomes of the source are moved onto zero, with a warning. A day is negative with probability 0.227, and with those outcomes clamped to zero the mean of the month rises from 294 to 357.

```
with warnings.catch_warnings(record=True) as caught:
    warnings.simplefilter('always')
    clamped = build('agg MonthClamped dfreq [21] sev agg.Day')
wprint(caught[0].message)
print(month.est_m, clamped.est_m)
```

```
agg.Day: a signed source is being used as an unsigned severity; 0.22693 of
probability at negative outcomes is clamped onto the zero atom. Write ssev to
keep the negative support -- a signed source in a loss context usually means the
value conventions do not match.
294.00000000697884 356.54100603545743
```

A portfolio as a severity

A portfolio reference, `sev port.NAME`, uses the distribution of the portfolio total as the severity. On a `port` the `hints` clause goes on the header line, after the name. The mean loss per policy of the book equals the portfolio mean, $0.5 \times 50 + 0.2 \times 200 = 65$.

```
pair = build('port Pair hints{log2=14; bs=1/2} '
             'agg A 0.5 claims sev lognorm 50 cv 1 poisson '
             'agg B 0.2 claims sev lognorm 200 cv 1 poisson')
book = build('agg PairBook 100 claims sev port.Pair poisson')
print(pair.est_m, book.est_m / 100)
```

```
64.99896477679953 64.99891365678513
```

Refused references

A reference to a declaration without a fixed grid is refused, and so is a reference inside a `bivariate`. The first error message includes the `hints` clause to add.

```
build('agg Loose 0.5 claims sev lognorm 50 cv 1 poisson')
refuse('agg Outer 100 claims sev agg.Loose poisson',
       'bivariate BV 25 claims agg A dfreq [1] sev agg.Policy '
       'agg B dfreq [1] sev lognorm 10 cv 1 copula independent poisson')
```

Outer: the severity reference 'agg.Loose' needs agg.Loose to pin its own grid, and it declares no log2 and bs. A reference stands for the distribution that declaration outputs, so the declaration has to say at what resolution, or the severity changes with the ambient defaults instead of with the model. Update the source at the resolution you want and re-register it with its grid pinned, `build(x.with_hints())`, or add the clause by hand: hints{log2=16; bs=1/4}
BV: a severity reference (sev agg.NAME / sev port.NAME) is not supported in a 'bvagg' body. Declare the compound as its own agg and reference that agg by name from an ordinary aggregate.