# Using `price`

**Distorted ask and bid prices of a loss distribution with Distortion.price and Aggregate.price**

Stephen J. Mildenhall

2026-11-06

A distortion $g$ (see Using distortions) prices a loss $X \ge 0$ supported by assets $a$ as

$$g[X \wedge a] = \int_0^a g(S(x))\,dx,$$

where $S$ is the survival function of $X$. Square brackets mark $g$ acting on a random variable, as a spectral risk measure, and parentheses mark $g$ acting on a probability. Losses above $a$ are not paid, so the price is that of $X \wedge a$, and $a = \infty$ prices the whole loss. This price is the **ask**, the premium for taking the loss on. The **bid** uses the dual $\check{g}(s) = 1 - g(1 - s)$. It equals $-g[-(X \wedge a)]$, the most a party pricing with $g$ would pay to receive $X \wedge a$, and for a concave $g$

$$\check{g}[X \wedge a] \le \mathsf{P}(X \wedge a) \le g[X \wedge a].$$

For a discrete $X$ with outcomes $x_0 < x_1 < \cdots$ the integral is a sum. The distorted probability of $x_k$ is $q_k = g(\Pr(X \ge x_k)) - g(\Pr(X > x_k))$, and $g[X \wedge a] = \sum_k \min(x_k, a)\, q_k$, an expectation under the probabilities $q$. The library computes the price this way, exactly, on the outcomes of the distribution.

The price is available in two ways. The method `Distortion.price(ser, a=np.inf, kind='ask')` prices any distribution given as a pandas Series of probabilities indexed by outcome. Its `kind` is `'ask'`, `'bid'`, or `'both'`, and it returns a named tuple `(bid, el, ask)`, or a one-row frame with `as_frame=True`; the asset level must be one of the outcomes. The method `Aggregate.price(p, g, kind='var')` prices an aggregate with assets equal to its VaR at probability `p`, or its TVaR with `kind='tvar'`. It returns the pricing **pentagon**: expected loss $L$, margin $M$, premium $P = L + M$, capital $Q$, and assets $a = P + Q$, with the loss ratio $\mathtt{LR} = L/P$, the premium to capital ratio $\mathtt{PQ} = P/Q$, and the return on capital $\mathtt{ROE} = M/Q$. The examples use one aggregate and one distortion.

```
agg Book 25 claims 2000 xs 0 sev lognorm 100 cv 4 poisson
distortion Gentle ph 0.6
```

```
import numpy as np
import pandas as pd
from aggregate import build, qd, Distortion, __version__
from aggregate.spectral import consistent_distortions
from style import wprint, GT, refuse, figures
figures()
print(__version__)
```

```
2.1.0a5
```

### A three-point loss priced by hand

A loss of 0, 100, or 1,000, with probabilities 0.9, 0.09, and 0.01, can be priced by hand. Its survival function is 0.1 on $[0, 100)$ and 0.01 on $[100, 1000)$, so with $g(s) = s^{0.6}$ the ask is $100\,g(0.1) + 900\,g(0.01) = 25.1 + 56.8 = 81.9$, against an expected loss of 19. The bid is $100\,\check{g}(0.1) + 900\,\check{g}(0.01) = 6.1 + 5.4 = 11.5$. With assets of 100 only the first layer is priced: the ask is 25.1, the expected loss 10, and the bid 6.1.

```
gentle = build('distortion Gentle ph 0.6')
x = pd.Series([0.9, 0.09, 0.01], index=[0, 100, 1000])
print(gentle.price(x, kind='both'))
print(gentle.price(x, a=100, kind='both'))
```

```
Price(bid=11.536811368514599, el=19.000000000000007, ask=81.90502531831322)
Price(bid=6.1259606640430615, el=9.999999999999996, ask=25.1188643150958)
```

### Prices of an aggregate with Distortion.price

The aggregate `Book`, used here and on Using `price` and `evaluate`, has 25 claims a year, Poisson, from a lognormal severity with mean 100 and CV 4 under a policy limit of 2,000. The limited severity has mean 91.3 and the aggregate mean 2,283.7.

```
book = build('agg Book 25 claims 2000 xs 0 sev lognorm 100 cv 4 poisson')
qd(book)
```

```
Aggregate object: Book. Frequency distribution poisson. Severity
lognorm, [0, 2,000], bounded; 2,000 xs 0. Updated with bucket size 1 and
log2 = 16. Validation: not unreasonable.

       Mean     SD      CV   Skew P01 Median  P99
X
Freq     25      5     0.2    0.2
Sev  91.349 218.98  2.3971 5.5614   0     24 1217
Agg  2283.7 1186.3 0.51947 1.0825 534   2036 5903
```

Any distribution in a Series can be priced, and the column `p_total` of `density_df` is the distribution of the aggregate. Unlimited, the ask is 2,994.9 against a mean of 2,283.7. With assets at the 99.5% VaR, $a = 6{,}438$, the limited expected loss is 2,280.1, the ask 2,945.6, and the bid 1,833.3.

```
ser = book.density_df.p_total
a = book.q(0.995)
GT(pd.DataFrame([gentle.price(ser, kind='both'), gentle.price(ser, a=a, kind='both')],
                index=pd.Index(['unlimited', f'{a:,.0f}'], name='assets')),
   float_format=',.1f')
```

| *assets* | bid | el | ask |
|---|---|---|---|
| unlimited | 1,835.5 | 2,283.7 | 2,994.9 |
| 6,438 | 1,833.3 | 2,280.1 | 2,945.6 |

### Prices of an aggregate with Aggregate.price

The method `Aggregate.price` takes a probability `p` or an asset level `a=`, one of them, and returns the whole pentagon. At `p = 0.995` the assets are 6,438, the 99.5% VaR, and the premium $P$ is the same as the ask from `Distortion.price` at those assets. The margin of 665.5 is a return of 19.1% on capital of 3,492.4, at a loss ratio of 77.4%. With `kind='tvar'` the assets are the 99.5% TVaR, 7,166 on the grid, and the larger capital lowers the return to 16.3%.

```
price = pd.concat([book.price(0.995, gentle, kind=k).rename(index={'Book': k})
                   for k in ['var', 'tvar']])
print(price.loc['var', 'P'] - gentle.price(ser, a=a).ask)
GT(price, float_format=',.1f', ratio_cols=['LR', 'PQ', 'ROE'])
```

```
-5.9117155615240335e-12
```

| *unit* | L | M | P | Q | a | LR | PQ | ROE |
|---|---|---|---|---|---|---|---|---|
| var | 2,280.1 | 665.5 | 2,945.6 | 3,492.4 | 6,438 | 77.4% | 84.3% | 19.1% |
| tvar | 2,282.4 | 686.2 | 2,968.6 | 4,197.4 | 7,166 | 76.9% | 70.7% | 16.3% |

A distortion with a mass at zero prices at any finite level, even on a book with unbounded losses, because $X \wedge a$ is bounded and the mass lands at $a$. The constant cost of capital gives $P = (L + ra)/(1 + r)$, and the two numbers below are the premium and that formula.

```
coc = Distortion('ccoc', r=0.1)
row = book.price(g=coc, a=a).iloc[0]
print(row.P, (row.L + 0.1 * a) / 1.1)
```

```
2658.0794379107156 2658.0794379111408
```

### Five distortions with one Gini index

The five distortions with gini $= 0.25$ from `consistent_distortions` have the same index and price the book differently. With assets 6,438 the ask runs from 2,695.8 for TVaR to 3,319.6 for `ccoc`, in the order of the weight each puts on the extreme tail. For `ccoc` the ask is $(1 - d)L + d\,a$ and the bid $(1 - d)L$, with $d = 0.25$: the expected loss moved a quarter of the way to the assets, or to the smallest loss, zero.

```
rows = {k: [d.gini, d.label, *d.price(ser, a=a, kind='both')]
        for k, d in consistent_distortions(0.25).items()}
GT(pd.DataFrame.from_dict(rows, orient='index',
                          columns=['gini', 'label', 'bid', 'el', 'ask']).rename_axis('kind'),
   float_format=',.1f', formatters={'gini': '{x:.3f}'})
```

| *kind* | gini | label | bid | el | ask |
|---|---|---|---|---|---|
| ccoc | 0.250 | ccoc(0.333) | 1,710.1 | 2,280.1 | 3,319.6 |
| ph | 0.250 | PH(0.6) | 1,833.3 | 2,280.1 | 2,945.6 |
| wang | 0.250 | Wang(0.451) | 1,810.2 | 2,280.1 | 2,830.2 |
| dual | 0.250 | dual(1.67) | 1,780.1 | 2,280.1 | 2,750.8 |
| tvar | 0.250 | TVaR(0.25) | 1,731.9 | 2,280.1 | 2,695.8 |

### Refused calls

The method `Distortion.price` refuses an asset level that is not an outcome, since it does not interpolate; a `kind` other than ask, bid, or both; and probabilities passed as an array without their outcomes. The method `Aggregate.price` refuses a `p` above 1, which is not a probability, and `p = 1` on a book with unbounded losses, which has no maximum. The method `apply_distortion` has no asset level, so it refuses a mass distortion on such a book.

```
refuse(lambda: gentle.price(ser, a=6438.5),
       lambda: gentle.price(ser, kind='mid'),
       lambda: gentle.price(ser.to_numpy()),
       lambda: book.price(1.5, gentle),
       lambda: book.price(1, gentle),
       lambda: book.apply_distortion(coc))
```

```
a=6438.5 must be in the index of ser
kind must be 'bid', 'ask', or 'both'
ser must be a pandas Series, not <class 'numpy.ndarray'>
price: p must be a probability of at most 1, got 1.5; pass an asset level as
a=1.5
price: Book is unbounded, so p=1 does not fix an asset level. It resolves to the
top of the FFT grid, which moves with log2 rather than with the risk, and every
number priced off it moves with it. Pass a= to set the level (a=Book.q(1) is the
top of the grid), use a p below 1, or certify the support with Book.bounded =
True. tail_behavior_df shows the tail.
mass distortion (ccoc(0.1)) on an unbounded aggregate: the mass lands on the
last represented bucket, a different bounded problem. Use price with p below 1
or a= for a finite asset level, or certify `bounded = True` if the support is in
fact bounded.
```