# Using `price` and `evaluate`

**Premiums from distortions and breakeven distortions from premiums on an Aggregate with price, price_pentagon and evaluate**

Stephen J. Mildenhall

2026-11-06

Pricing and evaluation solve the same equation for different unknowns. With assets $a$ behind a loss $X$, a distortion $g$ and a premium $P$ are consistent when

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

Square brackets mark $g$ acting on a random variable, as a spectral risk measure, and parentheses mark $g$ acting on a probability. Pricing fixes $g$ and solves for $P$, which is what `Aggregate.price(p, g)` does, with $a$ the VaR at probability $p$ (see Using `price`). Evaluation fixes $P$ and solves, for each family of distortions, for the shape that satisfies the equation. The position $P - X \wedge a$ is acceptable to every distortion in the family up to that shape, so the **breakeven** distortion is the most stress the position survives, the acceptability level of Cherny and Madan. Its gini (see Using distortions) is an acceptability index on one scale for every family: the larger it is, the more stress the premium allows for.

Both answers come with the accounting of the **pentagon**: expected loss $L = \mathsf{P}(X \wedge a)$, margin $M = P - L$, premium $P$, capital $Q = a - P$, and assets $a$, with the ratios $\mathsf{LR} = L/P$, $\mathsf{PQ} = P/Q$, and $\mathsf{ROE} = M/Q$. The method `Aggregate.price_pentagon` completes the pentagon from the assets, given by `p=` or `a=`, and one target, one of `P`, `M`, `Q`, `LR`, `PQ`, and `ROE`. No distortion is involved: the assets fix $L$, and the identities do the rest.

The method `Aggregate.evaluate(P, p=None, a=None)` returns an `EvaluationResult`, whose `evaluation_df` has one row per family. With `p=` or `a=` the position is $P - X \wedge a$ and the families are `ccoc`, `ph`, `wang`, `dual`, and `tvar`. Without either, the position is $P - X$ against the whole distribution, and `ccoc`, which needs an asset level, is left out. The argument `P` defaults to the premium declared in the DecL program. The examples use the same book as Using `price`.

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

```
import warnings
import pandas as pd
from aggregate import build, Distortion, __version__
from style import wprint, GT, refuse, figures
figures()
print(__version__)
book = build('agg Book 25 claims 2000 xs 0 sev lognorm 100 cv 4 poisson')
gentle = build('distortion Gentle ph 0.6')
```

```
2.1.0a5
```

### The premium and pentagon from a distortion

At $p = 0.995$ the assets are 6,438, and the method `price` with $g(s) = s^{0.6}$ gives a premium of 2,945.6, a return of 19.1% on capital. The method `price_pentagon` with the same assets and that premium returns the same pentagon from the identities alone. Given a cost of capital of 15% in place of the premium, it returns the premium that earns it, 2,822.4, which is $L + (a - L)\, r/(1 + r)$ with $r = 0.15$.

```
priced = book.price(0.995, gentle)
P, L = priced.loc['Book', ['P', 'L']]
pent = pd.concat([priced.rename(index={'Book': 'price'}),
                  book.price_pentagon(p=0.995, P=P).rename(index={'total': 'price_pentagon(P)'}),
                  book.price_pentagon(p=0.995, ROE=0.15).rename(index={'total':
'price_pentagon(ROE)'})])
GT(pent, float_format=',.1f', ratio_cols=['LR', 'PQ', 'ROE'])
```

| *unit* | L | M | P | Q | a | LR | PQ | ROE |
|---|---|---|---|---|---|---|---|---|
| price | 2,280.1 | 665.5 | 2,945.6 | 3,492.4 | 6,438 | 77.4% | 84.3% | 19.1% |
| price_pentagon(P) | 2,280.1 | 665.5 | 2,945.6 | 3,492.4 | 6,438 | 77.4% | 84.3% | 19.1% |
| price_pentagon(ROE) | 2,280.1 | 542.3 | 2,822.4 | 3,615.6 | 6,438 | 80.8% | 78.1% | 15.0% |

### Breakeven distortions from a premium

Evaluating the premium of 2,945.6 at the same assets reverses the pricing calculation: the `ph` row recovers the shape $a = 0.6$, to a solver error of $10^{-10}$. The `ccoc` row recovers the return of 19.1%, since `ccoc` at return $r$ prices to exactly the premium that earns $r$ on capital. The other rows are the shapes of the other families that give the same premium. Their indexes run from 0.160 for `ccoc` to 0.386 for `tvar`: the families differ in how much stress the same premium allows for, and the index is lowest for the families that load the tail.

```
ev = book.evaluate(P, p=0.995)
GT(ev.evaluation_df.loc['Book', ['gini', 'param_name', 'param', 'error', 'status']],
   formatters={'gini': '{x:.3f}', 'param': '{x:.4f}', 'error': '{x:.1e}'})
```

| *distortion* | gini | param_name | param | error | status |
|---|---|---|---|---|---|
| ccoc | 0.160 | r | 0.1906 | 0.0e+00 | ok |
| ph | 0.250 | a | 0.6000 | 1.4e-10 | ok |
| wang | 0.296 | lam | 0.5382 | 6.4e-08 | ok |
| dual | 0.344 | b | 2.0510 | -9.1e-13 | ok |
| tvar | 0.386 | p | 0.3857 | 7.4e-06 | ok |

Each breakeven distortion prices the book back to the premium. The check uses `Distortion.price` on the distribution of the book, at the same assets.

```
ser = book.density_df.p_total
for k, row in ev.evaluation_df.loc['Book'].iterrows():
    d = Distortion(k, **{row.param_name: row.param})
    print(f'{d.label:12s} {d.price(ser, a=ev.a).ask:,.2f}')
```

```
ccoc(0.191)  2,945.63
PH(0.6)      2,945.63
Wang(0.538)  2,945.63
dual(2.05)   2,945.63
TVaR(0.386)  2,945.63
```

The same equation is solved in three other places. The `tvar` breakeven level, 0.3857, is the threshold `p_star` of the `Bounds` class, which computes the envelope of every distortion that prices the book to the premium (see Using `Bounds` and Using `PricingBounds`). The method `calibrate_distortions(coc, p=)` solves the same equation with the premium set by a cost of capital. The method `PnL.evaluate` evaluates each step of a profit and loss walk, including one whose consideration is random, such as a swing-rated premium (see Using pnl and xpnl).

### Evaluation with and without assets

Without `p=` or `a=` the position is the premium against the unlimited loss, which has the tail beyond 6,438 to pay for, so the same premium allows for less stress. The `ph` breakeven rises from 0.600 to 0.618 and its index falls from 0.250 to 0.236. The panel `a = 6,438` is the one to compare with a calibration at the same assets.

```
both = pd.concat({'a = 6,438': ev.evaluation_df.loc['Book', ['gini', 'param']],
                  'unlimited': book.evaluate(P).evaluation_df.loc['Book', ['gini', 'param']]},
                 axis=1)
GT(both, float_format='.4f')
```

| | a = 6,438 | | unlimited | |
|---|---|---|---|---|
| ***distortion*** | **gini** | **param** | **gini** | **param** |
| ccoc | 0.1601 | 0.1906 | — | — |
| ph | 0.2500 | 0.6000 | 0.2358 | 0.6184 |
| wang | 0.2965 | 0.5382 | 0.2897 | 0.5253 |
| dual | 0.3445 | 2.0510 | 0.3410 | 2.0350 |
| tvar | 0.3857 | 0.3857 | 0.3827 | 0.3827 |

### The Gini index across premiums

A premium set by a loss ratio, $P = L/\text{LR}$, is evaluated below at loss ratios from 90% down to 70%. Every index rises as the loss ratio falls, and the families keep their order at every premium, from `ccoc` lowest to `tvar` highest. At a 75% loss ratio the indexes run from 0.183 to 0.432. The table puts premiums down and families across, with one index in each cell.

```
scan = {f'{lr:.0%}': book.evaluate(L / lr, p=0.995).evaluation_df.loc['Book', 'gini']
        for lr in [0.9, 0.85, 0.8, 0.75, 0.7]}
GT(pd.DataFrame(scan).T.rename_axis('LR'), float_format='.3f')
```

| *LR* | ccoc | ph | wang | dual | tvar |
|---|---|---|---|---|---|
| 90% | 0.061 | 0.104 | 0.121 | 0.138 | 0.153 |
| 85% | 0.097 | 0.160 | 0.187 | 0.215 | 0.242 |
| 80% | 0.137 | 0.218 | 0.258 | 0.299 | 0.336 |
| 75% | 0.183 | 0.280 | 0.333 | 0.388 | 0.432 |
| 70% | 0.235 | 0.345 | 0.412 | 0.481 | 0.529 |

## A premium below the expected loss

A premium below the expected loss, 2,280.1, has no breakeven, since every concave distortion prices the loss at or above its mean, and so above the premium. The method `evaluate` returns NaN with the reason in `status`, and issues one `DegenerateEvaluationWarning` for the whole panel.

```
with warnings.catch_warnings(record=True) as caught:
    warnings.simplefilter('always')
    low = book.evaluate(2200, p=0.995)
wprint(caught[0].message)
GT(low.evaluation_df.loc['Book', ['gini', 'param', 'status']])
```

```
no breakeven acceptability level for Book, reporting NaN (Book: E[M] = -80.0874
<= 0).
```

| *distortion* | gini | param | status |
|---|---|---|---|
| ccoc | — | — | E[M] = -80.0874 <= 0 |
| ph | — | — | E[M] = -80.0874 <= 0 |
| wang | — | — | E[M] = -80.0874 <= 0 |
| dual | — | — | E[M] = -80.0874 <= 0 |
| tvar | — | — | E[M] = -80.0874 <= 0 |

## Refused calls

Four calls are refused: `evaluate` on a book with no premium declared and none passed; `p=1` on a book with unbounded losses, where it does not determine an asset level; both `p=` and `a=`; and `price_pentagon` with two targets.

```
refuse(lambda: book.evaluate(),
       lambda: book.evaluate(P, p=1),
       lambda: book.evaluate(P, p=0.995, a=6438),
       lambda: book.price_pentagon(p=0.995, P=P, ROE=0.15))
```

```
Book has no premium to evaluate against: pass P=, or declare one in DecL (e.g.
"1000 premium at 0.7 lr").
evaluate: 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.
evaluate: pass at most one of p= or a=.
price_pentagon: pass exactly one pricing target (one of P, M, Q, LR, PQ, ROE);
got 2.
```